CSE491/596, Fall 2020 Problem Set 3 (really "2b") Due Mon. Oct. 12

Lectures and Reading. This week begins the general treatment of computation apart from bit-level details of machines but with attention to running time and space usage. Monday's lecture will highlight decidable problems and those further decidable in linear and polynomial time. Wednesday will prove that the following *diagonal language* is not c.e.:

$$D_{TM} = \{\langle M\rangle : M \text{ does not accept the string } \langle M\rangle\}$$

Here we need not give express details of how $\langle M\rangle$ codes the instructions of M as a (binary) string. We could write the instructions in ASCII or UNICODE and convert that to binary, or we could refer to some `.tmt` file for M in the *Turing Kit* software. One can even enumerate the codes of valid Turing machines as $M_0, M_1, M_2, \ldots$ and use the index i of M in this enumeration in place of $\langle M\rangle$, writing M_i or just i. Then i is called a *Gödel Number* after the way the logician Kurt Gödel similarly digitized logical formulas. One convenience of Gödel numbers is that you don't have to worry about invalid codes, so that with this revised definition,

$$D_{TM} = \{i : i \notin L(M_i)\} \qquad \text{and} \qquad K_{TM} = \{i : M_i \text{ does accept } i\}$$

are literally complements of each other. We next observe that K_{TM} **is** c.e., since it is subsidiary to

$$A_{TM} = \{\langle M_i, x\rangle : x \in L(M_i)\},$$

which is the language of the Acceptance Problem, and more simply, is the language of any universal Turing machine that uses the same scheme of coding Turing machines. It follows that:

- D_{TM} is not c.e., but it is co-c.e.
- K_{TM} and A_{TM} are c.e. but not co-c.e.
- In particular, all of these languages are **undecidable**.

Friday's lecture will introduce **reductions** as a technique for finding and proving further undecidable problems. For these subjects, we *first* refer to the notes by John Watrous, required reading #4 on the course webpage. (Watrous uses slightly different names, e.g. A_{DTM} and DIAG in place of A_{TM} and D_{TM}, but his order of business is similar to mine. Note however that he puts Theorem 7.5 in Debray's notes, which I've temporarily skipped over, after them in section 15.4, but I will cover that on Wednesday—so please do read chapter 15 by Wednesday if you can.) *Then* we read the last page of Debray's section 8 and Debray's section 9 in-tandem with chapter 16 of Watrous. Chapter 17 of Watrous will be treated as summing up the study of the classes REC, RE, and co-RE, especially the relation of RE to computable functions in section 17.4. The Prelim I exam will not have a problem involving creating a reduction, but attendant facts about these language classes will be relevant.

A reminder that Prelim I is being held on **Friday, Oct. 16**, but I have a change for the alternative Friday time to an hour earlier (the small group meeting then will be moved to earlier in that week). So the exam times will be **3–3:50pm** in class period and **10–10:50am** earlier. The exam will be *remote-only* with submission via *Autograder*. The exam is nominally open-book, but you are still recommended to compile a single notes sheet under the previous year's in-class terms because the exam (unlike the final) does not intend to allow time for going through books and lengthy notes. The domain of the exam is reflected by assignments through this one and lectures up to, **not** including, proofs of undecidability **via** mapping reductions.

Assignment 3 (really "2b"), due 10/12 11:59pm

(1) For any string $x \in \{0,1\}^*$, let $m = \#1(x)$, and let $n_x = (n_1, \ldots, n_{m-1})$ stand for the numbers of 0s between each 1 in x and the next 1 in x. Zeroes before the first 1 and after the last 1 in x do not count, so if x has at most one 1, then n_x is the empty vector. For the following conditions a, b, c let L_a, L_b, L_c stand for the language of strings meeting the corresponding condition:

(a) Every n_i in the list n_x is an even number.

(b) The list n_x is in nondecreasing order, i.e., $n_1 \leq n_2 \leq \cdots \leq n_{m-1}$.

(c) Every number in n_x is different; i.e., for all distinct $i, j \in \{1, \ldots, m-1\}$, $n_i \neq n_j$.

For each of L_a, L_b, L_c, say whether it is a regular language or not. If you say regular, give a DFA or regular expression whose correctness is clear. If you say not regular, you must give a proof via the Myhill-Nerode technique.

An element of this problem is that each of these definitions might be thought of as ambiguous when the list n_x is empty. You must make your interpretation of such cases explicit. Your DFA or regular expression will require such an interpretation anyway, but the nub of the matter is, in the case(s) where you say "not regular," to make your proof work regardless of the interpretation. (27 points total)

(2) Write in pseudocode a decision procedure for the following decision problem:

INSTANCE: Two DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$.

QUESTION: Is $L(M_1) \cup L(M_2) = \Sigma^*$?

You may appeal to the DFA minimization algorithm in Debray's notes as a given but need not reiterate its details. Also say whether your procedure runs in time that is polynomial in the number $m = |Q_1| + |Q_2|$ of states in the DFAs. (18 pts.)

(3) Let A be a c.e. language and let B be a co-c.e. language. Prove that their difference $A \setminus B$ is always a c.e. language. Give an example, however, where A is decidable and yet the difference $A \setminus B$ is still undecidable. (15 pts., for 60 on the assignment)

More Presentation Ideas—besides (a)–(d) specified on the Assignment 2 paper.

(e) The question footnoted in the lecture notes for the single-tape Turing machine M such that $L(M) = \{a^n b^n : n \geq 0\}$. You should do both of the following: (i) either demonstrate a bug or show why M nevertheless remains correct; and (ii) show how to edit the machine so that it enforces the stated "tape invariant" directly and so its correctness is transparent. [The general issue of verifying program code correctness will be an option for the week after the first exam.]

(f) Consider finite automata T that read just one input x and produce one output, $y = T(x)$. The idea was introduced in (d) but for finite transducers that add two inputs. Here is a general

example of a function $f(x) = y$ that can be computed this way: Let L be any language over $\Sigma = \{0, 1\}$. Define $f_L(\epsilon) = \epsilon$ and for any nonempty string $x = wc$ with $c \in \Sigma$:

$$f(x) = f(w) \cdot (\text{if } x \in L \text{ then 1 else 0}).$$

For instance, if L is the language of binary palindromes then $f_L(01101) = 10010$ (moreover, $f(10010)$ also equals 10010, thus mapping to itself). That is, the ith bit of $f(x)$ always tells whether the prefix $x_1 \cdots x_i$ belongs to L.

When L is regular, we can always convert a DFA M accepting L into a T computing f_L. Explain how to do this. You have some freedom of whether to make each output bit occur "on an arc" or "in a state"—these styles gave rise to so-called "Mealy machines" and "Moore machines," respectively.

(g) Using $\Sigma = \{(,)\}$, sketch a finite transducer T such that for all $x \in \Sigma^*$, one of the following happens:

- $T(x)$ rejects because $|x|$ is odd or because x begins with ')' or ends with '('.
- $T(x)$ outputs a string y such that y is balanced if and only if x is balanced, and $|y| < |x|/2$.
- $T(x)$ accepts because x is the empty string or the string ().

Conclude that by iterating T on its own output, we obtain a "streaming algorithm" that decides whether a string of parentheses is balanced and runs in linear time. (This can be paired with item (f) or done on its own.)