

Theory of Computation
CS 378
Fall 2019

Problem Set 3

Due **AT THE BEGINNING OF CLASS** Tuesday, October 15

- This homework covers material from pages 44-83 in Sipser and some additional topics that are not found in Sipser.
 - As always, answers to all exercises should be accompanied by explanations; brief explanations sometimes suffice, but all answers should be explained. Answers without explanations may not be awarded full credit.
 - When using the Pumping Lemma to prove that a language is not regular, be careful with the “there exist” ($\exists$) and “for all” ($\forall$) quantifiers: Whenever there is a $\exists$, you may choose strings arbitrarily. Whenever there is a $\forall$, you must show that the step works for any choice of strings. (Recall the *adversary* argument presented in class.)
 - *A general note:* When writing up your homework, please explain your arguments clearly and write neatly. Graders may not award credit to incomplete or illegible solutions. Clear communication *is* the point, on every assignment.
 - *A general note:* Neatly written (or typeset) solutions, with enough blank space on the page to allow graders to write comments before returning the papers, are greatly appreciated!
1. **Hans und Franz und Nonregular Languages!** Prove that each of the languages below is not regular, using the Pumping Lemma. Your proofs must be clear, complete, and rigorous.

(a) $L = \{0^i 1^{2i} \mid i \geq 0\}.$

(b) $L = \{0^i 1^j 0^{i+j} \mid i, j \geq 0\}.$

(c) $L = \{ww^R \mid w \in \{a, b, c\}^*\}.$

2. **Addition is Regular, but Multiplication is not!** In Homework 1 we showed that the language of valid binary additions is regular. In this problem, we show that the related language of valid binary multiplications is not regular.

As before, the alphabet is the set of all 3×1 binary vectors:

$$\Sigma = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

A correct multiplication of two binary numbers can be represented by a string in Σ^* . For example,

$$\begin{array}{r} 0 1 1 0 \\ \times 0 0 1 0 1 \\ \hline 1 1 0 1 0 \end{array}$$

would be represented by the following string of six symbols from Σ .

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Use the Pumping Lemma to show that this language is not regular.

3. **Regular Expressions and Dynamic Programming!** Construct a regular expression for the DFA in Sipser 1.21 (a) on page 86 *using the **dynamic programming** formulation that we derived in class*. Clearly show each step of the construction. Do not simplify the expressions found by the algorithm. The expressions may be clunky and may have many unnecessary ϵ 's, but that's okay in this context: The point here is that the procedure is easily mechanized, not that the regular expressions constructed are particularly concise. Notice that it's easy to construct a simple regular expression for this language by inspection, but that's not the point here. (*Note:* As noted in our lecture notes, the HMU textbook can be a resource for this exercise. Do not use the technique in the Sipser textbook—it's substantially more involved.)

For this exercise, showing all your work in each step in the construction will be a sufficient explanation of the process. In addition to that regular expression, however, please describe in English the language of the DFA and then briefly argue (a few sentences or so could suffice!) that the language of the regular expression is the same language. You may not refer to the construction in your argument! Instead, argue *soundness* and *completeness*: for soundness, briefly argue that every string described by the regular expression is in the language of the DFA; and for completeness, describe (in no more than a few sentences!) why every string in the language of the DFA is described by the regular expression.

4. **DFA Testers!** The National Center for Extremely Secret Automata Design has hired the famous automata theorist, Professor Anna Litik, as a consultant. Professor Litik's task is to determine if a DFA designed by the Center accepts an infinite number of strings. The problem is that the Center refuses to show Professor Litik the actual DFA. Instead, they will only tell her how many states the DFA has, the alphabet of the DFA, and they will answer any number of questions of the form "Is string w accepted by your DFA?" Professor Litik's goal is to determine in finite time whether the secret DFA accepts an infinite number of strings.

Your task is to prove that a DFA with n states accepts an infinite number of strings **if and only if** it accepts some string of length greater than or equal to n and less than $2n$. (Notice that this result allows Professor Litik to "simply" query the DFA on the finite number of strings of lengths between n and $2n$ in order to reach her decision.)

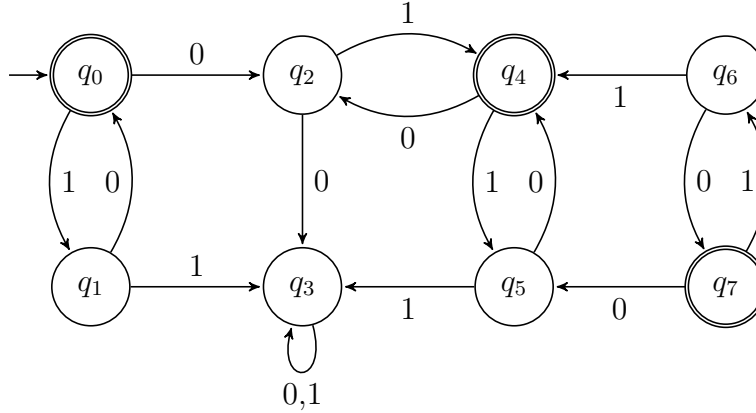


Figure 1: The DFA for the state minimization algorithm in Exercise 5.

Hint: As part of proving the claim that if the DFA accepts an infinite number of strings, then it accepts a string of length between n and $2n$, it could be helpful to consider the shortest-length string the DFA accepts that is of length n or more.

5. **State Minimization and More Dynamic Programming!** Use the dynamic programming state minimization algorithm from class to construct a DFA that accepts the language accepted by the DFA in Figure 1, but has the minimum possible number of states. (*Note:* As noted in our lecture notes, the HMU textbook can be a resource for this exercise; if there are differences between the method presented in class and the one in HMU, please be sure to follow the method from class.) Show your dynamic programming table with a **numerical index** in each “checked” cell indicating the length of the shortest distinguishing string for that pair of states. After you’ve built the table, draw the transition diagram for the new minimized DFA.

For this exercise, an explanation will be sufficient if it gives two illustrative examples of how a non-zero number was added to your dynamic programming table and gives two illustrative examples of how transitions were decided for the minimized DFA δ function. As with choosing test cases for code correctness, please be sure to choose the examples so that they are usefully illustrative of a correct answer, demonstrating that the answer shows full understanding of the state minimization algorithm. (That is, make sure they show what you know!)