CSE396, Spring 2021 Problem Set 10 (the last) Due Thu. 5/6, 11:59pm

The **Final Exam** will be *remote-only* on **Thursday, May 13**, still at the advertised time, **11:45am—2:45pm**. *It is not in Knox 20* despite being listed that way. The rules and logistics will be the same as for the prelim exams except that you are expressly allowed to refer to your notes and the text with no single-hardcopy-page restriction. This corresponds to the open-book rules of the in-person final exams of past years. We will trust that any use of electronic vehicles for these notes will obey University regulations, with your requirement to remain on-camera in Zoom as the one mode of invigilation.

Reading:

This week will cover the "computation histories" part of section 5.1 plus much of chapter 7, with these differences:

- A special case of a **linear bounded automaton** which I call a "two-head DFA" (2HDFA) will be used as the machine of choice for checking computations. (IMHO, a 2HDFA is also good for leading in to the idea of the Post Correspondence Problem, which is illustrated in one figure but otherwise skipped.) It is defined in my "Week 14" notes on the course webpage but not in the text.
- The proof of the Cook-Levin theorem will be the "alternative proof" given as Theorem 9.34 at the end of chapter 9, so when you hit the word tableau in section 7.4, stop there. I have notes at https://cse.buffalo.edu/~regan/cse396/CSE396CookLevin.pdf but is just as well to read my new "Week 14" notes.
- We will skip details of the reductions in section 7.5 from (3)SAT to the other mentioned graph problems.

Please read the posted "Week 14" notes along with the sections of the text. The lectures are structured to emphasize two main points:

- How the concept of *checking computation histories* carries into Boolean circuits, the equivalence of the "verifier" definition of NP to the nondeterministic machine definition, and ultimately the Cook-Levin proof.
- The analogy between $\mathsf{REC}, \mathsf{RE}, \mathsf{co\text{-}RE}$ and $\mathsf{P}, \mathsf{NP}, \mathsf{co\text{-}NP}$ and the relation of these classes to existential quantification ($\exists$) and universal quantification ($\forall$) in logic.

The latter especially matters to the *TopHat* portion. Please review the notes before attemption it—and hearing Tuesday's lecture first is recommended.

Homework—part online (TopHat), part written, all *individual work*, due **Thursday** 5/6:

(1) Using *TopHat*, the "Worksheet" titled *S21 HW10 Online Part* (10 Qs, 20 pts.)

The other *three* problems are to be submitted as PDFs using the *CSE Autograder* system.

(2) Consider the following decision problem:

EXCEPTION THROW
Instance: A Java program P and an input stream $x \in ASCII^*$.
Question: Does running P on x throw an ARRAYINDEXOUTOFBOUNDSEXCEPTION?

Show that this decision problem is undecidable—by arguing concretely that if it were decidable, then the Acceptance Problem for Turing machines would be decidable, which it isn't. (It may help you to know that the *Turing Kit* program, though it has other bugs, never throws this exception. 18 pts.)

(3) Consider the following decision problem:

- INSTANCE: Two Turing machines M_1 and M_2.
- QUESTION: Are their languages complementary, i.e., $L(M_1) \cup L(M_2) = \Sigma^*$, and also $L(M_1) \cap L(M_2) = \emptyset$?

Prove that the language of this problem is neither c.e. nor co-c.e. (18 pts.—there are shortcut answers to this question but you must explain fully why they work.)

(4) Consider the following decision problem:

- INSTANCE: A regular expression R over $\Sigma = \{0, 1\}$ that does not have any uses of the star operation; and a number n given as a string of n zeroes.
- QUESTION: Is there a binary string x of length n such that x does *not* match R?

Show that the language of this decision problem belongs to the class NP. You will need to argue that the relation "x matches R" belongs to P, i.e., is decidable in deterministic polynomial time. The most convincing way to do so (IMHO—and this works even when R *does* have stars) is to use the theorem in class that converted a regular expression into an equivalent NFA N_R and note the size of the NFA you get in terms of the size of R. Then argue you can simulate N_R on x *without* converting N_R into an equivalent DFA, by instead keeping track of which states of N_R it could possibly be in as you process each bit of x. Give a time bound in terms of the length n of x and the number m of states in N_R; you may also regard m as "order-of" the size of R. (18 pts. for a fully-reasoned answer, making 54 on the handwritten portion and 74 overall points on the set. The answer key will include the detail that this problem is in fact NP-complete.)