6.841/18.405J: Advanced Complexity Theory Feb. 11, 2002

Lecture 2

Lecturer: Madhu Sudan *Scribe: Vahab S. Mirrokni*

In this lecture, we will see the diagonalization method as the most powerful method in complexity theory. We will discuss some results from this method e.g. Ladner's theorem; relativization, its relation with diagonalization method, and Baker-Gill-Solovay's theorem which implies that the diagonalization method does not help to prove NP $\neq$ P. Finally, we will have an introduction to alternation.

1 The Diagonalization Method

In order to find a relation between different complexity classes and to construct a better view of complexity, the only powerful known method is diagonalization. The main advantage of this method is to prove that some problems are not computable e.g. the undecidability of halting problem. Time and space hierarchy theorems are based on diagonalization. The other result from this method is the following: if P $\neq NP$, it is not possible to describe if a language is NPcomplete. Ladner's Theorem, which we will describe briefly, is another proof based on this method. It says that between any two different complexity classes there is an infinite hierarchy of complexity classes. For example, if P $\neq$ NP, it says that there are some problems that are niether in P nor in NPcomplete. Furthermore, there are infinite hierarchy of complexity classes between these P and NPcomplete. Ladner's theorem has an important effect on the philosophy of complexity theory. It means that in the case of P $\neq$ NP we can't classify all the problems in a finite number of complexity classes. Some examples of NP problems that are not known to be in P or NPcomplete are Primality testing, Graph isomorphism, and Factoring. Linear Programming was imagined to be in the middle of P and NPcomplete, but after some time it has been proved to be in P. Primality testing has a randomized polynomial time algorithm and it may be in P. On the other hand, it will be surprising if Factoring or Graph isomorphism is in P. More precisely, Ladner's theorem is the following:

Theorem 1 *(Ladner, 1973) If* P $\neq$ NP*, then for all* $k \geq 1$*, there are languages* $L_1, \ldots, L_k \in$ NP *such that*

$$L <_P L_1 <_P \cdots <_P L_k <_P L'$$

where $L \in$ P *and* L' *is* NPcomplete.

Proof Idea We just explain the intuition behind a weaker statement by taking $k = 1$: $\exists L \in$ NP such that $L <_P L_1 <_P L'$. Let $n_1 = 1$ and $n_i = 2^{n_{i-1}}$, and let $L_1 = L$ for strings of length $[n_{i-1}, n_i)$ for odd i, and $L_1 = L'$ for strings of length $[n_{i-1}, n_i)$ for even i. Then probably $L_1 \notin$ P, because it has a large part of L' which is NPcomplete. Also, probably L_1 is not NPcomplete, because otherwise if there is a polynomial reduction from L' to L, for a string s of length $n \in [n_{i-1}, n_i)$ for odd i in L', it can not be reduced to a string of greater length in L_1, because in that case the length of this string must be exponentially greater than n and it contradicts the fact that the reduction is polynomial. On the other hand, if s can be reduced to a string with smaller length, its length is exponentially smaller; thus, we can use an exponential time algorithm for the new problem, and It solves the first problem in L' in polynomial time of the input size, and it contradicts the fact that there is no polynomial time algorithm for deciding L'. The flaw of this argument is that we are assuming that the hardness of the NPcomplete problem (L') is uniformly distributed. Ladner's proof is based on picking a more careful choice of n_i's so that the above statements are not probable but certain. He used diagonalization method to find these n_i's. The idea is the following: list all polynomial reductions from L' as $R_1, R_2, \ldots$ and all polynomial Turing machines $M_1, M_2, \ldots$, and then start from string of length 0, and include all strings in L until it contradicts with M_1, then exclude all strings in L until it contradicts the reduction R_1, and so on. Actually, this argument was not complete. ■

So far, we have seen power of diagonalization. The big question here is "can it resolve P vs. NP?" Baker-Gill-Solovay theorem answers this question negatively. To go through this theorem, first, we discuss relativization, i.e. stuff with oracles.

2 Relativization

Definition 2 *Let C be a complexity class of languages decidable with TM with some upper bounds on some resources, and let A be an arbitrary language. Then C^A is the set of languages accepted by oracle TM with access to oracle A with the same or similar resource bound as a TM in C.*

However, in the above definition, the effect of A on a TM in C is not clear, and it is not an exact definition. It can be restated for the class P and NP more precisely.

Definition 3 *P^A is the set of all languages accepted by deterministic polynomial time oracle TM's with access to oracle for A.*

Definition 4 *NP^A is the set of all languages accepted by non-deterministic polynomial time oracle TM's with access to oracle for A.*

By the Turing machine with access to oracle for A, we mean a TM which has the ordinary input, work, and output tapes and also the oracle tape. This new tape gets an input and gives the output 0 or 1 that indicates ifthe input is in A or not.

Proposition 5 *If diagonalization shows $C_1 \not\subset C_2$, then for every A, $C_1^A \not\subset C_2^A$. Roughly speaking, $C_1 \not\subset C_2$ relativizes.*

Proof Idea If we know $C_1 \not\subset C_2$ using diagonalization, then it means that we can find $m \in C_1$ as the universal TM for TM's in C_2 i.e. it can simulate every TM $N \in C_2$, then we simulate N and negate the answer. Now, we can augment these machines into oracle machines and have similar results. In other words, we have TM $m^A \in C_1^A$ as the universal TM for TM's in C_2^A i.e. it can simulate every TM $N^A \in C_2^A$; then we simulate N^A and negate the answer. Thus, the similar argument shows that $C_1^A \not\subset C_2^A$. ■

Actually, there are many philosophies behind this proposition and these definitions. In the case of P and NP all above make sense, but somewhere else we can not use the same arguments.

Theorem 6 (Baker-Gill-Solovay) *There exist oracles A and B such that*

1. $\mathrm{P}^A = \mathrm{NP}^A$, *and*
2. $\mathrm{P}^B \neq \mathrm{NP}^B$.

Proof The first part is straightforward. The idea is to take some language that is sufficiently powerful e.g. PSPACE-complete. Let A be TQBF. Clearly $\mathrm{P}^A \subseteq \mathrm{NP}^A$. For the other direction, as TQBF is PSPACE-complete, we have

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

For the second part, the idea is that non-determinism gives us more powerful access to the oracle, allowing us to ask more questions than a deterministic TM can.

Definition 7 *For any language B, let*

$$L(B) = \{x | \exists w : |x| = |w| \text{ and } B(w) = 1\}$$

One can easily observe that $\forall B : L(B) \in \mathrm{NP}^B$, because we can guess w the same length as x, and see if $B(w) = 1$. We want to show that there is a B for which $L(B) \notin \mathrm{P}^B$. In order to do this for all TM's, first we can examine how this is done for one. Let $M^? \in P^?$. We know that $M^?$ is a polynomail time oracle TM, its running time is a polynomial $p(n)$ and we can find n such that $p(n) < 2^n$. We want to construct B so that M^B gives a wrong answer on some string x of length n, say $x = 0^n$, provided M^B runs in time $\leq 2^n - 1$. To do so, we start simulating $M^?$ for an input string x.

- The answer for any query of length n from $M^?$ is 0.
- At the end, some $w \in \{0,1\}^n$ remians unasked, becuase $M^?$ runs in $p(n)$ and it can ask at most $2^n - 1$ strings of length n.
- Now, if $M^?$ says "yes, $x \in L^B$", then set $B(w) = 0$ for every $w \in \{0,1\}^n$.
- if $M^?$ says "no, $x \notin L^B$", then $B(w) = 1$.

In either case, $L(M^B) \neq L(B)$, because $M^B(0^n) \neq (0^n \in L(B))$ (This part of the proof is from [1]). Now we need an oracle B such that for all polynomial time oracle TM's $M^?$, $L(M^B) \neq L(B)$. Suppose $M_1, M_2, \ldots$ are an enumeration of polynomial time oracle TM's. We construct a sequence $B_0 \subseteq B_1 \subseteq \ldots \subseteq B$, with lengths $n_1 < n_2 < \ldots$, such that

1. $M_i^{B_i}(0^{n_i}) \neq (0^{n_i} \in L(B_i))$.
2. $M_i^{B_i}(0^{n_i}) = M_i^B(0^{n_i})$.
3. $w \in L(B_i) : 0^{n_i} \in L(B_i) \Leftrightarrow 0^{n_i} \in L(B)$.

The construction algorithm is as follows:

1. First let $B_0 = \emptyset$.
2. For $i = 1, 2, 3, \ldots$ do
   (a) Choose n_i such that $\forall j < i$, $M_j^{B_{i-1}}$ does not ask about any strings of length n_i on input 0^{n_j}. In addition, $M_i^{B_{i-1}}$ should run for fewer than 2^{n_i} steps on input 0^{n_i}.
   (b) Simulate M_i with oracle B_{i-1} on input 0^{n_i}. (At this point, B_i has no strings of length n_i.) This will answer no to all queries of length n_i.
   (c) If $M_i^{B_{i-1}}$ accepts, set $B_i = B_{i-1}$. (So $M_i^{B_i}$ has accepted 0^{n_i}, but $0^{n_i} \notin L_{B_i}$).
   If $M_i^{B_{i-1}}$ rejects, then find x such that $|x| = n_i$ and $M_i^{B_{i-1}}$ on input 0^{n_i} did not ask x. Then set
   $$B_i = B_{i-1} \cup \{x\}.$$

Set $B = \bigcup_{i=0}^{\infty} B_i$. ■

Again, notice that the above proof is only true for the special case (P vs. NP), and in general cases there are other issues that we won't discuss in this lecture.

One can similarly prove that there exists an oracle B such that $\mathrm{NP}^B \neq \mathrm{coNP}^B$.

Now, consider this question: why do we use TQBF (a PSPACE-complete) problem for the first part of BGS theorem? Is an NP-complete problem sufficiently powerful? To answer this question, we should answer if $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}$? The answer is No. Actually we get a new complexity class that is not known to be equal to NP. To argue about this question, first we define the following problem.

Definition 8 MinDNF *is the language consisting of pairs* (ϕ, k)*, where* ϕ *is a DNF formula and* k *is an integer such that* $\exists$ *a DNF formula* ψ *such that* $|\psi| \leq k$ *and* ψ *is equivalent to* ϕ.

Proposition 9 MinDNF *is in* $\mathrm{NP}^{\mathrm{NP}}$.

Proof We can construct a non-deterministic oracle TM using SAT oracle to solve MinDNF.

- First guess ψ of length $\leq k$.
- Ask SAT oracle if there exists an assignment x such that $\psi(x) \neq \phi(x)$.

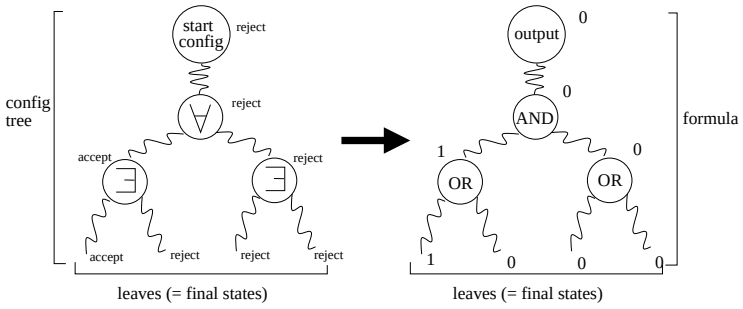

Figure 1: The computation tree of an ATM and corresponding circuit configuration

- Accept if oracle says No, otherwise reject.

This TM decides the MinDNF problem, and from the existence of this oracle TM, we can conclude $MinDNF \in \mathrm{NP}^{\mathrm{NP}}$. ■

Similarly, one can prove that co-NP$\subseteq \mathrm{NP}^{\mathrm{NP}}$. Notice that MinDNF has one $\exists$ quantifier. Let's investigate the case that there are more quantifiers.

3 Alternation

In the definition of non-deterministic TM, one way to think about it is a normal TM which has "existential states" ($\exists$). The converse of TM with Existential states is one with Universal ($\forall$) states. At each node, the machine can be thought of as spinning off two parallel actions and taking both paths at the same time. The machine accepts if all of its branches end in the accept state.

We can now consider machines with both Existential and Universal states. They are called Alternating Turing Machines (ATM).

Definition 10 *The Alternating Turing Machine (ATM) is a TM with two special states $\exists$ and $\forall$. Computation accepts after entering*

- *$\exists$, if at least one of the outgoing edges lead to accept.*
- *$\forall$, if ALL of outgoing edges lead to accept.*

Figure 3 illustrates one ATM.

Similar to TM's, main resources for the ATM's are TIME and SPACE, but the other resource for them is Alternation i.e. # times we alternate from $\exists$ to $\forall$ and vice-versa. More precisely, for machine M and input x, TIME is the deepest path you can find in the tree. SPACE is maximum space on paths. Alternation is maximum # of alternations along the paths.

According to these resources, we can define general complexity classes as follows:

Definition 11

$\mathrm{ATSP}[\mathrm{a}, \mathrm{t}, \mathrm{s}]$ = *class of all languages decided by ATM with* a(n) *alternation,* t(n) *time, and* s(n) *space.*

In particular, ATIME $(t) = \cup_{\forall a,s} ATSP[a,t,s]$ *and* $ASPACE(s) = \cup_{\forall a,s} ATSP[a,t,s]$.

We have the following facts:

- ATIME(poly)=PSPACE

- ASPACE(poly)=EXPTIME

The interesting point is that alternation increases the computation power and also change TIME and SPACE in these cases.

Definition 12

$$\mathrm{NP} = \mathrm{ATSP}[1, \mathrm{poly}, \mathrm{poly}], \ \textit{with condition starting } \exists \textit{ state}$$

$$\mathrm{NP}^{\mathrm{NP}} = \Sigma_2^{\mathrm{P}} = \mathrm{ATSP}[2, \mathrm{poly}, \mathrm{poly}], \ \textit{with condition starting } \exists \textit{ state}$$

$$\Sigma_i^{\mathrm{P}} = \mathrm{ATSP}[\mathrm{i}, \mathrm{poly}, \mathrm{poly}], \ \textit{with condition starting } \exists \textit{ state}$$

and similarly,

$$\Pi_i^{\mathrm{P}} = \mathrm{ATSP}[\mathrm{i}, \mathrm{poly}, \mathrm{poly}], \ \textit{with condition starting } \forall \textit{ state}$$

In the next lecture, we will study more about relations between these complexity classes.

References

[1] Lecturer: Dan Spielman, Scribe: Edward Early, *6.841/18.405J Advanced Complexity Theory: Lecture 6*, **Feb. 27, 2001**.