Homework 10 – Due Thursday, April 23 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. Collaboration is not allowed on problems marked "INDIVIDUAL."

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. **You may describe Turing machines at a high-level on this assignment.**

Problems There are 4 required problems.

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

   The time complexity class $\mathsf{P} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(n^k)$ consists of all languages decidable in polynomial time. The time complexity class $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$ consists of all languages decidable in exponential time (i.e., in time that is an exponential of a polynomial).

   (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. (**Closure Properties**)

   (a) Show that P is closed under the union operation, i.e., show that for all languages $L_1, L_2 \in \mathsf{P}$, we have $L_1 \cup L_2 \in \mathsf{P}$.

   (b) This question 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 $k \leq n$ and each way to break w into a concatentation of *strings* $s_1 \circ s_2 \circ \cdots \circ s_k$:
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 $w_1w_2\dots w_n \in L^*$ if and only if there exists an index $i \in \{0,\dots,n-1\}$ such that $w_1w_2\dots w_i \in L^*$ and $w_{i+1}\dots w_n \in L$.

Algorithm: $S_2(w)$

Input : String $w = w_1w_2\dots w_n$ of length n

1. If $n = 0$: *Accept.*
2. For each $i = 0, 1, 2, \dots, n-1$:
3. (Recursively) run S_2 on input $w_1w_2\dots w_i$ and run M on input $w_{i+1}\dots w_n$.
4. If both runs accept, *accept.*
5. *Reject.*

Explain why S_2 does *not* run in polynomial time.

(d) The main issue with the recursive algorithm in part (c) is that it for each prefix $w_1w_2\dots w_i$ 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 $w_1w_2\dots w_i$ 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 $w_1w_2\dots w_i \in L^*$?" 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.

3. **(NTM satisfiability)**

(a) Consider the language

$$TMSAT = \{\langle N, w, 1^t\rangle \mid \text{ NTM } N \text{ accepts input } w \text{ within } t \text{ steps}\}.$$

Prove that $TMSAT \in \mathsf{NP}$ by giving a **deterministic polynomial-time verifier** for it.

(b) If t was written in binary, would your solution to part (a) still work? Why or why not?

4. (**Decision vs. search**) You are consulting for a private equity firm that wants to improve the kitchen morale of all the ~~soulless, formulaic~~ beloved local restaurants it's cranking out. A "chef team" consists of a head chef, a saucier, and a pastry chef. The firm sends you sets X, Y, and Z, with r individuals each, corresponding to the three roles. You also receive a set M of triples, where each triple corresponds to a potential chef team where all 3 individuals can cook together without fighting. An individual can appear in multiple triples. The firm is asking you to find a way to arrange the largest possible number of happy chef teams. That is, your goal is to select a largest subset of triples from M so that each individual appears in at most one triple. (There may be more than one largest subset, in which case, you are free to pick any largest subset.)

In this problem, you will prove that if $\mathsf{P} = \mathsf{NP}$, you can *find* a largest set of happy chef teams in polynomial time.

(a) Consider the following language:
$COOK = \{\langle X, Y, Z, M, k\rangle \mid |X| = |Y| = |Z|$, each element of M is a triple (x, y, z) where $x \in X, y \in Y$ and $z \in Z$, and M contains a subset of size k where each element appears in at most one triple$\}$.

Prove that $COOK \in \mathsf{NP}$ by giving a **nondeterministic poly-time algorithm** for it.

(b) If $\mathsf{P} = \mathsf{NP}$, the solution you gave to part (a) implies that $COOK \in \mathsf{P}$. That is, in deterministic polynomial time you can decide whether it is possible to have k happy chef teams. Assume you have a deterministic poly-time subroutine that does this, and use it repeatedly to *find* a largest set of happy chef teams in deterministic polynomial time. (Note that the output of your algorithm should be a set of triples.)

Hint: Similar to Solved Problem 7.40 in Sipser.