6.841 Advanced Complexity Theory	February 7, 2005
Lecture 2	
Lecturer: Madhu Sudan	*Scribe: Kyomin Jung*

1 Overview

In this lecture we will discuss some approaches and results concerning the problem $\mathbf{P} \neq \mathbf{NP}$, that use diagonalization arguments[1]. Baker-Gill-Solovay relativization shows a limitation of diagonalization method for showing $\mathbf{P} \neq \mathbf{NP}$. And Lander's theorem says about the hierarchy of intermediate language classes between **P** and **NP** under the assumption that $\mathbf{P} \neq \mathbf{NP}$. Then we will introduce the circuit complexity of a language, which we will discuss more in detail in the next lecture.

2 Diagonalization

Diagonalization argument, which was first used by Cantor when he showed that there is no one to one correspondence between $\mathbb{N}$ and $\mathbb{R}$, is an important tool when we show that for classes of languages C_1 and C_2 that are enumerable, C_1 is strictly contained within C_2. Let

$$C_1 =< L_1, L_2, L_3, \ldots >$$

where each languages in C_1 appears at least once in the above sequence. Let $x_1, x_2, x_3 \ldots$ be a sequence of all the finite binary strings. We can get this sequence because the set of all the finite binary strings is enumerable. Then each language L_i can be thought as an infinite string $s_{i1}s_{i2}s_{i3}\ldots$ where $s_{ij} = 1$ iff $x_j \in L_i$ and otherwise $s_{ij} = 0$. Now think of an infinite string $L' = \bar{s_{11}}\bar{s_{22}}\bar{s_{33}}\ldots$, where $\bar{s_{ii}}$ is the negation of s_{ii}. Then L' is not in C_1 because L' differs from L_i in their i^{th} bits. So if we show that L' is in C_2(this part differs according to the specific problem), we obtain that C_1 is strictly contained within C_2. This type of argument is called diagonilization and usually this kind of argument can be applied to obtain some results like Time(n^2) is strictly contained within Time(n^3).

3 Baker-Gill-Solovay Relativization

For a complexity class of languages, C, which is characterized by some limit on some resources(for example, **P**), we can think of a "relativized" class of languages with respect to a given language A, that is, machines in this class are allowed to make oracle calls to A for a unit cost.

Definition 1 $\mathbf{P}^A$*, the relativization of* **P** *with respect to* A*, is the set of all languages decidable by polynomial time machines with oracle calls to* A*.*

Definition 2 $\mathbf{NP}^A$*, the relativization of* **NP** *with respect to* A*, is the set of all languages decidable by polynomial time machines with oracle calls to* A*.*

If diagonalization produces a language L' in C_2 but not in C_1, then it can be seen that for every language A, C_1^A is strictly contained in C_2^A using L'. With this fact in mind, next theorem due to Baker-Gill-Solovay shows a limitation of diagonalization arguments for proving $\mathbf{P} \neq \mathbf{NP}$.

Theorem 3 *(Baker-Gill-Solovay) There exist oracles* A *and* B *such that*

$$\mathbf{P}^A = \mathbf{NP}^A$$

$$\mathbf{P}^B \neq \mathbf{NP}^B.$$

[1]See Lance Fortnow's survey paper on diagonalization as a reference

Here we will only prove the first part of the above theorem. To this end, take the language A to be a PSPACE-complete language, for example TQBF. Then

$$\mathbf{NP}^A \subseteq \text{NPSPACE} = \text{PSPACE} \subseteq \mathbf{P}^{\text{A}} \subseteq \mathbf{NP}^{\text{A}}.$$

So, from this results we can see that diagonilization argument cannot help to prove $\mathbf{P} \neq \mathbf{NP}$.

4 Ladner's Theorem

Some examples of **NP** problems that are not known to be in **P** or **NP**-complete are Graph isomorphism problem and Integer Factoring problem. Although it was not known whether Primality testing is in **P** or not until some years ago, recently it was proved that Primality testing is in **P**. Althought currently we do not know whether $\mathbf{P} \neq \mathbf{NP}$ or not, above problems are usually thought to be as candidates of languages that would be in between **P** and **NP**-complete. Now we will state and prove a result due to Lander that says that if $\mathbf{P} \neq \mathbf{NP}$, there must be some(in fact, infinitely many) intermediate language classes between **P** and **NP**.

Theorem 4 *(Lander) If $P \neq \mathbf{NP}$, then there exists $L \in \mathbf{NP}$ such that L is not **NP**-hard and L is not in **P**.*

More general version of Lander's theorem, which we will not deal in this lecture, says that

Theorem 5 *If $C_1 \leq_P C_2$ holds but $C_2 \leq_P C_1$ does not holds, then there exists C_3 and C_4 such that $C_1 \leq_P C_3 \leq_P C_2$ and $C_1 \leq_P C_4 \leq_P C_2$ hold but none of $C_2 \leq_P C_3$, $C_2 \leq_P C_4$, $C_3 \leq_P C_1$, $C_4 \leq_P C_1$, $C_3 \leq_P C_4$, $C_4 \leq_P C_3$ holds.*

Theorem 5 shows that if $\mathbf{P} \neq \mathbf{NP}$, then there must be infinitely many intermediate language classes between **P** and **NP** up to polynomial time reduction. Now to show the simpler version of the Lander's theorem, we will explicitly define a language L that is in **NP**, but not in **P** nor a **NP**-complete.

Let $f : \mathbb{N} \to \mathbb{N}$ be a polynomial time computable function which we will define later. Then we will define

$$L = \{n \mid n \in \text{SAT and } f(|n|) \text{ is even}\}.$$

Intuitively, when $f(n)$ is even $L(n)$ is copied from SAT(n)(here SAT can be replaced by any **NP**-complete language), and when $f(n)$ is odd $L(n)$ is copied from the language $000000\ldots$ that is in **P**.

We will define f to be an increasing function so that L is in **NP** but it is different from any of languages in **P**, and different from any of **NP**-complete languages. Let $M_1, M_2, M_3 \ldots$ be a sequence of al the polynomial time deterministic Turing Machines (allowing that some machines may appear more than once) so that $M_i(x)$ runs in time $|x|^i$. Similarly let $g_1, g_2, g_3 \ldots$ be a sequence of all the polynomial time computable reductions so that $g_i(x)$ runs in time $|x|^i$. Inductively, define $f(n)$ by

1. When $f(n-1) = 2j$ for some integer j. $f(n) = f(n-1) + 1$ if there exist $x \leq \log\log n$ such that $M_j(x) \neq L(x)$. Otherwise $f(n) = f(n-1)$. I.e., f stays at the same value until L is sure to be different from M_j at some sufficiently small x.

2. When $f(n-1) = 2j+1$ for some integer j. $f(n) = f(n-1)+1$ if there exist $x \leq \log\log n$ such that $\mathrm{SAT(x) \neq L(g_j(x))}$. Otherwise $f(n) = f(n-1)$. I.e., f stays at the same value until it becomes to be sure that SAT is not polynomial time reducible to L via reduction g_i.

Now we will show that f tends to infinity. Suppose that f is stuck at some even number as n goes to infinity. Then by the definition of L, L=SAT except finitely many inputs, and $L = M_j$ for some j. A contradiction to the assumption that SAT is not in **P**. Similarly, if f is stuck at some odd number, then L is in **P**, and SAT is reducible to L via some reduction g_j, saying that SAT is in **P**, again a contradiction to the assumption. So f tends to infinity as n goes to infinity.

Then by the definition of f, we obtain that L is in **NP**(note that here we need the fact that SAT is in **NP**). And because L is different from any of M_j, L is not in **P**. And because SAT is not polynomial time reducible to L, L is not **NP**-complete.

5 Circuit Complexity

Circuit is a model of computation having a finite number of input bits and finite number of output bits and having finite number of NOT, AND, OR gates in the middle of it. Formally, circuit can be thought as a directed, acyclic graph. Then a circuit consists of three types of nodes:

1. input nodes: input nodes have 0 In-degrees and (finitely many) Out-degrees. They are labeled with $x_1, x_2, x_3, \ldots x_n$.

2. computation nodes: There are three types of computation nodes. NOT nodes have 1 In-degrees and 1 Out-degrees, and AND and OR nodes have 2 In-degrees and 1 Out-degrees.

3. output nodes: output nodes have 1 In-degrees and 0 Out-degrees.

Now we will define the circuit complexity of a language L.

Definition 6 *For a family of circuits $\{C_n\}$ with $C_n : \{0,1\}^n \to \{0,1\}$, we say $\{C_n\}$ decides language L if for all $n \in \mathbb{N}$, $L \cup \{0,1\}^n = C_n^{-1}(1)$.*

Definition 7 *Define the size of C_n(denoted by $|C_n|$) to be the number of gates(nodes) in the circuit C_n.*

Note that sometimes one may define the size of C_n to be the number of wires(edges), but one measure is polynomial to the other. With these definitions we can define the circuit complexity of a given language L as usual.

Definition 8 *We say a language L has circuit complexity $f(\cdot)$ if there exists a circuit family $\{C_n\}$ deciding L and for all $n \in \mathbb{N}$, $|C_n| = f(n)$.*

Then why do we do the circuit complexity? That's because sometimes it provides some tools for comparing some complexity classes. We will discuss this more in detail in the next lecture.