CSE 311: Foundations of Computing I

Homework 8 (due June 6th at 11:00 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. However, you may use results from lecture, the theorems handout, and previous homeworks without proof.*

1. State of Mind [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 with at most three 0s **or** at least four 1s (or both).

(b) [7 Points] The set of binary strings that contain the substring 000 **and** whose third to last digit is 1.

Submit and check your answers to this question here:

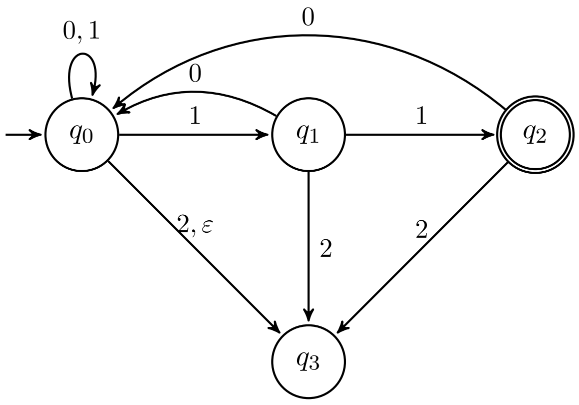
https://grinch.cs.washington.edu/cse311/fsm

Think carefully about your answer to make sure it is correct before submitting. You have only 5 chances to submit a correct answer.

2. Swing States [Online] (15 points)

Use the construction from lecture to convert the following NFA to a DFA.

Label each state of the DFA using corresponding states of the original NFA. 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 set" or something similar.



Submit and check your answers to this question here:

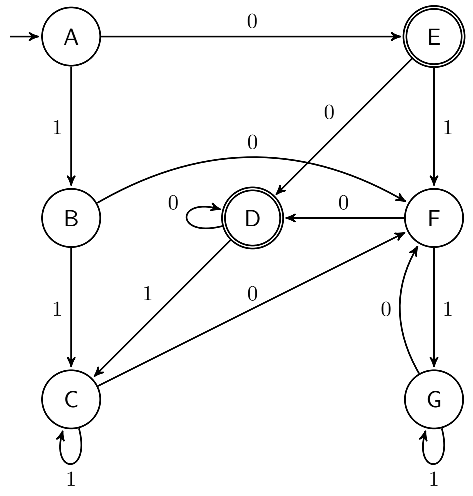
https://grinch.cs.washington.edu/cse311/fsm

Think carefully about your answer to make sure it is correct before submitting. You have only 5 chances to submit a correct answer.

3. Enemy of the State (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 and the reason for splitting that group. At the end, write down the minimized DFA.



4. Regular as Clockwork [Online] (6 points)

Draw an NFA that recognizes the language described by the regular expression $((10)^*(1 \cup 000))^*$. 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

Think carefully about your answer to make sure it is correct before submitting.
You have only 5 chances to submit a correct answer.

5. Just Irregular Guy (40 points)

Use the method described in lecture to prove that each of the following languages is **not regular**.

(a) [18 Points] The set of binary strings of the form $\{0^m 1^n 0^{n+2m} \mid m \geq 0, n > 0\}$.

(b) [20 Points] The set of strings generated by this context-free grammar G:

S $\rightarrow$ **A C** likes tuna fish
A $\rightarrow$ the cat | the dog | the mouse
B $\rightarrow$ chased | bit | sniffed
C $\rightarrow$ **A C B** | ε

This language includes sentences such as "the cat likes tuna fish", "the cat the dog chased likes tuna fish", and "the cat the dog the mouse bit chased likes tuna fish" (i.e., the cat that the dog the mouse bit chased likes tuna fish).

Now, answer the following:

(c) [2 Points] What does part (b) tell us about the English language?

6. A Link to the Past (20 points)

Consider the following definition of a linked list of numbers:

Bases Step: `null` is a **List**.

Recursive Step: If L is a **List** and $x \in \mathbb{R}$, then $\texttt{Node}(x, L)$ is a **List**.

We define the set of values stored in a list as follows:

$$\begin{aligned} \text{values}(\texttt{null}) &= \{\} \\ \text{values}(\texttt{Node}(x, L))) &= \{x\} \cup \text{values}(L) \end{aligned}$$

Finally, we define the function find(v, L), which determines whether the **List** L contains the value $v \in \mathbb{R}$. It is implemented using the structure of the second argument (L) as follows:

$$\begin{aligned} \text{find}(v, \texttt{null}) &= \text{F} \\ \text{find}(v, \texttt{Node}(x, L)) &= \begin{cases} \text{T} & \text{if } x = v \\ \text{find}(v, L) & \text{otherwise} \end{cases} \end{aligned}$$

Prove that this definition of find is correct. Specifically, prove that, for any $v \in \mathbb{R}$ and **List** L, the function find(v, L) returns true if and only if $v \in$ values(L). Your proof should be a structural induction on L.

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

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

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

(where the ... includes states $s_3, \dots, s_{m-2}$). 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 $1 \leq j < m$, 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.