Homework 9 – Due Wednesday, November 24, 2021 at 11:59 PM

Reminder Collaboration is permitted, but you must write the solutions *by yourself without assistance*, and be ready to explain them orally to the course staff if asked. You must also identify your collaborators and write "Collaborators: none" if you worked by yourself. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Note You may use various generalizations of the Turing machine model we have seen in class, such as TMs with two-way infinite tapes, stay-put, or multiple tapes. If you choose to use such a generalization, state clearly and precisely what model you are using.

Problems There are 4 required problems and one bonus problem.

1. (**Hierarchy Theorems**) You may assume without saying so that any reasonable-looking function (logarithms, polynomials, exponentials, and combinations thereof) are time-constructible.

   (a) Show that $\mathsf{P} \subseteq \mathsf{TIME}(n^{\log n})$.

   (b) Use the time hierarchy theorem to show that $\mathsf{EXP} \not\subseteq \mathsf{TIME}(n^{\log n})$.

   (c) Combine parts (a) and (b) to conclude that $\mathsf{P} \neq \mathsf{EXP}$.

2. (**Fun with Encodings**)

   (a) Prove that there is **no** polynomial-time algorithm that takes as input a natural number n (written in binary) and outputs (i.e., writes to its tape) the number $n!$ (again, written in binary).

   Hint: Explain why the expected output for this problem is so long that it is impossible for a poly-time algorithm to even write it down.

   (b) Give a high-level description of a polynomial-time algorithm that takes as input a natural number n (written in **unary**) and outputs the number $n!$ (written in **binary**). Explain why your algorithm is correct and why it runs in polynomial time.

   Hint: You can use without proof the fact that Turing machines can perform basic arithmetic operations on binary numbers, like addition and multiplication, in polynomial time.

3. (**Polynomial-Time Algorithms**)

   (a) Let $A = \{ww^R \mid w \in \{0,1\}^*\}$. Show that $A \in \mathsf{TIME}(n^2)$ and $A \in \mathsf{SPACE}(n)$ by i) giving an **implementation-level** description of a basic, single-tape Turing machine M that decides A, ii) briefly explain why your TM correctly decides A, and iii) analyzing the running time and space usage of M.

   (b) If $G = (V, E)$ is an undirected graph, and u, v are vertices in V, let $d_G(u, v)$ denote the length of the shortest path from u to v in G. (Define $d_G(u, v) = \infty$ if no path exists.) Let $CLOSER = \{\langle G, u, v, w\rangle \mid G \text{ is an undirected graph}, u, v, w \text{ are vertices, and } d_G(u, v) < d_G(u, w)\}$. This corresponds to the following computational problem: Given a graph G and vertices u, v, w, is u closer to v than it is to w?

Show that $CLOSER \in \mathsf{P}$ by i) giving a high-level description of a polynomial-time algorithm deciding $CLOSER$, ii) analyzing the correctness of your algorithm, and iii) explaining why your algorithm runs in polynomial time.

You don't need to specify the exact polynomial runtime that your algorithm runs in, since this may depend on implementation details that are suppressed in a high-level description. Just give a convincing argument that the runtime is polynomial as in the examples in Chapter 7.2 of Sipser.

4. (**Closure Properties**)

   (a) Show that P is closed under the union operation.

   (b) This part of the problem will walk you through a proof that P is closed under the star operation. Let L be a language in P, and let M be a TM deciding L in time $O(n^c)$ for some constant c. Consider the following algorithm S_1 that decides L^*. Explain why S_1 does *not* run in polynomial time.

Algorithm: $S_1(w)$

Input : String $w = w_1 w_2 \dots w_n$ of length n

1. For each way to break w into a concatentation of *strings* $s_1 \circ s_2 \circ \cdots \circ s_k$ (for $k \leq n$):
2. For each $i = 1, \dots, k$:
3. Run M on input s_i.
4. If all runs have accepted, *accept*.
5. *Reject*.

   (c) The following recursive algorithm uses a slightly different idea. Its correctness relies on the fact that a string $w_1 w_2 \dots w_n \in L^*$ if and only if there exists an index $i \in \{0, \dots, n-1\}$ such that $w_1 w_2 \dots w_i \in L^*$ and $w_{i+1} \dots w_n \in L$. Explain why S_2 does *not* run in polynomial time.

Algorithm: $S_2(w)$

Input : String $w = w_1 w_2 \dots w_n$ of length n

1. If $n = 0$:
2. Run M on input ε. If it accepts, *accept*. If it rejects, *reject*.
3. For each $i = 0, 1, 2, \dots, n-1$:
4. Run S_2 on input $w_1 w_2 \dots w_i$ and run M on input $w_{i+1} \dots w_n$.
5. If both runs accept, *accept*.
6. *Reject*.

   (d) The main issue with the recursive algorithm in part (c) is that it for each prefix $w_1 w_2 \dots w_i$ of w, it keeps repeating the work of checking whether that prefix is in L^*. Wouldn't it be great if for each i, you only had to check whether $w_1 w_2 \dots w_i$ is in L^* once?

   It turns out you can, and this forms the basis of an actual polynomial time algorithm. Think of filling out an array $T[0, 1, \dots, n]$ where each cell $T[i]$ contains the answer to the question, "is $w_1 w_2 \dots w_i \in L^*$?" Design an algorithm S_3 that systematically fills in this array and uses it to determine whether $w \in L^*$. Briefly explain why your algorithm runs in polynomial time, and how it allows you to conclude that P is closed under the star operation.

5. (**Bonus problem**) Show that your algorithm from problem 3(a) is optimal: There is no basic, single-tape TM algorithm deciding the language $A = \{ww^R \mid w \in \{0,1\}^*\}$ in time $o(n^2)$.