CSE 311: Foundations of Computing I

Homework 8 (due December 5th at 11:59 PM)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. You may use results from lecture, the theorems handout, and previous homeworks without proof.*

1. State Fair [Online] (30 points)

For each of the following, create a *DFA* that recognizes exactly the language given.

(a) [10 Points] The set of all binary strings that contain at least two 0's or at most two 1's.

(b) [10 Points] Binary strings where if we treat them as a binary number, that number is congruent to 3 modulo 5. For example, 01000 is in the language (because $8 \equiv 3 \bmod 5$) but 100 is not.

(c) [10 Points] The set L of binary strings that contain a palindrome of length 3 as a substring. So, $01101 \in L$ because it contains the palindrome 101, but $01100 \notin L$ because it contains no palindromes of length 3.

Submit and check your answers to this question here:

https://grinch.cs.washington.edu/cse311/fsm

You **must also** submit documentation in Canvas. For each part, include a screenshot of your submitted DFA along with, for each state s of your machine, a description of which strings will take the DFA from the start state to s.

2. Rage Against the (State) Machine [Online] (14 points)

For each of the following, create an *NFA* that recognizes exactly the language described.

(a) [7 Points] The set of binary strings that contain 00 or do not contain 11.

(b) [7 Points] The set of binary strings that contain 00 and do not contain 11.

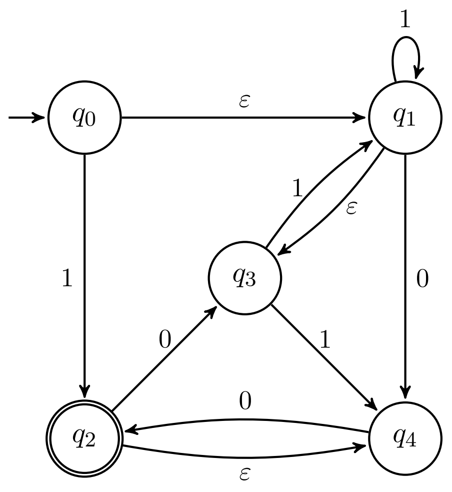
Submit and check your answers to this question here:

https://grinch.cs.washington.edu/cse311/fsm

You **must also** submit a screenshot of your submitted NFA in Canvas.

3. Do do do do do... Automata [Online] (15 points)

Use the construction from lecture to convert the following NFA to a DFA. Label each state of the DFA using appropriate states of the original NFA.



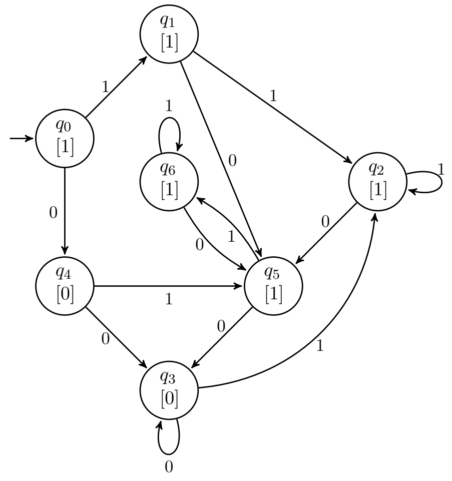
Submit and check your answers to this question here:

https://grinch.cs.washington.edu/cse311/fsm

You **must also** submit a screenshot of your submitted NFA in Canvas. As noted above, make sure that your states are labelled by the corresponding NFA states (if any).

4. Keep It To A Minimum (15 points)

Use the algorithm for minimization that we discussed in class to minimize the following automaton. For each step of the algorithm write down the groups (of states), which group was split in the step the reason for splitting that group. At the end, write down the minimized DFA.



5. Cog In The Machine [Online] (6 points)

Draw NFAs that recognize the language described by the regular expression $(101^* \cup 001^*)^*011^*$. Use the construction given in lecture or in the book or produce something simpler if you can.

Submit and check your answers to this question here:

https://grinch.cs.washington.edu/cse311/fsm

You **must also** submit a screenshot of your submitted NFA in Canvas.

6. Correct Grammar (20 points)

In this problem, you will work toward proving that the set of languages that can be described by context free grammars over Σ is at least as large as those that can be described by regular expressions over Σ.

Specifically, consider the following function mapping a regular expression $R \in$ **RegExp** to a CFG:

$$\begin{aligned} \text{cfg}(\varnothing) &= \text{the CFG with no productions} \\ \text{cfg}(\varepsilon) &= \text{the CFG with just the production } S \to \varepsilon \\ \text{cfg}(a) &= \text{the CFG with just the production } S \to a \qquad \text{for any } a \in \Sigma \end{aligned}$$

For the recursive cases, let A and B be two arbitrary **RegExp**s. Then, let C_A and C_B be cfg(A) and cfg(B), respectively, after renaming any nonterminals used in C_B to new names not used in C_A and after renaming the start symbols of these to be S_A and S_B, respectively. Then, we define the recursive cases like this:

$$\begin{aligned} \text{cfg}(AB) &= \text{the CFG with the productions of } C_A \text{ and } C_B \text{ and } S \to S_A\, S_B \\ \text{cfg}(A \cup B) &= \text{the CFG with the productions of } C_A \text{ and } C_B \text{ and } S \to S_A \mid S_B \\ \text{cfg}(A*) &= \text{the CFG with the productions of } C_A \text{ and } S \to \varepsilon \mid S_A\, S \end{aligned}$$

(In each case above, S is the start symbol of the CFG produced. Its terminal symbols are Σ, and its non-terminals are S and all symbols appearing on the left hand side of some production described.)

Prove that, for any $R \in$ **RegExp**, the language accepted by R is a subset of the language accepted by cfg(R). That is, prove that every string in the language of R is also in the language of cfg(R). (In fact, the two languages are actually the same, but the other direction is a little harder to prove, so we will skip that. However, note that showing the two languages are always the same would prove the original claim above: that the set of all context free languages is at least as large as the set of regular languages.)

7. Extra Credit: Pratt-Pratt-Pratt (0 points)

Suppose we want to determine whether a string x of length n contains a string $y = y_1 y_2 \ldots y_m$ with $m \ll n$. To do so, we construct the following NFA:

$$\begin{tikzcd}
{} \arrow[r] & s_0 \arrow[r, "y_1"] \arrow[loop below, "{0,1}"] & s_1 \arrow[r, "y_2"] & s_2 \arrow[r, "y_3"] & \cdots \arrow[r, "y_{m-1}"] & s_{m-1} \arrow[r, "y_m"] & s_m \arrow[loop below, "{0,1}"]
\end{tikzcd}$$

(where the ... includes states $s_3, \ldots, s_{m-2}$). We can see that this NFA matches x iff x contains the string y.

We could check whether this NFA matches x using the parallel exploration approach, but doing so would take $O(mn)$ time, no better than the obvious brute-force approach for checking if x contains y. Alternatively, we can convert the NFA to a DFA and then run the DFA on the string x. *A priori*, the number of states in the resulting DFA could be as large as 2^m, giving an $\Omega(2^m + n)$ time algorithm, which is unacceptably slow. However, below, you will show that this approach can be made to run in $O(m^2 + n)$ time.

(a) Consider any subset of states, S, found while converting the NFA above into a DFA. Prove that, for each $1 \leq j \leq m$, knowing $s_j \in S$ *functionally determines* whether $s_i \in S$ or not for each $1 \leq i < j$.

(b) Explain why this means that the number of subsets produced in the construction is at most $2m$.

(c) Explain why the subset construction thus runs in only $O(m^2)$ time (assuming the alphabet size is $O(1)$).

(d) How many states would this reduce to if we then applied the state minimization algorithm?

(e) Explain why part (c) leads to a bound of $O(m^2 + n)$ for the full algorithm (without state minimization).

(f) Briefly explain how this approach can be modified to count (or, better yet, find) *all* the substrings matching y in the string x with the same overall time bound.

Note that any string matching algorithm takes $\Omega(m + n) = \Omega(n)$ time in the worst case since it must read the entire input. Thus, the above algorithm is optimal whenever $m^2 = O(n)$, or equivalently, $m = O(\sqrt{n})$, which is the case for normal inputs circumstances.