Chapter 12

Miscellaneous Results

12.1 Relativized Separation of NP and P

We construct an oracle relative to which NP $\neq$ P. And also another oracle relative to which NP = P.

THEOREM 12.1 (BAKER, GILL, SOLOVAY) *There are computable languages A and B such that*

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

Proof (Part I): This is easy to show. Choose A to be any PSPACE-Complete language. Then,

$$\mathrm{NP}^A \subseteq \mathrm{PSPACE}^A \subseteq \mathrm{PSPACE} \subseteq \mathrm{P}^A$$

The first inclusion is because, we can simulate an NP machine in PSPACE by running through all the possible guesses of the NP machine. The second inclusion is because A is in PSPACE, and hence we can resolve queries to A in PSPACE without asking the oracle. As A is PSPACE-Hard, any PSPACE language can be reduced to A. So the last inclusion holds.

Proof (Part II): For any language B, let T_B be the language

$$T_B = \{1^n | \exists x, |x| = n, x \in L\}$$

We first show that for any B, $T_B \in \mathrm{NP}^B$. Given 1^n as input, simply guess a string x of length n, then, use the oracle to verify whether $x \in B$. If so accept, else reject.

We construct a language B such that $T_B \notin P^B$. The basic idea is simple. Intuitively, any machine that decides T_B, given 1^n as input, has to find out whether there is some string of

length n in B. But there are 2^n such strings. No polynomial machine, can ask that many queries to oracle B.

Let $N_1, N_2, \ldots$ be an enumeration of all deterministic polynomial time Turing machines. Let the running time of N_i be bounded by n^i. Such an enumeration can be achieved by, first enumerating all deterministic TMs and then adding a "clock" of size n^i to the ith TM. We shall diagonalize over all these N_i. We construct B in stages. We can think of B being empty initially, and we add some strings to it in each stage. In the kth stage, we make sure that, N_k does not decide T_B.

We describe the kth stage. We heave already simulated $N_1, N_2, \ldots N_{k-1}$ in previous $(k-1)$ stages and would have answered many oracle queries. Let n_0 be the maximum length of any query asked by any of these $k-1$ machines in our simulations. By the end of stage $k-1$, we would have constructed B up to some length n_0. At the kth stage, we make sure that N_k does not decide T_B. Now, choose n, such that $n > n_0$ and $2^n > n^k$ and simulate $N_k(1^n)$. The catch is in the second condition. N_k is a machine with time bound n^k. It can ask at most n^k queries. If it asks a query of length smaller than n_0, we already know the answer-(we have already constructed B up to length n_0). If it asks any query of length more than n_0, we simply answer "no". (That is, keep those strings out of B). Now it's time to diagnolize. At the end of simulation, suppose N_k accepts 1^n. We keep all strings of length n out of B. Suppose N_k rejects 1^n. It could have asked at most n^k queries of length n (all of which we answered "no"). As $2^n > n^k$, there exists at least one string y of length n that was not a query of N_k. We add this string to B. In either case, N_k has failed to decide 1^n correctly.

A formal algorithm to enumerate B is as follows. Think of B being empty initially and we add strings to it in stages. At any stage k, Q_k represents the set of queries asked by N_k in that stage. And n_0 is set to 0 initially. Run the following procedure for $k = 1, 2 \ldots$.

1. Choose an n, with $n > n_0$ and $2^n > n^k$. Set Q_k to be empty set.
2. Simulate $N_k(1^n)$ for n^k steps. If q is a query by N_k,
   - Add q to Q_k.
   - If $|q| \leq n_0$ then answer $B(x)$ else answer "no".
3. At the end of simulation:-
   - If n_k rejects 1^n:– choose a string y of length n not in Q_k and add it to B.
   - If n_k accepts 1^n:– do nothing.
4. Let q be the longest string in Q_k. Set $n_0 \longleftarrow \max\{n_0, |x|\}$.

Bibliography

[1] D. Du and K. Ko. *Theory of Computational Complexity*. John Wiley and Sons, 2000.

[2] M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[4] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.