Homework 3. Due November 7, 9:59PM.

CS181: Fall 2022

GUIDELINES:

- Upload your assignments to Gradescope by 9:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results proved in class without proofs as long as you state them clearly.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.
- Note that we have a **modified grading scheme for this assignment**: A sincere attempt will get you 100% of the credit and a reasonable attempt will get you 50% for each problem. Nevertheless, please attempt the problems honestly and write down the solutions the best way you can - this is really the most helpful way to flex your neurons in preparation for the exam.

1. As we mentioned in class the critical part of KMP string matching algorithm is the following: Given a pattern $p \in \{0,1\}^*$, we have to design a DFA that accepts only those strings that contain p. Let us explore this idea for a bit.

   Suppose the pattern p is 0110, can you design a DFA that only accepts strings that contain this pattern?

2. Converting a regular expression to a NFA. In class, we saw a procedure to build a NFA for every regular expression r. Let us understand the 'size' of the NFA created by this procedure as a function of the length of the regular expression r. Here, length of the regular expression refers to the number of characters in the regular expression.

   Suppose the regular expression r had length m. How many states would be present in the final NFA built by the procedure in class as a function of m (use big-Oh notation and avoid computing constants)? Explain your reasoning in a few sentences [.5 points]

   Here, by size of a regex you can take it to mean the length of the string that corresponds to the regex. For instance, the regex $(01)^*$ would have size 5. Alternately, more intuitively,

you can take size to mean the number of recursive subpieces and operations that you'd see within the regular expression. Any way you interpret the size, the answer should be same up to big-Oh notation.

Do not worry about figuring out the edge-cases and/or the exact constants.

[Hint: Look at each of the steps in the compound case, and see how the number of states can increase.]

3. Design a regular expression for the following language:

$$L = \{x : \text{ every 1 in } x \text{ is followed by at least two zeroes }\}.$$

For instance, $010010000, 1000, 0010010000$ are in the language but $10101, 011001000$ are not.You don't have to prove your expression works but write a few sentences describing why your expression could/does work. [.75 point]

4. Show that the following functions/languages are not regular:

   (a) $L = \{x : x = 0^m 1^{3m}\}$. For instance, $0111, 00111111$ would be in L, but $0100, 0011$ would not be elements of L. [.75 points]

   (b) $F : \{0,1\}^* \to \{0,1\}$ defined by $F(x) = 1$ if and only if $x = 1^{i^3}$ for some $i > 0$. [.75 points]

   (c) $UNEQUAL = \{0^m 10^n : m \neq n\}$. [.75 points]
   [Hint: This one is a bit tricky (but remember the relaxed grading scheme). You can try to directly show the language is not regular by using the pumping lemma by a careful choice of m, n. Think what choice of m, n would allow that repeating a piece of the first block of 0's can match the length of the second block of 0's? (You'll have to look at factorials ...).
   An alternate approach is to use closure properties of regular languages: recall the following corollary of what we showed in class if L, L' are regular then $L \setminus L' = L \cap (\overline{L'})$ is also regular (because $\overline{L'}$ is regular; and the intersection of two regular languages ir regular). Can you find a regular language L such that $L \setminus UNEQUAL$ can be shown to be not regular by pumping lemma much more easily?]

Your proofs should be at the similar level of detail as the examples from lecture 11; that is, write down the different steps as we did in class.

Additional practice problems - Not to be graded

1. Design a regular expression for the following languages:

   - $L = \{x : \text{ third bit from end of } x \text{ is a } 1\}$.
   - $L = \{x : \text{ number of 1's in } x \text{ is divisble by } 5\}$.
   - $L = \{x : \ x \text{ has an odd number of 1's}\}$.
   - $L = \{x : \ x \text{ has at least three 1's}\}$.
   - L = All strings except the empty string.

- $L = \{x : \ x$ has an even number of 0's or contains exactly two 1's$\}$.

2. Continuing problem (2) above:

   2a How many transitions/edges are present in the final NFA built by the procdure in class as a function of m (use big-Oh notation and avoid computing constants)?

   [Hint: Look at each of the steps in the compund case, and see how the number of edges added changes.]

   2b (Advanced) Do you see a way to modify the procedure to only add at most $O(m)$ transitions in the NFA? Such an NFA is what gets used in grep and other regex matching tools.

   [Hint: The most expensive step in terms of number of edges added is the compound case corresponding to the (*) operation. What happens if you maintain the invariant that all the NFAs you build only have one accept state?Can we maintain this invariant?]