CSE 311: Foundations of Computing I

Homework 8 (due Part I (questions 1-4), Wednesday, March 11 at 11:00 PM Part II (question 5), Friday, March 13 at 12:00 PM)

1. NFA Design (20 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 11 or do not contain 00

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

(c) [6 Points] Binary strings with at least three 1s **and** ending in 010.

You should submit and check your answers to this question using
`https://grinch.cs.washington.edu/cse311/fsm`.

2. RE-to-NFA (10 points)

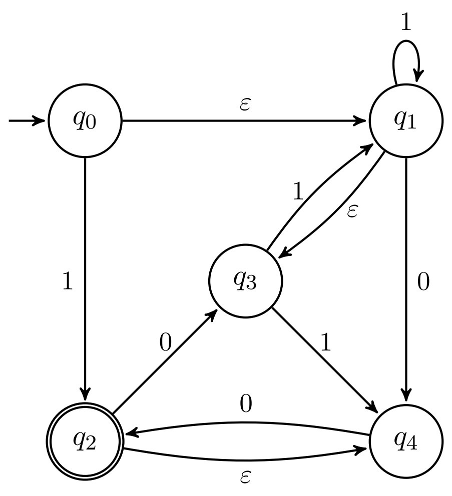
Draw an NFA that recognizes the language described by the regular expression $0110^*(110^* \cup 001^*)^*$. Use the construction given in lecture or in the book or produce something simpler if you can.

You should submit and check your answers to this question using
`https://grinch.cs.washington.edu/cse311/fsm`.

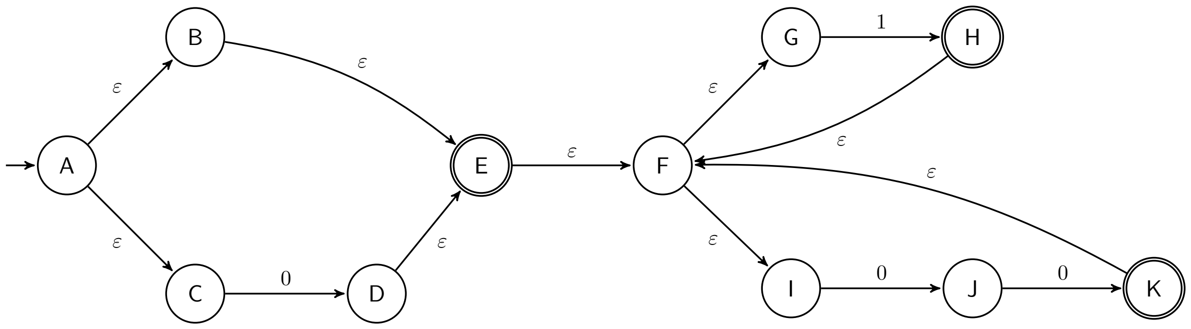
3. NFA-to-DFA (30 points)

Use the construction from lecture to convert each of the following two NFAs to DFAs. Label each state of the DFA using names for states just as they were done using that construction. You will need to document that you have done this. For example, if the state corresponds to $\{q_0, q_1, q_3\}$, then label the state "q_0, q_1, q_3". The empty state can be labeled "empty", "null", or have a blank label.

(a) [15 Points]



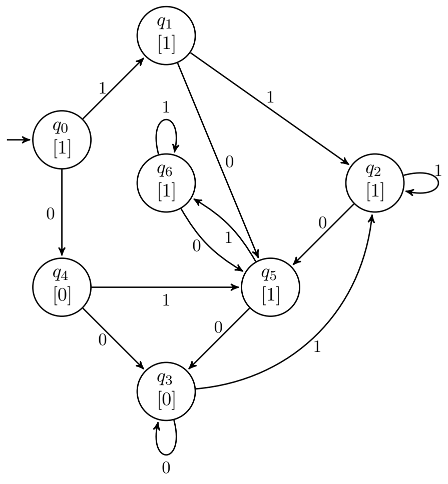
(b) [15 Points] The NFA below, which is nearly what we get by applying the construction described in class (the only simplification is using three states rather than four to represent the sub-expression 00) to the regular expression $(\varepsilon \cup 0)(1 \cup 00)^*$:



You should submit and check your answers to this question using
https://grinch.cs.washington.edu/cse311/fsm.
In addition, you must also take a screenshot of your diagram and submit it in Gradescope, to confirm that your states are labelled by the corresponding NFA states (if any). This documentation will be worth 4 points per part.

4. Mini-Me (20 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.



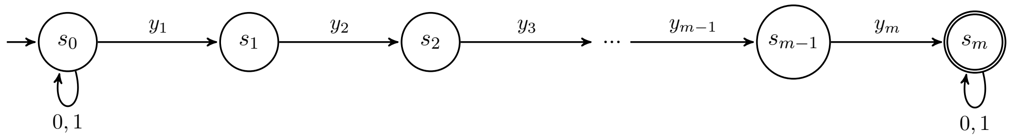
You can submit and check your answers to this question using
https://grinch.cs.washington.edu/cse311/fsm.
In addition, you must submit the documentation of each step of the algorithm, as described above, in **Gradescope**. This documentation is worth 5 points.

5. Irregularity (Due Friday March 13 12:00 PM) (20 points)

Prove that that the set of binary strings of the form $\{0^n 1^m 0^n : m < n\}$ is not regular. (You must use the method described in class for this.)

6. 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:



(where the $\ldots$ 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.