Oracles, Hierarchies, and Monotone Circuits.

CS254

Chris Pollett

Nov. 29, 2006.

Outline

- Oracle Machines
- Baker-Gill-Solovay
- Monotone Circuits

Oracle Machines

- We now consider TM which have access to a black box called an oracle.
- It turns out many of the proofs about relationships between complexity classes carry over to the oracle setting.
- So oracle results give us bounds on what can happen for the usual complexity classes without oracles.
- The oracle setting also tells us something about the strength of reductions.
- Namely, one might ask: Can an “NP-reduction” be more powerful that a “P-reduction”?

Definition

A **Turing Machine $M^?$ with oracle** is a multi-tape DTM (a similar definition works for NTMs) with a special query tape. It also has three distinguished states $q_?$, q_{yes}, q_{no}. We feed into the "?" slot of $M^?$ an oracle language $A \subseteq \Sigma^*$ to get a machine M^A. On input x, M^A computes as normal unless it enters the state $q_?$, in which case if y is the contents of the query tape then the next state will be q_{yes} if y is in A and will be q_{no} if y is not in A. The computation keeps going until a halt state is reached.

- M^A might enter the query state $q_?$ several times during its computation, so might ask for several different strings if they belong to A.
- Given a DTM or NTM space or time bounded complexity class C, let C^A denote the class of languages one gets by allowing the machines in C to be oracle machines with access to A. That is, P^A is the class of languages recognized in p-time by DTMs M^A.

Baker-Gill-Solovay

Thm. There are oracle sets A, B such that $P^A = NP^A$ and $P^B \neq NP^B$.

Proof. From the homework we know there is a PSPACE-complete language A. For this language we have:

PSPACE $\subseteq P^A \subseteq NP^A \subseteq$ NPSPACE $\subseteq$ PSPACE.

The construction for B is a little more involved. Let L be the following language:

$L = \{ 0^n \mid$ There is an x in B with $|x| = n\}$.

This language is in NP^B. We guess an x of length n and check if it is in B using the oracle. We will show that we can choose B so that this language is not in P^B.

BGS proof cont'd

- To build B we enumerate oracle DTMs, $M^?_1$, $M^?_2$,.. by listing out strings in lex order and then checking if they are oracle DTMs.
- We define B in stages ($B = \cup_i B_i$) based on which oracle DTM we have just enumerated.
- Our construction has the property that B_i contains all strings in B of length $\leq i$.
- B_0 is the empty set.
- Assume we have constructed B_{i-1} and have just written $M^?_i$ on the tape where we are doing the enumeration. We then simulate $M^B_i(0^i)$ for $i^{\log i}$ steps.
- Notice this is more than polynomially many steps.
- Since we haven't completed B yet how do we answer oracle queries? …

Yet More proof

- Answering queries "y in B?":
  - If |y| < i then answer according to B_{i-1}.
  - If |y| ≥ i then answer "no" and make sure to remember y in some "no" set stored on another string, so that we never add y to B.
- Suppose after $i^{\log i}$ steps M^B_i rejects. Then we pick some string of length i that was never queried by any M^B_j for j≤i.
- This is possible since

  $$\sum^i_{j=1} j^{\log j} \leq \sum^i_{j=1} i^{\log i} = i*2^{\log^{\wedge}2\, i} < 2^i.$$
- On the other hand, if M^B_i accepts, we set $B_i = B_{i-1}$, so that there are no strings of length i in B and so L does not contain 0^i.
- The last case is that M^B_i did not halt within $i^{\log i}$ steps. This might happen even if M^B_i is p-time if the cofficients in the polynomial bounding p(i) its runtime are such that $i^{\log i} \leq p(i)$. Again, we set $B_i = B_{i-1}$. We know that an equivalent machine to M^B_i will evenetually be listed out with large enough index I so that $I^{\log I} \geq p(I)$ in which case the first two cases will ensure that M^B_i 's is not L.

Monotone Circuits

- We earlier saw that if we could prove super-polynomial lower bounds on circuit size for some NP language we would know that P/poly≠NP and hence P≠NP.
- Such lower bound results are hard to obtain.
- We also know that at least as far as the CVP goes monotone circuits are also P-complete, so in some sense are at least as hard as nonmonotone circuits.
- Maybe, it is easier to prove circuit lower bounds for monotone circuits?
- Is it possible to express any NP-complete problem so that it could even be solved by monotone circuits?

CLIQUE$_{n,k}$

- We have seen that whether a graph has a clique of size k is NP-complete. Call the n node version of this problem CLIQUE$_{n,k}$.
- One can also build monotone exponential size circuits to test if a graph G=(V,E) of n nodes has a clique of size k:
  - The inputs g_{ij} correspond to the entries of the adjacency matrix for G.
  - There are $\binom{n}{2}$ gates such g_{ij} and a given one is true iff there is an edge from i to j in G.
  - For each subset S of V, with |S|=k, we have an AND of the O(k^2) many gates which correspond to a clique on this set of vertices.
  - We then have a big OR over the $\binom{n}{k}$ many different subsets S.
  - This circuit thus has size $O(k^2\binom{n}{k})$.

Razborov's Theorem

Thm. There is a constant c such that for large enough n all monotone circuits for $CLIQUE_{n,k}$ with $k = (n)^{1/4}$ have size at lest $2^{c(n)^{\wedge}\{1/8\}}$.

Proof. We will give the proof next day.