COS 487: Theory of Computation **Fall 2000**

Handout 3: November 27

Sanjeev Arora

3.1 TIME$(n) \neq$ TIME(n^2)

We use diagonalization. Suppose $M_1, M_2, M_3, \ldots$ is an effective enumeration of all Turing Machines. Consider the following Turing Machine, D:

On input x, if $x = 0^j 1^k$ for some j, k then construct M_k and simulate it on x for $|x|^{1.5}$ steps. If M_k halts and accepts, reject. If M_k halts and rejects, accept. In every other situation (for example if M_k does not halt), accept.

This machine runs in time at most $2n^2$. Specifically, it has to maintain a timer that keeps tracks of the number of steps in the simulation of M_k. Maintaining this counter introduces an overhead so the running time of the modified machine will be $O(n^{1.5} \log n) = o(n^2)$.

We say that D *evades* M_k if there is an input y on which D and M_k give different answers. Note that if the running time of M_k is at most $cn + d$, then D answers differently from M_k on every input of the form $0^j 1^k$, where $j > ck + d$. We conclude that D evades every TM M_k that runs in linear time. Hence the language accepted by D is not in TIME(n) but as we saw it is in TIME(n^2).

3.2 NTIME$(n) \neq$ NTIME(n^2)

The above technique does not apply directly. A nondeterminisitic machine that runs in $O(n)$ time may have $2^{O(n)}$ branches in its computation. It is unclear how in $O(n^2)$ time we determine whether or not it accepts and then flip this answer.

We use *lazy* diagonalization instead. Let $M_1, M_2, M_3, \ldots$ be an enumeration of all NDTMs that run in time $O(n)$.

For any integers i, j, let $f(i, j)$ be $2^{2^{2^{i} 3^{j}}}$ Note that this imposes an ordering on the set of all pairs of integers.

We construct a new Turing Machine D_1.

On input x, if $x = 1^n$, then compute the largest i, j and the smallest k, l such that

$$f(i, j) \leq n < f(k, l).$$

Note that $f(k,l) > 2^{(f(i,j))^2}$. If $n > 2^{(f(i,j))^2}$, accept 1^n iff M_i rejects $1^{f(i,j)}$. (Note that this requires going through all possible nondeterministic branches of M_i.) Otherwise simulate M_i on input 1^{n+1} using nondeterminism in time n^2 and output its answer.

We prove by contradition that D_1 accepts a different language than any M_i. For, suppose M_i accepts the same language. Then the following must be true for "large enough" j. For all n such that $f(i,j) \leq n \leq 2^{(f(i,j))^2}$, 1^n is accepted by D_1 (and hence M_i) iff 1^{n+1} is accepted by M_i. But by construction, $1^{2^{(f(i,j))^2}+1}$ is accepted by D_1 iff $1^{f(i,j)}$ is *not* accepted by M_i. This is a contradiction.

3.3 Why diagonalization may not resolve P vs NP

Diagonalization relies upon the ability of one TM to simulate another. Alternatively, we may say that diagonalization treats TMs as blackboxes i.e., does not examine their internal workings. Such simulations also work if all Turing Machines are provided with the same oracle. (Whenever the TM being simulated queries the oracle, so does the simulating TM.) If we could resolve P vs NP (in whichever direction) using such diagonalization then the proof would also work in the presence of any oracle. However, we now exhibit oracles B, C such that $P^C = NP^C$ and $P^B \neq NP^B$, which implies that such a proof cannot exist.

For C we may take $TQBF$ since $P^{TQBF} = NP^{TQBF}$. Now we construct B. For any language A, let A_u be the unary language

$$A_u = \{1^n : \text{some string of length } n \text{ is in } A\}.$$

For every A, language A_u is clearly in NP^A. Below we construct a B such that $B_u \notin P^B$. For such a B, we conclude that $NP^B \not\subseteq P^B$.

Let $M_1, M_2, M_3, \ldots$ be an effective enumeration of all polynomial-time Oracle Turing Machines. We construct B in stages. Initially, B is empty, but we gradually add strings to it. We say that M_i^B *queries the oracle for string y on input x* if while computing on x M_i at some point asks the oracle whether or not $y \in B$.

Let $t_1 = 1$. The ith stage of constructing B is as follows:

Run M_1^B on the inputs $1^{t_i}, 1^{t_i+1}, 1^{t_i+2} \ldots$ until you find a $j \geq t_i$ such that the following happens: When M_i^B is run on 1^j, it asks the oracle for $< 2^j$ strings of length j whether or not they are in B. When this happens and if M_i^B rejects, take a string of length j that M_i did not query and put it in B. Otherwise if M_i^B accepts, do nothing to B. But in both cases, set t_{i+1} to be the smallest integer k such that none of $M_1, M_2, \ldots, M_i$ queried any string of length $\geq k$ while computing on inputs of length $\leq j$. Proceed to the $i+1$st stage.