CS 810: Introduction to Complexity Theory 02/13/2003

Lecture 8: Relativizations. Baker-Gill-Solovay Theorem

Instructor: Jin-Yi Cai Scribe: Matthew Lee, Yingchao Liu, Uchechukwu Okpara

In this lecture, we construct two oracles A and B such that $P^A = NP^A$, whereas, $P^B \neq NP^B$. This suggests that a relativizable proof technique may not be useful in resolving the NP vs P question.

1 Relativization

Many known proofs in complexity theory relativize. That means, the statement of theorem is true with respect to arbitrary oracles. For example, consider the space hierarchy theorem. For any "nice" functions, $S(n)$ and $T(n)$ such that $S(n) = o(T(n))$, we have DSPACE$(S(n)) \neq$ DSPACE$(T(n))$. This is also true with respect to arbitrary oracles: for any oracle A, DSPACE$^A(S(n)) \neq$ DSPACE$^A(T(n))$. The proof technique used to prove the hierarchy theorem, namely diagonalization, relativizes. The goal of this lecture is to show that such a proof technique is unlikely to resolve the NP vs P question. We shall prove the Baker-Gill-Solovay theorem, which presents oracles A and B such that $P^A = NP^A$ and $P^B \neq NP^B$.

2 Baker-Gill-Solovay Theorem (BGS)

Theorem 1. *(a) There exists a recursive set A such that $P^A = NP^A$*
(b)There exists a recursive set B such that $P^B \neq NP^B$

Proof. (a) Take QBF (Quantified Boolean Formula) as the set A. First note that, $P^{QBF} = PSPACE$. As QBF is complete for PSPACE, $PSPACE \subseteq P^{QBF}$. On the other hand, as $QBF \in PSPACE$, we can resolve queries to QBF in PSPACE. Thus, $P^{QBF} \subseteq PSPACE$.

Next, we claim that $PSPACE = NP^{QBF}$. Clearly, $PSPACE \subseteq P^{QBF} \subseteq NP^{QBF}$. For the other direction, upon input x, we do DFS on the NTM's computation tree. In this simulation, we only need to store the information of one path of the tree any time. The depth of the tree is polynomial. Each configuration or node takes only polynomial space to store. Thus, storage space required is polynomial. Answering QBF queries takes only polynomial space. Thus, the simulation can be carried out in polynomial space. So, $NP^{QBF} \subseteq PSPACE$.

At this point, we have $P^{QBF} = PSPACE = NP^{QBF}$.

(b) For any set B, define a set

$$L_B = \{1^n \| \exists x, |x| = n, x \in B\}$$

We'll show that $\forall B$, $L_B \in NP^B$, and $\exists B$, $L_B \notin P^B$.

It's easy to see that $\forall B$, $L_B \in NP^B$, because for any input 1^n we can guess a string x of length n and check whether x is in B by making a query to oracle B. So $\forall B$, $L_B \in NP^B$.

We now construct a set B such that $L_B \notin P^B$. The basic intuition is that any deterministic machine that decides B, given 1^n as input, has to find out whether there is a string of length n in B. This search process needs to check 2^n such strings in polynomial time, while no deterministic machine can ask that many queries to oracle B.

Now we give a rigorous proof. Notice that the class of all deterministic polynomial time oracle TMs are enumerable. We represent them as N_1, N_2,...

The idea is a diagonalized construction. We construct B in stages. Initially B is empty. At each stage i, we add some strings to B such that N_i^B does not decide L_B. It's clear that if we accomplish this task, then $L_B \notin L(N_i^B)$, for any N_i. Thus, $L_B \notin P^B$.

Now we describe stage i. We assume that by the end of stage $i-1$, we have already defined integer n_{i-1} and a finite set $B_{i-1} \subseteq \{x \in \{0,1\}^* : |x| \leq n_{i-1}\}$. We add a "clock" of polynomial size $P_i(n)$, which is the polynomial that bounds the runtime of machine N_i. Then choose the least integer n such that $2^n > P_i(n)$ and $n > n_{i-1}$. Set $n = n_i$, and $x_i = 1^{n_i}$. Simulate N_i on x_i. When N_i asks whether a string y is in the oracle, we simulate with answer YES if $y \in B_{i-1}$, NO otherwise. **Here comes the diagonalization**: There are two possibilities. When N_i halts, if it accepts x_i, we let $B_i = B_{i-1}$ (and so B_i has no element with length n_i). If it rejects x_i, we pick a string y' of length n_i, which has not been queried in the computation of $N_i(x_i)$ and let $B_i = B_{i-1} \cup \{y'\}$. Since there are 2^{n_i} possible strings of length n_i, and N_i can query at most $P_i(n) < 2^{n_i}$ times, so there always exists such string y'. Finally, if N_i had asked a query of length longer than n_i, we set n_i to length of longest such query. This is just to make sure that, we answer all queries from all the machines in a consistent manner.

In the above algorithm, we always guarantee $B_i \subseteq \{x \in \{0,1\}^* : |x| \leq n_i\}$. Let $B = \cup_{i=1}^{\infty} B_i$. We can no machine N_i decides L_B correctly. The reason is,$\forall i, N_i^B(x_i)$ accepts iff the simulation of $N_i^{B_{i-1}}$ on x_i in stage i accepts. This is true because we restrict $n_{i+1} > 2^{n_i} > P_i(n_i)$ and so we never add any string to B of length $\leq P_i(n_i)$ in later stages. In addition, whenever we add a string y' to B_i in stage i, we have guaranteed that the computation of $N_i^{B_{i-1}}(x_i)$ never queries y'. Thus, the queries made by N_i to B has the same membership in B_{i-1} as in B. That is to say, in either case, we have made sure:

$$x_i \in L_B \iff N_i^B(x_i) \text{ rejects} \tag{1}$$

Therefore we have found out such a recursive set B for which $P^B \neq NP^B$. □

3 Conclusion

In BGS theorem, part (a) suggests that it would be difficult to prove using relativization that $P \neq NP$, while part (b) suggests it would be difficult to prove $P = NP$ using relativization. So it strongly suggests that all currently available techniques will not suffice for proving that $P = NP$ or that $P \neq NP$.