CSC 363, Winter 2010 — Long Assignment #1

*Due at **start** of lecture on February 24. You may hand in up to **one** long assignment up to **one** day late without explanation (put it under my door in PT209E or SS6026A by 7pm). Other late assignments will be accepted only with a valid excuse (presented as soon as possible), and only up to the next tutorial time, when the solution will be reviewed (if this is not enough time, some other accommodation will be made). I prefer that you hand the assignment in on paper, but if you need to, you can email it (as a plain text file or PDF attachment, no Microsoft Word files please) to `radford@cdf.utoronto.ca`. (Please use this email address only for assignment submission.)*

This assignment is to be done by each student individually. You are encouraged to discuss the course material in general with other students, but you should not discuss this assignment (verbally, in writing, by email, or in any other way) with people other than the course instructor and tutors, except to clarify the meaning of the question. Handing in work that is not your own is a serious academic offense.

In answers to the questions below, you should give high-level descriptions of any Turing Machines you construct, similar to the proofs in Sipser's book. In particular, you needn't specify an exact method for encoding Turing Machines. You can use the fact that a Turing Machine can simulate the operation of another Turing Machine without giving the details of how this might be done. You can also use the fact that a k-tape Turing Machine can be converted to an equivalent 1-tape Turing Machine. Any other theorems proved in Chapters 1–4 of Sipser's book can also be used. You should, however, give details when they are essential to the proof idea. In general, a proof is an argument that convinces a skeptical but suitably knowledgeable person. You can assume the person to be convinced has understood the basic ideas in Chapters 3 and 4 of Sipser's book.

Question 1: For any language L, define the language L_π as the set of all permutations of strings in L. In other words,

$$L_\pi \;=\; \{\, s_1 s_2 \cdots s_n \,|\, \text{for some permutation } \pi_1\pi_2\cdots\pi_n \text{ of } 1,\ldots,n,\ s_{\pi_1}s_{\pi_2}\cdots s_{\pi_n} \text{ is in } L \,\}$$

Here, n can be any non-negative integer. Note that L_π will include the empty string if L does.

Give a procedure that takes a nondeterministic Turing Machine M that recognizes L and produces a nondeterministic Turing Machine, M_π, that recognizes the language L_π. For simplicity, assume that the input alphabet of M is $\{0, 1\}$ (the solution should easily generalize to any input alphabet), but the tape alphabet of M is not constrained. The state space for M_π should be the state space for M plus no more than five new states, one of which is the start state for M_π. (Solutions with more than five but less than ten additional states will receive part marks.) The transitions from the states in M_π that are taken from M should be the same in M_π as in M. You may extend the tape alphabet for M_π to include symbols not in the tape alphabet for M.

You should give complete details of the transitions for the new states, as a table for the δ function or as a state transition diagram.

Hint: Any permutation can be expressed as some sequence of swaps of adjacent symbols.

Question 2: Let's say that a tape square of a Turing Machine "has been used" if at any time during the computation so far it was set to a symbol other than blank (even if it has since been set to blank again). (Note: "has been used" is not standard terminology; I've just introduced it for this question.) At any time during the computation by a Turing Machine, M, on an input string, w, a finite set of tape squares will have been used. Let m be the number of the rightmost square that has been used, numbering squares 1, 2, 3, ... starting at the beginning of the tape (the leftmost square). Let h be the number of the square where the tape head is positioned. Prove that for every M, there is a constant k (which may depend on M, but not on the input, w) such that if at some point in the computation of M on input w, the difference $h - m$ is k or greater, then M does not halt on input w.

Hint: If the tape head is k squares past the last square that has been used, how must it have gotten there?

Question 3: Let L_0 and L_1 be languages on the alphabet $\Sigma_{01} = \{0, 1\}$. Define the language L_2 on the alphabet $\Sigma_{012} = \{0, 1, 2\}$ that consists of strings of length zero or greater in which each symbol is the sum of the symbols in corresponding positions from some string of the same length in L_0 and some string of the same length in L_1. In other words,

$$L_2 \;=\; \{\, s_1 s_2 \cdots s_n \,|\, \text{for some } a_1 a_2 \cdots a_n \in L_0 \text{ and } b_1 b_2 \cdots b_n \in L_1, \text{ each } s_i \text{ is equal to } a_i + b_i \,\}$$

where n may be any non-negative integer. For example, if $0110 \in L_0$ and $1100 \in L_1$, then $1210 \in L_2$.

Prove that if L_0 and L_1 are Turing recognizable, then L_2 is also Turing recognizable.

Question 4: For any language L, define the language $B(L)$ as follows:

$$B(L) \;=\; \{\, 0w \,|\, w \in \overline{L} \,\} \;\cup\; \{\, 1w \,|\, w \in L \,\}$$

Prove that there exists a language L such that both $B(L)$ and $\overline{B(L)}$ are not Turing recognizable.

Question 5: Let's say that the computation of a Turing Machine M on input w is "confined" if during computation the tape head is positioned at only a finite number of different tape squares. (Note: "confined" is not a standard term; I've just introduced it for this question.) Obviously, any computation that halts (in the accept or reject state) must be confined. If the computation loops, it might or might not be confined.

Define the following language:

$$\text{CONF}_{TM} \;=\; \{\, \langle M, w \rangle \,|\, M \text{ is a TM and the computation of } M \text{ on } w \text{ is confined} \,\}$$

Prove the following:

a) CONF_{TM} is Turing recognizable.

b) $\overline{\text{CONF}_{TM}}$ is not Turing recognizable.

Hints: If a computer has a finite amount of memory, and loops forever, what must happen sooner or later? If a computer program loops forever, can you modify it so that it at least does something as it loops?