198:538 Complexity of Computation — Homework 1 Solutions

Rutgers University, Spring 2007

Problem 1

(a) Since $L_R \in P$, there is a polynomial-time algorithm A which on input $(M, x, z, 1^t)$ decides if there is a y (with z a prefix of y and $|y| \leq t$) such that M accepts (x, y) in at most t steps. We are going to construct a polynomial-time search algorithm S for L, using A as a subroutine. Our algoritm S will start with z and by asking A the proper questions will extend it bit by bit to an answer y (if one exists).

S$(M, x, z, 1^t)$

$p \leftarrow z$
if A$(M, x, p, 1^t)$ rejects
    then return No
while TRUE
        do if A$(M, x, p0, 1^t)$
                then $p \leftarrow z0$
            elseif A$(M, x, p1, 1^t)$
                then $p \leftarrow p1$
            else return p

It is easy to see that before each while loop (and if an answer y exists), it holds that z is an extendable prefix of some y. The algorithm will terminate after at most t iterations.

(b) Let R' be any NP-search problem described by verifier M, input x, polynomial bound $p(\cdot)$. Then the search problem R (defined as in part (a)) is an NP-search problem, and by our assumption that $P = NP$ there must be a polynomial time algorithm for L_R. Hence, we can run the search algorithm S for R on input $(M, x, \varepsilon, 1^{p(|x|)})$ (where ε is the empty string).

Problem 2

First note that there is a polynomial-time turing machine V, which on input (x, y) verifies whether y is a valid answer for x or if it is not. Now let $M_1, M_2, \ldots,$ be an enumeration of turing machines. Our algorithm A on input x will simulate machines $M_1, M_2, \ldots, M_n$ (where $n = |x|$) on x. Since A doesn't know if those machines ever stop, it cannot simulate them sequentially. A will simulate one step of M_1, then one of M_2, and so on; until it reaches M_n, at which point it starts all over again.

In the process of this simulation, when a machine M_i halts and outputs a y, our algorithm runs V to see whether $(x, y) \in R$; if the answer is positive it halts and returns y, otherwise it continues with the simulation.

To take care of the case when there is no y such that $(x, y) \in R$, A runs in parallel an exponential search algorithm S for R. Let the running time of S to be at most 2^{n^c}, for a constant c.

Suppose now, that a search algorithm M for R exists among the machines $M_1, M_2, \ldots, M_n$. In this case, if t is the running time of M, A will simulate at most t steps of machines $M_1, M_2, \ldots, M_n$ until the answer is found. This can be done in $O(nt^2)$ time for the n simulations (the square on t accounts for the simulation overhead) plus an additional n^c for the verification.

If M is not among $M_1, M_2, \ldots, M_n$, then the answer will be given (if not from one of these machines) from the exponential search algorithm for R that is run in parallel.

All in all, if M is the k-th machine in the enumeration, we have the following running times. If $x \in L$ then the running time is $O(nt^2 + n^c)$. (When $n < k$ the running time is $O(2^{k^c})$, but this is just a constant consumed by the O-notation.) If $x \notin L$ then the running time is $O(2^{n^c})$, as required.

Problem 3

(a) As it was shown in class, there exist functions $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by any circuit of size $s(n)$. For each such function f, let $L'_f = \{x \in \{0,1\}^n \mid f(x) = 1\}$. Now order the set of these languages by inclusion, and pick a minimal language L'. There has to be at least one element x_0 in L' (otherwise f would be an easy function). Observe that by the minimality of L' we know that $L = L' - \{x_0\}$ has to be in SIZE$(s(n))$.

(b) In view of part (a) it is enough to argue that $L \cup \{x_0\}$ is in SIZE$(s(n) + O(n))$. This is true because we can augment the circuit for L with a small circuit that checks whether $x = x_0$.

(c) The same argument for Turing Machines would have to consider functions that take as input a string of any length. This has the effect that there might be no minimal element in the corresponding ordering of the functions.

Problem 4

(a) From problem 3 we know that there are languages in SIZE(n^{11}) that are not in SIZE(n^{10}). It suffices to show that such a language is in Σ_4. Now fix an input length n and consider the smallest circuit C_n that computes a function on n bits not computable by any circuit of size n^{10}. We know C_n will have size at most n^{11}. Define L on inputs of length n as the set of all x accepted by C_n.

Recall that circuits of size s can be described by strings of $O(s \log s)$ bits, and when we say one circuit is smaller than another we mean that it is described by a lexicographically smaller string.

We show that L is in Σ_4. For this, observe that $C = C_n$ can be uniquely described as the circuit with the following two properties:

- If D is a circuit of size n^{10}, then C and D do not compute the same function.

- If E is a smaller circuit than C, then E computes some function in SIZE(n^{10}). Namely, there is a circuit F of size n^{10} such that E and F compute the same function.

Formally, we have that

$$\begin{aligned} x \in L \iff \; & \exists C \text{ of size at most } |x|^{11} \text{ such that} \\ & \forall D \text{ of size } |x|^{10}, \exists y \text{ such that } C(y) \neq D(y) \text{ and} \\ & \forall E \text{ smaller than } C \\ & \quad \exists F \text{ of size } |x|^{10} \text{ such that } \forall z, E(z) = F(z) \text{ and} \\ & C(x) = 1. \end{aligned}$$

By construction, for sufficiently large input lengths n, L is not computable by any circuit of size n^{10}.

(b) Consider the relation of NP and SIZE(n^{10}). If NP $\not\subseteq$ SIZE(n^{10}), then clearly $\Sigma_2 \not\subseteq$ SIZE(n^{10}). On the other hand, if NP $\subseteq$ SIZE(n^{10}), then $\Sigma_2 = \Sigma_4$ by the Karp-Lipton theorem. It follows from part (a), that $\Sigma_2 \not\subseteq$ SIZE(n^{10}).