MATH 4LT/6LT3 Assignment #5 Solutions
Due: Friday, 21 November by 11:59pm.

1. Determine if the following language is computable. Justify your answer.

   > NON-BLANK is equal to the set of strings $\ulcorner M \urcorner$ such that M is a DTM that when started with an empty tape eventually writes a non-blank symbol on its tape.

   Solution: If a Turing machine M never writes a nonblank symbol then all that it does during its computation is move to the left or right, and write out blank symbols on the tape. Because of this limited behaviour, M will very quickly start repeating itself, or come to an early halt. On empty input, the tape will remain blank and so M's configuration can be specified by its current state, and the position of the read/write head, relative to the left end of the tape.

   Since M's tape has a left end and since, whenever it is reading the first cell, an instruction to move left results in the read/write head not moving at all, it follows that if M ever reaches some non-halting state q more than once, then M will never halt. So, if we let M run on empty input for $|Q|$ steps then either we will see it write some non-blank symbol, or M will halt, without writing a non-blank symbol before $|Q|$ steps or M's tape will remain blank, and M will have repeated some non-halting state q.

   In the latter case, we can conclude that M has entered an infinite loop (and will occupy state q infinitely often) and will never write a non-blank symbol.

   So, we conclude that the language NON-BLANK is computable. The halting DTM M' that does the following on input x will have NON-BLANK as its language:

   - Check to see if $x = \ulcorner M \urcorner$ for some DTM M. If not, reject.
   - Let n be the number of states of M.
   - Run M on empty input for up to n steps. If before then M has written a non-blank symbol, then M' accepts x. Otherwise it rejects x.

2. Exercise 2.10.31 from the textbook.

   Solution: Let B and C be languages over the alphabet Σ, with B computable and C nontrivial. Let x and y be strings over Σ with $x \in C$ and $y \notin C$. Recall that in a previous homework assignment, it was shown that the function $\chi_B : \Sigma^* \to \{0, 1\}$ is computable, since B is, where $\chi_B(z) = 1$ if $z \in B$ and is equal to 0 if $z \notin B$.

   Define $\rho : \Sigma^* \to \Sigma^*$ to be the function with $\rho(z) = x$ if $z \in B$ and $\rho(z) = y$ if $z \notin B$. Then ρ is computable, since it can easily be computed from the function χ_B. It can be seen that ρ is a many-one reduction of B to C, showing that $B \leq_m C$.

3. Exercise 2.10.38, parts 4 and 5, from the textbook.

   Solution: For part (4), to show that HALT-ALL is not computable it suffices to show that HP $\leq_m$ HALT-ALL since we know that HP is not computable. Let $\rho : \Sigma^* \to \Sigma^*$ be the computable function that on input a string $\langle \ulcorner M \urcorner, x \rangle$ produces the string $\ulcorner M + x' \urcorner$, where $M + x'$ is the DTM that on input a string y does the following:

   - Erases y, and writes x on its tape.
   - Runs the DTM M.

   We see that if M halts on input x, then M'_x will halt on all strings y and so $\ulcorner M'_x \urcorner \in$ HALT-ALL. On the other hand, if M doesn't halt on input x, then M'_x doesn't halt on any inputs and so $\ulcorner M'_x \urcorner \notin$ HALT-ALL in this case. This shows that ρ is a many-one reduction from HP to HALT-ALL.

   For part (5) we can use Rice's theorem to show that this language is not computable. Let $\mathcal{P}$ be the property of a language that it has exactly 12 members. Then this is a nontrivial property of CE languages and so the language $L_{\mathcal{P}} = \{\ulcorner M \urcorner \mid M$ is a DTM such that $L(M)$ has property $\mathcal{P}\}$ is not computable. But this is the language ACCEPTS-DOZEN.

4. Exercise 3.10.12 from the textbook.

   Solution: To show that Y is not computable, we will show that $HP \leq_m Y$. Let ρ be the computable function that on input a string $\langle \ulcorner M \urcorner, x \rangle$,

with M a DTM and x a string, outputs the string $\ulcorner M'_x \urcorner$, where M'_x is the DTM that on input a string y does the following:

- Erases y, and writes x on its tape.
- Runs the DTM M.

Note that this is the same function used in the previous exercise. We claim that ρ is a many-one reduction of HP to Y. If $\langle \ulcorner M \urcorner, x \rangle \in HP$ then on input x, M eventually halts. Let C be the number of steps for this to happen. Then the runtime of M'_x on input a string y is equal to $|y| + |x|$. This is roughly the number of steps needed by M'_x to erase the input string y and write the fixed string x. So, on input $|y|$, the runtime of M'_x is approximately $|y| + (|x| + C)$, which is a polynomial in $|y|$ (since x is a fixed string in the definition of M'_x. Thus M'_x is a polynomial-time DTM and so $\rho(\langle \ulcorner M \urcorner, x \rangle) \in Y$.

On the other hand, if $\langle \ulcorner M \urcorner, x \rangle \notin HP$ then M doesn't halt on input x and so for any string y, M'_x does not halt, which implies that M'_x is not a polynomial-time DTM and so $\rho(\langle \ulcorner M \urcorner, x \rangle) = \ulcorner M'_x \urcorner \notin Y$.

5. A triangle of a graph G is a set of three distinct vertices a, b, and c of G such that there are edges in G between each pair of vertices. Let

$$T = \{\langle G \rangle \mid G \text{ is a graph that has a triangle}\}.$$

Show that T is PTIME, i.e., demonstrate that there is an algorithm that decides if a given graph has a triangle or not, and whose run time can be bounded by a polynomial in the number of vertices of G.

Solution: Let G be a graph with n vertices. To determine if G has a triangle, we need only search over all triples of vertices (a, b, c) of G to see if any of them form a triangle in G. The number of such triples to check is equal to $n(n-1)(n-2) < n^3$. To check if a given triple (a, b, c) is a triangle, we need to search through the edge set of G to determine if (a, b), (b, c), and (c, a) are edges. The number of edges of G is bounded by n^2, and so the number of steps needed to determine if a given triple is a triangle in G can be bounded by a quadratic polynomial in n. Since we need to conduct this test at most n^3 times (one for each potential triangle), the above algorithm has run-time that can be bounded by a 5th degree polynomial in n.

The following question is for students enrolled in MATH 6LT3. Students in MATH 4LT3 can treat it as a bonus question.

B1 Let $A = \{\ulcorner M \urcorner \mid M \text{ is a DFA with } L(M) \neq \emptyset\}$, i.e., A consists of the codes of all DFAs that accept at least one string. Here the details of the coding scheme for DFAs are not important for the purpose of this question. One can use a similar scheme to the one for coding DTMs. Show that A is a computable language, and in fact that $A \in \mathcal{P}$, the class of polynomial-time solvable languages. To show this you can informally describe an algorithm that solves this problem and provide a polynomial upper bound on its run-time, as a function of the size of the DFA.

Solution: Consider the transition diagram of a DFA M. It can be viewed as a digraph G whose edges are labelled with the letters of M's alphabet. If M accepts some word $a_1 a_2 \dots a_k$, then we can trace out a directed path in G of length k from M's initial state to one of its accepting states (the edges in this path will be labelled with the a_i's, in order). Conversely, if there is a directed path in G from M's initial state to one of its accepting states, then the word corresponding to the concatenation of the labels of the edges in the path will be accepted by M. Thus, M will accept some word if and only if there is a directed path from M's initial state to one of its accepting states. If there is such a path, then there will be one whose length can be bounded by the number of edges of G, since if any edge is repeated, then a shorter path can be constructed by removing a segment of the given path.

The above shows that the following algorithm will correctly decide membership in A: on input $\ulcorner M \urcorner$ for some DFA $M = (Q, \Sigma, q_s, F, \delta)$,

- First check to see if $q_s \in F$. If so, then M accepts the empty word, and in this case, we accept $\ulcorner M \urcorner$ and halt. If not, then
- build the transition diagram of M, as a directed graph G. The vertices of G will be the states of M, and for states q and q', there will be an edge from q to q' if, for some symbol $a \in \Sigma$, $\delta(q, a) = q'$.
- For each accepting state $q \in F$, check to see if there is a directed path from q_s to q in G. If there is one, we accept $\ulcorner M \urcorner$ and halt.

- If after checking for each accepting state q we do not find a directed path, we reject $\ulcorner M \urcorner$ and halt.

Since we are given the transition function of M as input (maybe as a table of transitions, or a list of the transitions, represented as tuples of the form (q, a, q'), we can construct the graph G in time bounded by some small degree polynomial in $|\ulcorner M \urcorner|$. Note that $|\ulcorner G \urcorner|$ will have size less than (or at least comparable to) $|\ulcorner M \urcorner|$, assuming that we are using a reasonable scheme to represent graphs and DFAs as strings. Checking if $q_s \in F$ can also be done in time bounded by a degree one polynomial in $|\ulcorner M \urcorner|$. In the discussion of the language EUL-CYCLE in Example 3.2.12 of the textbook, part of the polynomial-time algorithm presented there checks to see if pairs of vertices are mutually reachable. In the third step in the above algorithm, something similar (and simpler) needs to be done, namely, check to see if some accepting state is reachable from the initial state. As noted in the example, this sort of check can be carried out in time bounded by some polynomial in the size of the graph, and hence in $|\ulcorner M \urcorner|$. This shows that $A \in P$, since the runtime of the above algorithm can be bounded by a polynomial in $|\ulcorner M \urcorner|$.