CSC364 **Problem Set 3** **Fall/2000**

This problem set is worth 5% of your mark, and it is important to do your own work. You may consult with others concerning the general approach for solving problems on assignments, but you must write up all solutions entirely on your own. Anything else is *cheating* and is subject to prosectuion under the University's Code of Behavior. This homework is due at the beginning of tutorial on Wednesday, November 29. Late assignments will receive a grade of 0. This homework will help to prepare you for the third test, which will take place during tutorial on December 6.

1. Sipser, Problem 7.17 Show that if $P = NP$ then every language $A \in P$ except the trivial languages ($A = \emptyset$ and $A = \Sigma^*$) are NP-complete. For this problem, you must use the more restricted definition of a polynomial-time reduction as defined in Sipser, Definition 7.24. Namely, a language L is NP-complete if (a) L is in NP and (b) for all $L' \in NP$, L' is polynomial-time many-one reducible to L.

   Assuming $P = NP$, we want to show that any language in P (except the empty language and the set of all strings over the alphabet) is NP-complete. Fix some $A \in P$. Since A is not the emptyset, and also not the entire set of strings, we know that there exists some string $x \in A$, and another string $y \notin A$. We want to show that A is NP-complete. First, A is in NP since $A \in P$.

   Secondly we need to show that for every $B \in NP$, that there is a polynomial-time reduction, f, from Σ_B^* (the set of all strings over the language of B) to Σ_A^* with the property that $f(b)$ is in A iff b is in B. Since we are assuming $P = NP$, there exists some polynomial time Turing machine, M_B that decides B. The reduction is as follows (on input b): 1. Simulate M_B on b. 2. If M_B accepts, output x, and otherwise output y.

   The function f is polynomial time since M_B always runs in polynomial time. Secondly, it has the property that $f(b)$ is in A iff b is in B, so we are done.

2. Sipser, Problem 7.26

   We will reduce SAT to PUZZLE. Let $f = C_1 \wedge C_2 \wedge ... \wedge C_m$ be an instance of SAT with n underlying variables. Assume without loss of generality that each clause contains distinct literals. There will be $n + 1$ cards altogether. Each card has m rows, one for each clause, and 2 columns. There will be n cards, for each of the n variables. The card for x_i will have a blank circle in position (j,2), that is the jth row and 2nd column if x_i occurs positively in clause C_j; there will be a blank circle in position $(j, 1)$ if x_i occurs negatively in clause C_j. There will be two blank circles in (j,1) and (j,2) if x_i does not occur at all in clause C_j. Additionally there is one more card, with blank circles everywhere in the first column, and all circles filled in everywhere in the second column.

   We want to prove that if f is satisfiable, then the cards have a solution. Let α be a satisfying assignment. Then if $x_i = 1$ in α, place the card for x_i as we described in above (with x_i on the right, and $\neg x_i$ on the left). Otherwise, if $x_i = 0$ in α, place the card for x_i turned over (with $\neg x_i$ on the right and x_i on the left). The row corresponding to C_j will be covered if $x_i \in C_j$ and $x_i = 1$ or if $\neg x_i \in C_j$ and $x_i = 0$. Since every clause is satisfied, every row is covered.

We also want to prove that the converse is true; that is, if the cards have a solution then f is satisfiable. Suppose we have a card solution. If the extra $n+1^{st}$ card is placed in the other direction, turn the entire stack over. Otherwise, if the ith card representing x_i is placed as shown (x_i to the right), set $x_i = 1$ and otherwise set $x_i = 0$. For any C_j, the jth row can only be covered by those cards that represent literals in C_j. A card x_i covers the jth row iff $x_i \in C_j$ and card i is placed with x_i to the right, or $\neg x_i \in C_j$ and card i is placed with x_i to the left. In other words, the ith card covers the jth row iff the assignment to x_i forces C_j to true.

3. Sipser, Problem 7.29

   Assume $P = NP$, and construct a polynomial time algorithm for finding a satisfying assignment, if one exists.

   Assume that M is the polytime Turing machine deciding SAT. Our algorithm is intuitively as follows. First, we ask M if f is satisfiable. If M says no, then we quit and return $UNSAT$. Otherwise, we ask M if f has a satisfying assignment where $x_1 = 1$. If the answer is yes, then we ask M if f has a satisfying assignment where $x_1 = 1$ and $x_2 = 1$. If the answer is no, then we ask M if f has a satisfying assignment where $x_1 = 0$ and $x_2 = 1$. Thus at the ith step, we will know of a partial assignment α setting variables $x_1, .., x_i$ such that α can be extended to a total assignment that satisfies f, and we will ask if there is a satisfying assignment consistent with α and $x_{i+1} = 1$. Any path of the algorithm will ask n queries to M, so the algorithm will run in polytime.

   To implement the above, we can't really ask if f has a satisfying assignment where, for example, $x_1 = 1$, $x_2 = 0$, $x_3 = 1$. Instead, we must create a new f' with the property that f' is satisfiable iff f has a satisfying assignment where $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$. To create f, we simply set $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$ in f, and simplify as follows: if a clause has a literal in it that was set to 1, then this clause is removed, and if a clause has a literal in it that was set to 0, then this literal is removed. Note that f' will involve only the remaining variables not asked about (in this case $x_4, .., x_n$) and f' will be satisfiable iff f has a satisfying assignment where $x_1 = 1$, $x_2 = 0$ and $x_3 = 1$.

   Thus the algorithm is as follows. First run M on f. If M rejects, then quit and return "UNSAT" Otherwise continue. Let $f_1 = f$. For $i = 1$ to n do the following: (i) Obtain f' from f_i by setting x_i to 1 in f_i; (ii) Obtain f'' from f_i by setting x_i to 0 in f_i. (iii) Run $M(f')$. (iv) If it returns YES, then output "$x_i = 1$" and set $f_{i+1} = f'$ (iv) Otherwise if it returns NO, then output "$x_i = 0$" and set $f_{i+1} = f''$.

4. a. Let B be the set of all infinite sequences over $\{0, 1\}$. Show that B is uncountable using a proof by diagonalization. Your proof should proceed along the following lines. Assume for contradiction that B is countable. This means that there is a one-to-one onto function f_B mapping the natural numbers, N to B. Construct a 2-dimensional table, where row 1 contains the sequence $b_1 \in B$ such that $f(1) = b_1$, and more generally row i contains the sequence $b_i \in B$ such that

$f(i) = b_i$. Show that this leads to a contradiction by finding a sequence $b' \in B$ that is not listed in any of the rows of the table.

Assume for sake of contradiction that B is countable, and thus there is a one-to-one, onto mapping f from the natural numbers N to B. We will construct a 2-dimensional table, where row i contains the sequence b_i in B such that $f(i) = b_i$. Now consider the following infinite sequence, s over $\{0, 1\}$. The i^{th} bit of the sequence, s_i will be 1 if the i^{th} bit of $f(i)$ is 0, and otherwise s_i will be 0. In other words, s_i will be the complement of the infinite sequence on the diagonal of the table. By construction, s_i is not any of the sequences listed in the table since s_i differs from the i^{th} row in the i^{th} bit position. Therefore, the mapping f was not onto as assumed, and we have reached a contradiction.

b. Now let B' be the set of all *finite* sequences over $\{0, 1\}$. Since now B' is countable, the proof that you gave above should fail for B'. Show specifically where and why the proof that you constructed above goes wrong.

   Now let B' be the set of all finite sequences over $\{0, 1\}$. We will attempt to prove that B' is uncountable as we did above. Assume for sake of contradiction that there is a one-to-one onto mapping f from N to B'. Again we can construct a 2-dimensional table, where again row i contains the sequence $f(i)$. As above, we can construct an infinite sequence s that differs from the diagonal. But this sequence will be infinite, so it will not be an example of a finite sequence that differs from all of the ones in the table.

5. Which of the following problems about Turing machines are recursive which are not recursive? Explain your answers carefully.

   a. Determine given a Turing machine, M, whether the language accepted by M is finite.

   Call the above language L_{finite}. We will show that L_{finite} is not recursive. Recall that the universal language, A_{TM}, consists of all strings of the form $< M, w >$ such that M accepts w. We will reduce A_{TM} to L_{finite} by defining a function f that maps instances of A_{TM} to instances of L_{finite} such that $x \in A_{TM}$ if and only if $f(x) \in L_{finite}$.

   We will reduce the universal language to the complement of L_{finite}. This implies that the complement of L_{finite} is not recursive, and hence that L_{finite} is not recursive.

   Let x be of the form $< M, w >$. From $x =< M, w >$, construct the encoding of a new Turing machine, $B_{M,w}$, where $B_{M,w}$ will be a machine that takes a string α as input and accepts α if and only if M accepts w. (This is similar to the construction done in class to show that L_{empty} is not recursive.) Thus, $f(< M, w >) =< B_{M,w} >$. It is not too hard to see that f is computable. We want to show that $x =< M, w >$ is in A_{TM} if and only if $f(x) =< B_{M,w} >$ is in $\overline{L}_{finite}$. This is true because if $< M, w > \in A_{TM}$, then M accepts w, and therefore $B_{M,w}$ will accept every string, and thus $< B_{M,w} > \notin L_{finite}$. Conversely, if $< M, w >$ is not in A_{TM}, then M does not accept w, and thus

$B_{M,w}$ will accept no strings, and thus $< B_{M,w} > \in L_{finite}$. This completes the argument.

b. Determine, given a Turing machine M, whether M ever writes a nonblank symbol when started on the empty tape.

   This language is recursive. We will describe an algorithm that always halts and that accepts M if and only if M writes a nonblank symbol when started with the all blank tape. Recall that since M is given by an encoding, that q_1 is the initial state of M. First, the algorithm computes k, where k is the total number of states of M. (This is easily determined by looking at the code for M.) Then the algorithm simply simulates M on the all blank tape, keeping track of the states that have been passed through so far. If we ever follow a transition where a nonblank symbol is written, then we immediately halt and accept M. Otherwise, within $k+1$ steps, we will reach some state that has already been passed through, and still no nonblank symbol has been written. If this happens, then the computation will be in an infinite loop where a nonblank symbol is never written, so in this case, we halt and reject M.

c. Determine, given a Turing machine M, a state q and a string w, whether M ever reaches state q when started with input w from its initial state.

   Call the above language $L_{initial}$. will show that $L_{initial}$ is not recursive by defining a reduction, f from instances of the universal language, A_{TM}, to instances of $L_{initial}$, such that $x \in A_{TM}$ if and only if $f(x) \in L_{initial}$.
   Let $x =< M, w >$. Since M is given by a fixed encoding as discussed in class, we know that q_1 is the start state, and q_2 is the single accept state of M. (If this were not the case, then we would first need to obtain the encoding of a machine M' that is the same as M except that M' has only one accept state, q_a.) Let $f(< M, w >) =< M, q_2, w >$. Clearly f is computable. Now $x =< M, w > \in A_{TM}$, if and only if M accepts on w, which holds if and only if M reaches state q_2 on w, and this holds if and only if $f(x) =< M, q_2, w > \in L_{initial}$. Since A_{TM} is not recursive, it follows from our reduction that $L_{initial}$ is also not recursive.