CSC363 - Summer 2007
Assignment 2

Problem 1 [20p]

Show that a language L is decidable iff there exists an enumerator E that enumerates the strings in L in lexicographical order.

Problem 2 [10p]

Let Σ be a finite alphabet.

1. Let A be the set of all functions $f : \Sigma^* \to \{0, 1\}$. Is A countable or uncountable? Justify briefly.
2. Let B be the set of all functions $f : \{0, 1\} \to \Sigma^*$. Is B countable or uncountable? Justify briefly.

Problem 3 [10p]

Prove that the halting problem for C programs is undecidable. In your proof, you may *not* mention the Church-Turing thesis or Turing Machines. Simply adapt the diagonalization-like argument seen in class to work for this problem.

Formally, let us assume that C programs are single-functions with signatures of the form `int f(char *w)`. Assume that a C program either halts with an integer output by calling the special `return(int value)` function, or it does not halt (in particular, treat any crash as a not halt). Furthermore, assume that you have access to a built-in function `int run(char *P, char *x)` that interprets `P` as a C program and `x` as an input, and it effectively runs program `P` on input `x` (in reality, one can actually implement this function by compiling the code of `P` at run-time and running the resulting executable on input `x`). If `P` halts on input `x`, the `run` function returns the value `P(x)`. If `P` does not halt on input `x`, the call to the `run` function will not halt either.

Show that there is no C program with signature `int halt(char *P, char *x)` such that for all C programs `P` and inputs `x`, `halt(P,x)` returns 1 if the call `P(x)` halts and 0 if it does not halt.

Note: We said that C programs take only one `char *` argument and now we're asking about a C program that takes two arguments. Do not get stuck into this kind of details, that's not the point of the question. If it makes you feel better, assume that both `halt` and `run` work as they should when `P` is a C program with a single argument and crash otherwise.

Problem 4 [20p]

We can encode each TM M as a unique string $\langle M \rangle$ over the alphabet $\{0, 1\}$. Let $\langle M_1 \rangle, \langle M_2 \rangle, \ldots$ be the list of all TM encodings in lexicographical order. You may assume that it is computationally possible to (i.e. some TM can) accomplish the following: when presented with an input w, find out if $w = \langle M_i \rangle$ for some i, and output that index i.

Define $f : \mathcal{N} \to \mathcal{N}$ by $f(k) =$ index in the above list of the k-th TM M such that $L(M) = \emptyset$.

Notice that f is well defined for every k, since there are infinitely many TMs M with $L(M) = \emptyset$. For example, if $L(M_2) = L(M_5) = \emptyset$ and $L(M_1), L(M_3), L(M_4)$ are not empty, then $f(1) = 2$ and $f(2) = 5$.

Prove that f is not computable. Hint: assume it is computable by some TM M_f and show how to use M_f to construct another decider TM for a language we know is undecidable.

Problem 5 [10p]

Consider the following problem. We are given two TMs M_1, M_2 with the same input alphabet and a string w, and we need to decide if it is the case that: *if M_1 accepts w, then M_2 accepts w*. So: if M_1 does not accept w (rejects or doesn't halt), then the answer should be YES, regardless of what M_2 does. If M_1 accepts w and M_2 accepts w, again the answer should be YES. If M_1 accepts w and M_2 doesn't accept w, then the answer should be NO.

We formalize this problem as language membership by defining the language

$$L = \{\langle M_1, M_2, w \rangle : M_1, M_2 \text{ are TMs and } w \in L(M_1) \Rightarrow w \in L(M_2)\}$$

1. Show that $A_{TM} \leq_m L$.
2. Show that $\overline{A_{TM}} \leq_m L$.