CSE491/596 F23 **Problem Set 3** **Due Fri. 10/13 11:59pm**

There will be a lecture **online** via Zoom on **Monday, Oct. 9**, **7:30–8:20pm**. It will use the same Zoom as for office hours, which I will re-post just before on Piazza. This will be followed by an optional *Review Session* going over Assignment 2 and taking questions on Assignment 3.

The **First Prelim Exam** will be in class period on **Wednesday, Oct. 18**. It will be timed for 48 minutes after a 2-minute read-through. It will be based on assignments through this one and topics through decidability and undecidability. It does *not* include the topic of "mapping reductions" or the class NP. Lectures will, however, begin those topics before the exam. Skim Debray's sections 8–11, skip 12,read the rest of his section 13, and read 14 stopping just before the Cook-Levin Theorem. Coverage will also start blending in the Allender-Loui-Regan CRC Handbook chapters 27 and 28. Read the background discussion of time complexity and the definitions of the "canonical complexity classes" from P, NP upward for now. We will actually cover section 3 of ALR ch. 27 and sections 1–4 of ALR ch. 28 on NP-completeness before we cover space complexity and the detailed theorems in ch. 27 and ch. 28, section 5.

(1) One of the following three languages is regular; the other two are nonregular. All three are over the alphabet $\Sigma = \{a, b\}$. For the non-regular ones, prove that fact using the Myhill-Nerode technique from notes and lecture. For the regular one, give a regular expression for the language and explain why it works. ($3 \times 12 = 36$ pts. Besides renaming the "dummy variables" x and y in the statements of these languages to, say, u and v, another trick is to reword the definitions to have just a single variable before the colon. For instance, $L_1 = \{w : w$ can be broken as $w = u \cdot v$ such that $\#a(u) = \#b(v)\}$.)

(i) $L_1 = \{xy : x, y \in \{a, b\}^*, \#a(x) = \#b(y)\}$.

(ii) $L_2 = \{\, a^k b^m a^n : k, m, n \geq 0$ and $(k < m \vee m < n)\,\}$.

(iii) $L_3 = \{xay : x, y \in \{a, b\}^*, \#a(x) = \#b(y)\}$.

(2) Define the "double word" language $DW = \{xx : x \in \{a, b\}^+\}$. For example, *abab* belongs to DW but *abba* does not; so do *aaaaaa* and *aabaab* but not *aaabbb*. Because we wrote $\{a, b\}^+$ not $\{a, b\}^*$, the empty string is not in DW with this definition. It is easy to prove by a Myhill-Nerode argument that DW is not a regular language.

But now let us fix a number n and consider DW_n = the set of words of length $2n$ that belong to DW. That is, DW_n is the set of doublings of a-b binary words of length n. It is finite, as there are just 2^n such words, so it is regular. Show, however, that every DFA M_n such that $L(M_n) = DW_n$ must have at least 2^n states. (18 pts. For extra credit, show that every DFA must have some number $f(n) > 2^n$ ofr states. The higher your function $f(n)$, the more credit, up to another 18 pts.)

(3) Give a decision procedure for the following computational problem:

INSTANCE: A DFA M.

QUESTION: Is there a string x such that M accepts the double-word xx?

Here are some hints to get you started: First, note that x can have any length. If the start state of M is an accepting state the answer is immediately *yes* since $\epsilon \cdot \epsilon = \epsilon$. So you may presume this is not the case. But this also means that you can't make an algorithm that works by trying out a bunch of doubled strings—you'd have to try infinitely many.

Second, observe that the answer is *yes* if and only if there is a state q such that M processes x from its start state s to q and M would also process x from q to some final state. Let m be the number of states in M. For $j = 1$ to m make the following DFAs:

- M_j is the same as M except that it has state j as its start state. (If you number $s = 1$ then M_1 is just M.)
- M'_j is the same as M, keeping start state s, but has j as its *only* accepting state.

Then the answer is *yes* if and only if what is true about the language

$$L' = \bigcup_{j=1}^{m} \left(L(M_j) \cap L(M'_j) \right)$$

—? Explain how you could make a single big DFA M_{big} and apply a basic algorithm from the Friday 10/6 lecture or notes to it. But finally estimate, how many states does M_{big} have? Is it polynomial or exponential in m? Do you get a decision procedure that runs in polynomial time? (24 pts. total, for 78 regular-credit points on the set.)