18.405J/6.841J Advanced Complexity Theory	February 26, 1998
Lecture 6	
Lecturer: Dan Spielman	*Scribe: Matthew Levine*

1 Diagonalization and NP

The main result of this lecture is the following theorem:

Theorem 1 (Baker-Gill-Solovay) *There are oracles A and B such that*
a) $P^A = NP^A$ *and*
b) $P^B \neq NP^B$

This theorem is interesting, because it rules out many approaches for trying to resolve the question of whether $P = NP$ or not. In particular, many proofs that two complexity classes are different use a diagonalization argument, where one constructs a member of one class that is explicitly different from each object of the other class. For example, it is possible to prove that the halting problem is not computable in this way. People thought this technique might work to prove $P \neq NP$, but diagonalization arguments tend to hold relative to an oracle, so the first half of the above theorem shows that this is not possible. Likewise, it is unlikely that we can show $P = NP$ by an argument that a P machine can simulate an NP machine, because that too would probably hold relative to an oracle, contradicting the second half of the theorem. The bottom line is that any proof regarding the $P = NP$ question must *not* hold relative to an oracle, or it will contradict the theorem. This suggests that resolving the P versus NP question will be difficult.

We now give the proof.

1.1 Part a

To show the first part of the theorem, we need to construct an oracle A such that $P^A = NP^A$, that is, an oracle where nondeterminism won't add any power. Since $NPSPACE = PSPACE$, an oracle for a $PSPACE$-complete problem seems like a good idea.

Recall that the language of true quantified boolean formulas ($TQBF$) is $PSPACE$-complete under polynomial time mapping reductions. (A quantified boolean formula is an expression of the form $\exists x_1 \forall x_2 \forall x_3 \exists x_4 \ldots \text{boolean formula}(x_1, x_2, \ldots, x_n)$) Let A be an oracle for $TQBF$.

We have that $NP^A \subseteq NPSPACE$, because an $NPSPACE$ machine can easily simulate an NP machine, and can simulate the $TQBF$ oracle, without using nondeterminism. We also have $PSPACE \subseteq P^A$, because $TQBF$ is a complete problem for $PSPACE$. Since $NPSPACE = PSPACE$, this gives $NP^A \subseteq P^A$. It is obvious that $P^A \subseteq NP^A$, so in fact $P^A = NP^A$.

1.2 Part b

To show this part of the theorem, we want to construct an oracle B and a language L such that $L \in NP^B$, but $L \notin P^B$. We start by defining a language associated with an oracle B:

$$L(B) = \{x | \exists y \in B \text{ s.t. } |y| = |x|\}$$

So $L(B)$ contains all strings of the same length as some element of B. (Think of B as a set of the strings that it accepts.) Observe that $L(B) \in NP^B$, because on input x an NP machine can guess a $y \in B$ such that $|x| = |y|$ and then check with one call to B.

Now we want to construct B so that no deterministic polynomial time machine can accept $L(B)$. The idea is that on an input of length n there are 2^n possible strings that could be in B, and the deterministic machine has no way to tell which except by trying them all.

We start by showing that for any fixed deterministic polynomial time oracle Turing machine $M^?$, we can construct a B such that M^B cannot accept $L(B)$. We will then show how to construct such a B for all such $M^?$ at once.

By the definition of running in polynomial time, there exists an n such that M runs in time less than 2^n. Since there are 2^n y's of length n, on input 0^n $M^\emptyset$ doesn't query B about some y. Therefore, if $M^\emptyset$ accepts 0^n we can set $B = \emptyset$, and M^B will make a mistake testing membership in $L(B)$. Likewise, if $M^\emptyset$ rejects 0^n we can set $B = \{y\}$, and M^B will make a mistake testing membership in $L(B)$.

We extend this by a diagonalization argument: we will enumerate the deterministic polynomial time Turing machines and arrange to trick each one for a different length input. This will succeed in tricking them all.

There exists an enumeration of polynomial time oracle Turing machines. Consider the enumeration $M_1^?$, $M_2^?$, $M_3^?$, ...

The following "algorithm", gives the construction of B:

1. Let $B_0 = \emptyset$, $n = 0$
2. for $i = 1, 2, 3, \ldots$ do
   (a) choose $n_i > n$, such that $M_i^{B_{i-1}}$ on input 0^{n_i} asks fewer than 2^{n_i} questions.
   (b) If $M_i^{B_{i-1}}$ accepts 0^{n_i} then let $B_i = B_{i-1}$ (don't include any new strings)
   Else (it rejects) let $B_i = B_{i-1} \cup \{x \text{ s.t. } |x| = n_i \text{ and } x \text{ was not asked about by } M_i^{B_{i-1}}\}$.
   (c) set $n = \max_i\{\text{length of longest query asked by } M_i^{B_{i-1}}, \text{ on input } 0^{n_i}\}$.
3. set $B = \bigcup_{i=0}^{\infty} B_i$

It is clear that $L(B) \notin P^B$, because each M_i^B gives the wrong answer on 0^{n_i}. Therefore $P^B \neq NP^B$.