CSE 311: Foundations of Computing I

Homework 8 (due December 10th at 11:00 PM)

Directions: *Write up carefully argued solutions to the following problems. Each solution should be clear enough that it can explain (to someone who does not already understand the answer) why it works. However, you may use results from lecture, the reference sheets, and previous homeworks without proof.*

1. Design Intervention [Online] (15 points)

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

(a) [5 Points] Binary strings with at least two 0s **or** at least two 1s.

(b) [5 Points] Binary strings with at least four 0s **and** end with 010.

Hint: This can be done without the product construction.

(c) [5 Points] Binary strings that **either** have every occurrence of a 0 immediately followed by a 1 **or** contain at least two 1s **but not both**.

Submit and check your answers to this question here:

https://grin.cs.washington.edu

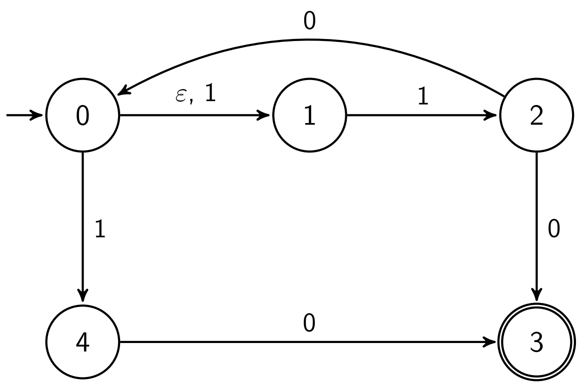
Think carefully about your answer to make sure it is correct before submitting. You have only 5 chances to submit a correct answer.

2. I've Got a Machine Headache [Online] (15 points)

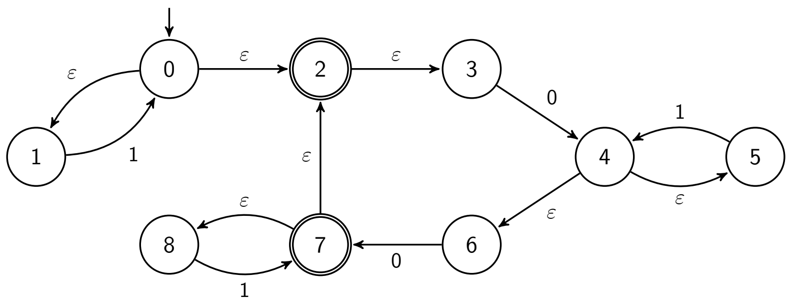
Use the algorithm from lecture to convert each of the following NFAs to DFAs. Label each DFA state with the set of NFA states it represents in the powerset construction.

(a) [5 Points]

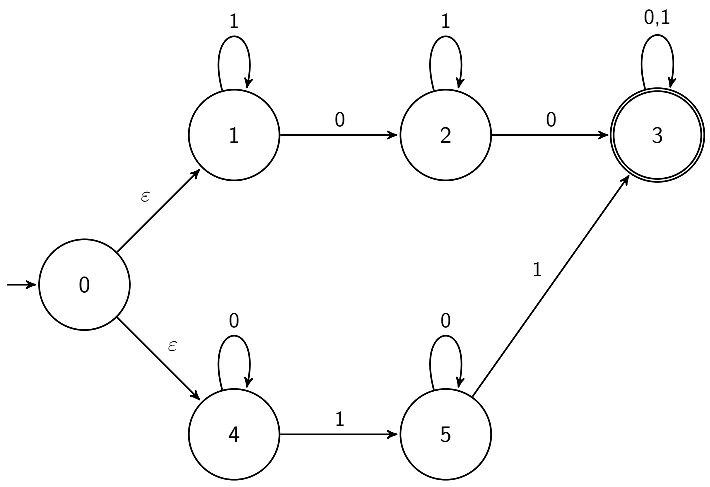
The NFA below:



(b) [5 Points] The NFA below, which is a slightly simplified version of the one that is produced by the construction described in class on the regular expression $1^*(01^*01^*)^*$:



(c) [5 Points] The NFA below, which is another way to match the language from HW7 Problem 5e.



Submit and check your answers to this question here:

https://grin.cs.washington.edu

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

3. Expression Is the Better Part of Valor (10 points)

Use the algorithm from lecture to convert each of the following regular expressions into NFAs that accept the same language. You may skip adding ε-transitions for concatenation if they are *obviously* unnecessary, but otherwise, you should **precisely** follow the construction from lecture.

(a) [5 Points] $1(0 \cup 111)^* \cup 000$

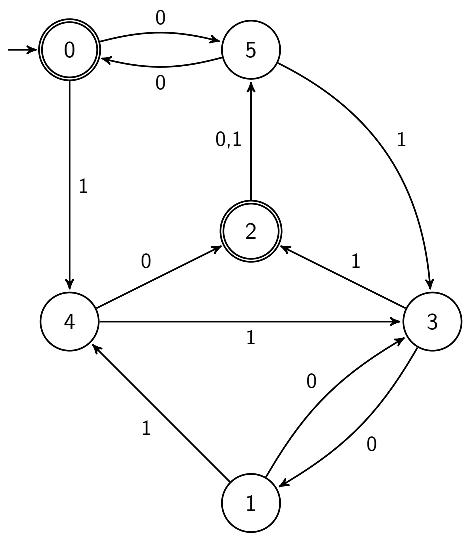
(b) [5 Points] $((1 \cup 00)^*11)^*$

4. A Whole New Small Game (16 points)

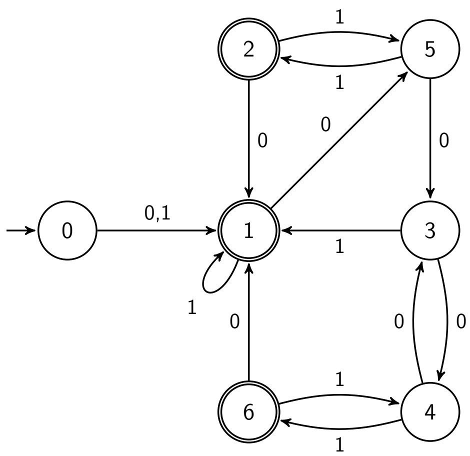
Use the algorithm from lecture to minimize the each of the following DFAs.

For each step of the algorithm, write down the groups of states, which group was split in that step and the reason for splitting that group. At the end, write down the minimized DFA, with each state named by the set of states of the original machine that it represents (e.g., "B, C" if it represents B and C).

(a) [8 Points]



(b) [8 Points]



5. Just Irregular Guy (20 points)

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

(a) [10 Points] All binary strings in the set $\{0^m 1^n 0^{2n+m} : m, n \geq 0\}$.

(b) [10 Points] All binary strings of the form $x\#y$, with $x, y \in \{0,1\}^*$ and x a subsequence of y^R. (Here y^R means the reverse of y. Also, a string w is a subsequence of another string z if you can delete some characters from z to arrive at w.)

The next problem reuses definitions from lecture and prior assignments. We repeat them here for convenience.

We defined lists of numbers in lecture as a recursively defined set:

Basis Step: $\texttt{nil} \in$ **List**
Recursive Step: for any $a \in \mathbb{Z}$, if $L \in$ **List**, then $a :: L \in$ **List**.

We defined the function concat in Homework 6. It concatenates two lists into a single list and was defined recursively as follows:

$$\begin{aligned} \text{concat}(\texttt{nil}, R) &::= R && \forall R \in \textbf{List} \\ \text{concat}(a :: L, R) &::= a :: \text{concat}(L, R) && \forall a \in \mathbb{Z}, \forall L, R \in \textbf{List} \end{aligned}$$

We defined the function rev in Homework 7. It reverses a list and is defined recursively like this:

$$\begin{aligned} \text{rev}(\texttt{nil}) &::= \texttt{nil} \\ \text{rev}(a :: L) &::= \text{concat}(\text{rev}(L), a :: \texttt{nil}) && \forall a \in \mathbb{Z}, \forall L \in \textbf{List} \end{aligned}$$

6. Up the Ladder to the Proof (24 points)

(a) [4 Points] Warmup: Prove, in English, that for all $a \in \mathbb{Z}$, the equation $\text{rev}(a :: \texttt{nil}) = a :: \texttt{nil}$ holds.

Hint: Your proof should primarily consist of a "calculation", i.e., a sequence of equations that are each justified by a definition.

Hint: You are trying to prove a "for all" claim.

In lecture, we defined palindromes recursively as a subset of strings but also said that they are strings that "read the same forward and backward". In the remainder of this problem, we will prove one direction of that claim.

To make it easier, however, we will make this a statement about lists rather than strings. (As noted in lecture, our definitions of lists and strings are almost perfectly analogous.) We define the palindromic lists, **Pal**, recursively as follows:

Basis Step: $\texttt{nil} \in$ **Pal** and $a :: \texttt{nil} \in$ **Pal** for every $a \in \mathbb{Z}$
Recursive Step: for any $a \in \mathbb{Z}$, if $L \in$ **Pal**, then $\text{concat}(a :: L, a :: \texttt{nil}) \in$ **Pal**.

(This is just a translation of our recursive definition of palindromes into list notation.)

(b) [4 Points] Prove, in English, that $1 :: 2 :: 1 :: \texttt{nil} \in$ **Pal**.

Note: You should only need to cite the definitions of **Pal** and concat in your proof. When you cite the definition of **Pal**, state whether you are using the basis step or recursive step.

(c) [16 Points] Use structural induction to prove that every palindromic list is the same when read forward and backward. Formally, we want a proof of the following:

$$\forall L \in \textbf{Pal}\, (\text{rev}(L) = L)$$

You can use without proof the following fact:

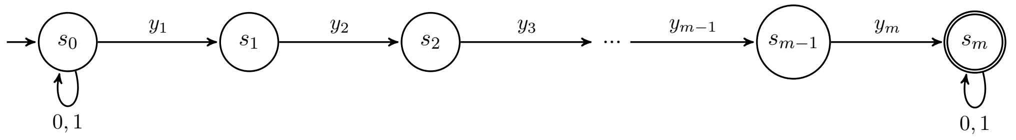
$$\text{Lemma 3: } \text{rev}(\text{concat}(L, R)) = \text{concat}(\text{rev}(R), \text{rev}(L)) \text{ for all } L, R \in \textbf{List}$$

You are also free to cite results from prior homework assignments such as Homework 6 Problem 4, which proved that concat is associative, as well as your result from part (a).

Hint: In the inductive step, make sure you are proving $P(\cdot)$ for the correct list! It is **not** $P(a :: L)$.

7. Extra Credit: Strings to Mind (0 points)

Suppose we want to determine whether a string x of length n contains a string $y = y_1y_2 \dots y_m$ with $m \ll n$. To do so, we construct the following NFA:



(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.