CSC 363, Winter 2010 — Solutions to Long Assignment #1

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.

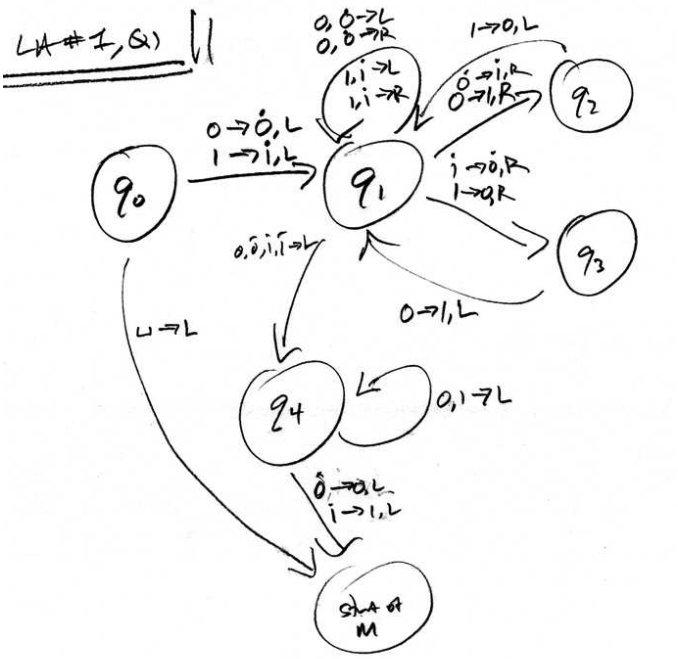
Give a procedure that takes a nondeterministic Turing Machine M that recognizes L and produces a nondeterministic Turing Machine, M_π, that recognizes the language L_π. For simplicity, assume that the input alphabet of M is $\{0,1\}$ (the solution should easily generalize to any input alphabet), but 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.

We can nondeterminismistically perform some number of swaps, changing 01 to 10 or 10 to 01, then move the tape head to the beginning, and finally transition to the start state of M. A swap of 01 to 10 can itself be done in a way that exploits nondeterminism, by changing a 0 to 1, moving right, then changing the following 1 to 0, but having no transition if the following symbol isn't a 1, so that that computation branch ends if changing the 0 to a 1 wasn't the right thing to do. Finally, we need to somehow mark the front of the tape to allow us to move the tape head back to the beginning. I'll use versions of 0 and 1 symbols marked with dots.

These operations are implemented with five additional states in the state diagram below:



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?

Let k be the number of states in M plus one. The basic idea is that if the tape head is k or more squares past the last square used, M must have been in the same state more than once since it moved past the last square used, which indicates that it is in a loop that will keep moving further right.

Here is a more detailed argument. Suppose the tape head is k or more tape squares past the last tape square (numbered m) that has been used. Since M starts with the tape at the leftmost square, and moves by only one square per step, there must have been some most recent time when the tape head was positioned at square m. We consider the computation since that time (call it t). Since the tape head moves only one square at a time, it must have been positioned since time t at all the squares numbered $m + 1$, $m + 2$, ..., $m + k$. Now let q_i be the state that M was in the first time the tape head was at square $m + i$ (after time t). Since k is the number of states plus one, it's not possible that q_1, ..., q_k are all different. Suppose that $q_j = q_{j+\ell}$ with $\ell > 0$. We know that the operation of M took it from being in state q_j with the head at square $m + j$ to being in the same state with the tape head at a square further to the right, numbered $m + j + \ell$. Furthermore, while M did this, it always saw tape squares that were blank (since all this happed after time t, which was the last time the tape head was at a square that has been used). M will therefore operate in exactly the same way again, moving the tape head to the square numbered $m + j + 2\ell$, and so forth, moving to the right without limit. M will therefore never halt.

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,

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

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.

If L_0 and L_1 are recognizable, they are recognizable by some Turing Machines, say M_0 and M_1. To show that L_2 is recognizable, we can show how to construct a Turing Machine M_2 that recognizes L_2, using runs of M_0 and M_1 as "subroutines". We need to do this in a way that will allow M_2 to accept when it should even if some of the runs of M_0 and M_1 loop.

If a string w in $\{0, 1, 2\}^*$ has k occurrences of 1, there will be 2^k ways that it can be written as a sum of two strings in $\{0, 1\}^*$. For example, 1210 can be written as $0100 + 1110$, $0110 + 1100$, $1100 + 0110$, or $1110 + 0100$. We can construct M_2 so that in parallel it runs M_1 on each of the 2^k strings that are the first in these sums, while also running M_2 on each of the 2^k strings that are the second in these sums. So for the example above, M_1 is run on 0100, 0110, 1100, and 1110 and M_2

is run on 1110, 1100, 0110, *and* 0100 *(of course, these are the same sets of strings, just in different orders). By "in parallel" is meant that it does one step of each in turn, so that each machine (on each input) will perform computations regardless of whether or not some machine on some input is looping. If at any point in these parallel runs, the computations of M_1 and M_2 on strings that add up to w have both accepted, then M_2 accepts. Otherwise, M_2 just loops.*

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.

Let L be any language that is not recognizable, such as $\overline{A_{TM}}$.

$B(L)$ cannot be recognizable, because if it were recognizable, say by the Turing Machine M, then L would also be recognizable. A machine S to recognize L could be constructed that on input w would change the input tape from w to $1w$, put the head at the beginning of the tape, and then transfer to the start state of M. The machine S will accept w iff M accepts $1w$, which by the definiton of $B(L)$ happens iff w is in L. So S recognizes L, which we know is impossible. Hence no machine M that recognizes $B(L)$ can exist.

$\overline{B(L)}$ cannot be recognizable, because if it were recognizable, say by Turing Machine M', then L would also be recognizable. A machine S' to recognize L could be constructed that on input w would change the input tape from w to $0w$, put the head at the beginning of the tape, and then transfer to the start state of M'. The machine S' will accept w iff M' accepts $0w$, which happens iff $0w \in \overline{B(L)}$, which happens iff $0w \notin B(L)$, which by definition happens iff $w \notin \overline{L}$, which is equivalent to $w \in L$. So S' recognizes L, which we know is impossible. Hence no machine M' that recognizes $\overline{B(L)}$ can exist.

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?

***Proof for (a)**: $CONF_{TM}$ can be recognized by a 4-tape Turing Machine that operates as follows. The input $\langle M, w\rangle$ is on tape 1. We reject if it does not describe a valid Turing Machine M. Otherwise, we copy w to tape 2 and simulate the operation of M on this input tape, using tape 3 for scratch storage. On tape 4, we record every configuration of M as it is simulated. A configuration contains*

all the information that determines the subsequent course of the computation of M — namely, the state of M, the position of the tape head of M, and the contents of M's tape. After simulating each step of M, we compare the current configuration of M with all previous configurations stored on tape 4. If the current configuration is the same as a previous configuration, we know that M is looping through a finite number of configurations, which involve only a finite number of tape squares. We can therefore accept, since the computation of M on w is confined. We also accept if M enters its accept state or reject state. If we do not accept for one of these reasons, M must pass through an infinite number of configurations, which must involve an infinite number of tape squares, so the computation is not confined. This machine therefore recognizes $CONF_{TM}$.

***First proof for (b)**: $\overline{CONF_{TM}}$ is not recognizable because if it were recognizable we could also recognize $\overline{HALT_{TM}}$, which we know is impossible. Given a Turing Machine S that recognizes $\overline{CONF_{TM}}$, we could create a Turing Machine T that recognizes $\overline{HALT_{TM}}$ that operates as follows. On input $\langle M, w\rangle$, it rejects if the input is invalid, and otherwise modifies the description of M to produce a description of a machine M' that behaves the same as M except that after every step of M it uses one more tape square. After replacing the input tape with $\langle M', w\rangle$, the machine T transfers control to the start state of S, so T accepts $\langle M, w\rangle$ iff S accepts $\langle M', w\rangle$*

The tape alphabet for M' is $\Gamma' = \Gamma \times \{0, 1, 2\}$, where Γ is the tape alphabet for M (the blank symbol for M' is identified with $(blank, 0) \in \Gamma'$). The $\{0, 1, 2\}$ part of a symbol in Γ' allows M' to use more and more tape squares without actually changing what is computed. After performing each step according to M, the machine M' write 2 to this part of the symbol under the tape head to mark where it is, and then moves right until it reaches a blank square (ie, $(blank, 0)$. It replaces this blank with $(blank, 1)$, moves the tape head back to the square marked with 2, and then continues with the next step according to M.

It's easy to see that M' uses t tape squares after it has done the equivalent of t steps of M. So a computation of M' will be confined iff M' (and hence M) halts on input w. The machine T will therefore recognize $\overline{HALT_{TM}}$, since it accepts iff $\langle M', w\rangle$ is in $\overline{CONF_{TM}}$, which happens iff $\langle M, w\rangle$ is in $\overline{HALT_{TM}}$.

***Second proof for (b)**: As for the first proof, we will show that $\overline{CONF_{TM}}$ is not recognizable because if it were recognizable we could also recognize $\overline{HALT_{TM}}$, which we know is impossible. Given a Turing Machine S' that recognizes $\overline{CONF_{TM}}$, we could create a Turing Machine T' that recognizes $\overline{HALT_{TM}}$ that operates as follows. On input $\langle M, w\rangle$, it rejects if the input is invalid. Otherwise T' runs the recognizer S' for $\overline{CONF_{TM}}$ on $\langle M, w\rangle$ in parallel with a simulation of running M on w that checks whether the current configuration is a repeat of one seen before, as in the proof of part (a). If S' accepts, or if a repeat configuration is found, T' accepts (we know that M will not halt on w). If the simulation of M on w halts, T' rejects. One of these must eventually happen, so T' recognizes $\overline{CONF_{TM}}$ (and in fact it decides $\overline{CONF_{TM}}$).*