The Baker-Gill-Solovay Theorem

This proof was presented in class on February 4, 2016.

We use the definitions of *nondeterministic Turing Machines* and *oracle Turing Machines* that are given in the Arora-Barak textbook; see Definitions 2.5 and 3.4 and the text immediately preceding Definition 2.5. Recall that, for any oracle O, the class of languages recognizable by deterministic (respectively, nondeterministic), polynomial-time oracle TMs that are given access to oracle O is denoted P^O (respectively, NP^O). The Baker-Gill-Solovay Theorem tells us that there are oracles A and B such that $P^A = NP^A$ and $P^B \neq NP^B$.

We show first that there is a set A such that $P^A = NP^A$. One such set is EXPCOM, which is defined as $\{(M, x, 1^n)$ such that M accepts x in time at most $2^n\}$. It is fairly easy to see that $P^{\text{EXPCOM}} = NP^{\text{EXPCOM}} = EXP$, where EXP is the union, over all nonnegative integers c, of DTIME(2^{n^c}). By definition, $P^{\text{EXPCOM}} \subseteq NP^{\text{EXPCOM}}$, because every deterministic polynomial-time oracle TM is also a nondeterministic polynomial-time oracle TM. To see that $EXP \subseteq P^{\text{EXPCOM}}$, let S be an arbitrary set in EXP, and let M be a deterministic TM that runs in time 2^{n^c} and recognizes S. A deterministic polynomial-time oracle TM with access to EXPCOM can determine whether x is in S with a single query to an EXPCOM oracle: If $x \in \{0,1\}^t$, the query is $(M, x, 1^{t^c})$; note that the length of this query is polynomial in t, as required. Finally, note that $NP^{\text{EXPCOM}} \subseteq EXP$, because the computation of a nondeterministic polynomial-time machine with access to an EXPCOM oracle can be simulated deterministically in singly exponential time. Let W be a nondeterministic machine that runs in time $p_1(n)$, where p_1 is a polynomial, and has access to an EXPCOM oracle. Consider a deterministic machine W' that simulates W without accessing the oracle; that is, if at some point W makes the EXPCOM query $(M, y, 1^t)$, W' instead computes the oracle answer by simulating M on y for 2^t steps. Note that, on input x of length n, any strings y or 1^t that W computes when forming oracle queries must be of length polynomial in n. Thus, there is a polynomial p_2 such that the time that W' requires to simulate an oracle query by W is bounded above by $2^{p_2(n)}$. The entire tree of possible computations by W on input x has size $2^{p_1(n)}$. W' can explore the entire tree (and accept if and only if it finds at least one accepting path) in time $2^{p_1(n)} \cdot 2^{p_2(n)}$, because the tree is of size $2^{p_1(n)}$, and the most time-consuming thing W' will ever have to do at any node of the tree is to compute the answer to an oracle query. Since the total running time of this deterministic simulation is singly exponential (specifically, bounded above by $2^{(p_1+p_2)(n)}$), the language $L(W^{\text{EXPCOM}})$ is in EXP. Since W is an arbitrary NP machine, this means that $NP^{\text{EXPCOM}} \subseteq EXP$.

Why did we not define EXPCOM as $\{(M, x, n)$ such that M accepts x in time at most $2^n\}$? Had we done so, we would still have that $P^{\text{EXPCOM}} \subseteq NP^{\text{EXPCOM}}$ and that $EXP \subseteq P^{EXPCOM}$. However, the foregoing proof that $NP^{\text{EXPCOM}} \subseteq EXP$ would not go through. We have an input x of length n for which we are trying to determine membership in $L(W^{\text{EXPCOM}})$. In time $p_1(n)$, W could construct a query (M, y, t), where $t = 2^{p_1(n)}$, because writing down such a t requires simply writing down one 1 followed by $p_1(n)$ zeroes. Simulating M on input y for 2^t steps would, in this case, require time $2^{2^{p_1(n)}}$, which is doubly exponential in the length of $|x|$, not singly exponential in $|x|$ as we need it to be for this proof.

We now turn to the proof that there is a set B such that $P^B \neq NP^B$.

For any set B, let $U_B = \{1^n \text{ such that } \exists x \text{ of length } n \text{ in } B\}$.

Then, for any B, $U_B \in NP^B$, because an NP machine can just guess a string of length n, ask the oracle whether it is in B, and accept 1^n if an only if the answer is yes.

We will construct a B such that $U_B \notin P^B$. Let M_i be the oracle TM given by the binary representation of i. Define the set B in an infinite number of "stages," one for each non-negative integer i, so that it has the property that, for all i, M_i^B does not recognize U_B in time $\frac{2^n}{10}$. Note that this is *stronger* than $U_B \notin P^B$.

Stage 0: $B \leftarrow \emptyset$.

Assume that we've done stages 0 through $i-1$ of the construction. At this point, membership or nonmembership in B has been fixed for some finite number of strings; say that the longest one of them has length $n-1$. This value of n will be used in stage i of the construction.

Stage i: Run machine M_i on input 1^n for $\frac{2^n}{10}$ steps. When M_i needs the answer to an oracle query q, it does the following:

* If it has been determined in an earlier stage whether q is in B, then answer consistently with the earlier decision.

** If it has not been determined in an earlier stage whether q is in B, then answer NO (whether or not $|q| < n$) and fix q as a non-member of B.

Finish off Stage i of the definition of B (by doing (1) and (2) below) in such a way that, if M_i halts within $\frac{2^n}{10}$ steps, it makes the wrong decision. Note that * and ** above ensure that, if M_i halts within $\frac{2^n}{10}$ steps on 1^n, it can actually make an ACC/REJ decision, because they ensure that it has answers to all of the oracle queries that it makes.

1. If it accepts, then define $B \cap \{0,1\}^n$ to be $\emptyset$. Note that this is consistent with * and **, because, up until this point, the answer to all oracle queries q such that $|q| \geq n$ has been NO.

2. If it does not accept (*i.e.*, if it rejects or simply does not halt within $\frac{2^n}{10}$ steps), then choose a string q of length n whose membership in B has not yet been determined and put it in B. Note that such a q must exist, because no oracle queries of length n were made before stage i, and at most $\frac{2^n}{10}$ such queries can be made in stage i; thus there are at least $\frac{9 \cdot 2^n}{10}$ strings of length n for which membership in B has not yet been determined.

In case 1, M_i accepts 1^n, but there is no string of length n in B. In case 2, M_i rejects 1^n, but q is a string in B that has length n. Thus U_B is not correctly decided by M_i^B in time $\frac{2^n}{10}$. This holds for all i, and thus U_B is not recognized by *any* deterministic oracle machine with access to oracle B in time $\frac{2^n}{10}$. *A fortiori*, $U_B \notin P^B$.

Note that B has not yet been defined fully; there are strings q that have never been considered in *, **, (1), or (2) in any stage i, and thus we have not yet decided whether they are in B. In fact, we put any such q into B or leave it out of B, and the proof goes through. The point is simply that, for B to be completely well defined, a choice must be made about every string in $\{0,1\}^*$.

One interpretation of this result is that A is a very powerful oracle: If the class of deterministic, polynomial-time machines is given access to A, then it can gain no additional power by obtaining access to nondeterminism. Under the same interpretation, B is a weaker oracle: With access to B, deterministic, polynomial-time machines still cannot do everything that nondeterministic, polynomial-time machines can do.