Homework 3. Due November 8, 9:59PM.

CS181: Fall 2021

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. Show that for any NFA N, there exists a NFA N' which is equivalent to N (i.e., computes the same function as N) but has only **one** accept state. [.5 point]

   Consider NFA N with accepting states $S = \{S_1, S_2, \ldots, S_k\}$.
   We can construct a N' with a single accepting state S' by creating ϵ transitions between all previous accepting states in S to our new accepting state S'. S' has no transitions coming out of it.
   All inputs that would have ended on accepting states in N will now transition using newly constructed ϵ transitions to S'.

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]

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

To do this we should show the number of states for each of the operators.
Base Case
Representing 0 requires just two states, a start state with a transition to an accepting state on 0. Representing 1 requires the same but transition on 1.

1. Concatenation
If we are concatenating an NFA N with n states with NFA M with m states, we would add ϵ transitions from accepting states of N to the beginning on M and make accepting states from N no longer accept. This would require no additional states.

2. Kleene Star
To add the Kleene Star operator to an NFA that has m states we would just add a new accepting start state and ϵ transitions from the accepting states to the new start state. This requires a single additional state.

3. Union
To union two NFAs N of size n and M of size n, we add a new start state that has epsilon transitions to the start states of N and M. This requires 1 additional.

Since all operations require 0 or 1 additional states and the base case of a single character requires a constant number of states, we have shown that Regex with m characters can be represented by an NFA with O(m) states. Note that the number of union operators is bounded by the number of characters.

3. Design a regular expression for the following language:

$$L = \{x : \ x \text{ has an equal number of 10's as 01's.}\}.$$

For instance, 1001 is in the language as there is exactly one 10 and one 01. So are $101001, 0100010$. But $01, 1000, 001001$ 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]

This is equivalent to saying we accept all words that start and end with the same bit. If we start with 0, the only way the number of 10 is not equal to the number of 01 is if it ends in a 1. If we start with a 1, the counts would only not be equal if it ends with a 0.
Our regex would be $R = (0(0|1)^*0)|(1(0|1)^*1)|(0|1)|\epsilon$

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

   (a) $L = \{x : x = 0^m 1^n, m > n\}$. For instance, $00011, 001, 0001$ would be in L, but $0100, 0011$ would not be elements of L. [.75 points]

   Suppose L is regular. There exists a number p such that the pumping lemma holds.
   Suppose $x = abc$ where (1) $|b| > 0$ and (2) $|ab| \leq p$. For all i, (3) $x' = ab^i c$ is also in the language F by pumping lemma.
   Consider $x = 0^{p+1}1^p$

Note that by (2) b must include at least one zero.
Pump down to i = 0.
We are removing at least one zero from x, so now we know that $m \leq n$. This means $m > n$ no longer holds. Therefor $x = abc$ is in language L but $x' = ac$ is not, which is contradiction to property (3) of the pumping lemma. L must not be regular.

(b) $F : \{0,1\}^* \to \{0,1\}$ defined by $F(x) = 1$ if and only if $x = 1^{3^i}$ for some $i > 0$. [.75 points]
Suppose F is regular. There exists a number p such that the pumping lemma holds.
Suppose $x = abc$ where (1) $|b| > 0$ and (2) $|ab| \leq p$. For all j, (3) $x' = ab^jc$ is also in the language F by pumping lemma.
Consider $x = 1^{3^p}$.
Note that by (2) ab consists of at most p 1's.
Pump up to j = 2.
We now have between 1 and p more 1's in x' than in x. We know that $3^p+1 \leq |x'| \leq 3^p+p$. However the next largest word in the language has length 3^{p+1}.
Since $3^p < 3^p + 1 \leq |x'| \leq 3^p + p < 3^{p+1}$, x' lies between two valid words in F. It is longer than x but not long enough to be the length of the next largest word in F.
Therefor while $x = abc$ is in language F, $x' = abbc$ must not be in the language which is a contradiction to property (3). F must not be regular.

(c) $ADD = \{0^m10^n10^{m+n} : m, n \geq 1\}$. [.75 points]
Suppose ADD is regular. There exists a number p such that pumping lemma holds.
Suppose $x = abc$ where (1) $|b| > 0$ and (2) $|ab| \leq p$. For all i, (3) $x' = ab^ic$ is also in the language F by pumping lemma.
Consider $x = 0^p1010^{p+1}$
Note that by (2) b must consist of only zeros.
Pump up to i = 3.
x' must have at least 2 more zeros than x in the first p characters, so x' is of the form 0^k1010^{p+1} where $k > p+1$. Since $k+1 = p+1$ no longer holds, this means that x' is not in language ADD. This is a contradiction to property (3) of the pumping lemma since $x = abc$ is in the language ADD but $x' = abbbc$ is not. ADD must not be regular.

You need to use the pumping lemma appropriately for the above. 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. Show that if a language L is regular, then so is $Reverse(L) = \{x : Reverse(x) \in L\}$.
2. Design a regular expression for the following languages:
   - $L = \{x :$ third bit from end of x is a 1$\}$.
   - $L = \{x :$ number of 1's in x is divisible by 5$\}$.
   - $L = \{x :$ x has an odd number of 1's$\}$.
   - $L = \{x :$ x 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$\}$.

3. Continuing problem (2) above:

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

   [Hint: Look at each of the steps in the compound 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? We can maintain this invariant by Problem (1)!]