# 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. 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. Feedback to Where You Once Belonged (0 points)

Approximately how much time (in minutes) did you spend on each problem of this homework? Were any problems especially difficult or especially interesting?

## 2. Design Intervention [Online] (10 points)

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

(a) [5 Points] Binary strings with at least three 1s **or** end with 000.

(b) [5 Points] Binary strings with at least three 1s **and** end with 000.

*Hint*: This can be done without the product construction.

Submit and check your answers to this question here:

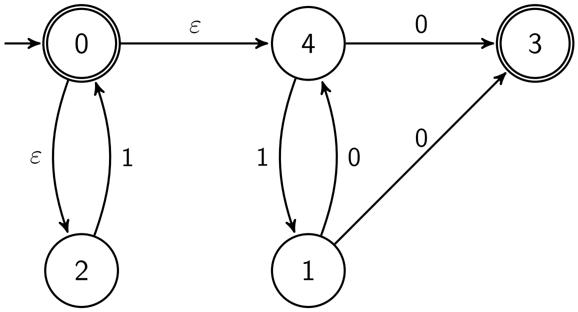
https://grin.cs.washington.edu

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

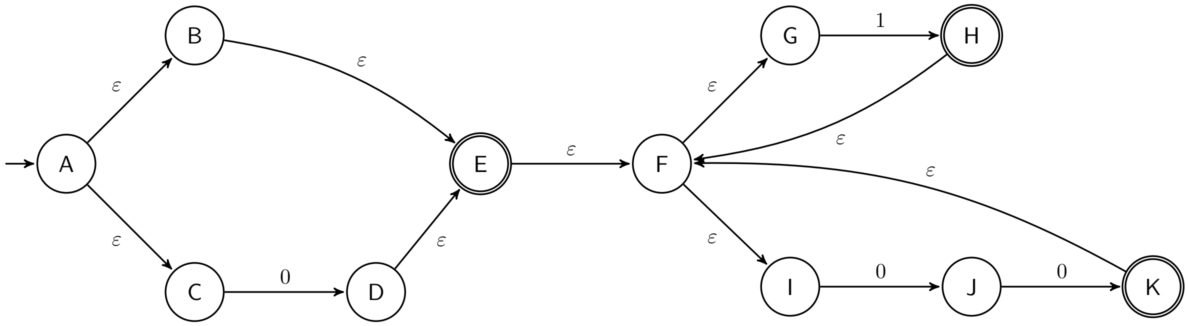
## 3. Get Your Machine Fix [Online] (16 points)

Use the algorithm from lecture to convert each of the following NFAs to DFAs.

(a) [8 Points] The NFA below:



(b) [8 Points] The NFA below, which we get by applying the construction described in class[1] to the regular expression $(\varepsilon \cup 0)(1 \cup 00)^*$:



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 8 chances to submit a correct answer.

## 4. 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 $\varepsilon$-transitions for concatenation if they are *obviously* unnecessary, but otherwise, you should **precisely** follow the construction from lecture.

(a) [5 Points] $11(0 \cup 10)^*11$

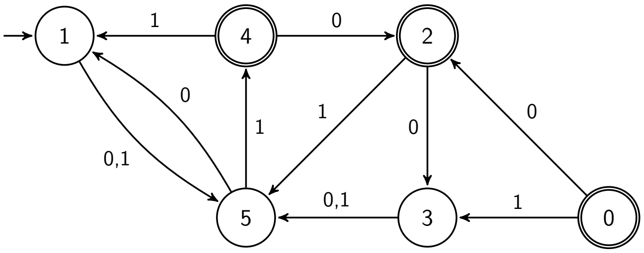
(b) [5 Points] $((0 \cup 10)^*11)^*$

## 5. A Whole New Small Game (16 points)

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

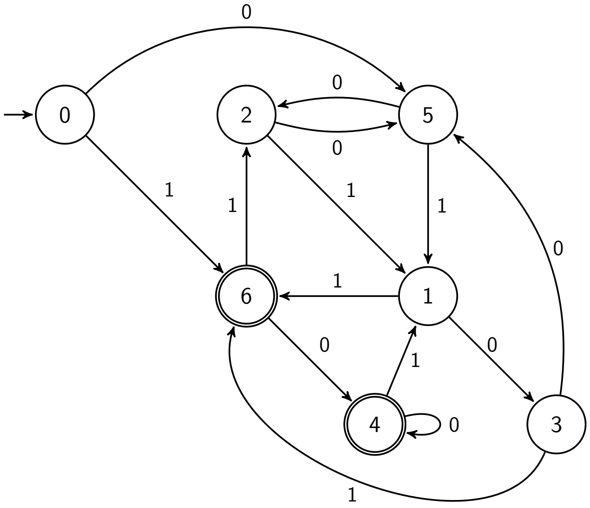
For each step of the algorithm, write down the groups of states, which group was split in that step and the reason for splitting that group. At the end, write down the minimized DFA, with each state named by the set of states of the original machine that it represents (e.g., "$B, C$" if it represents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

(a) [8 Points]



[1] The only simplification performed was using three states rather than four to represent the sub-expression $00$.

(b) [8 Points]



## 6. 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 $\{0^m 1^n 0^{n+2m} : m, n \geq 0\}$.

(b) [10 Points] All strings over $\{0, 1, 2\}$ of the form $x2y$, with $x, y \in \{0, 1\}^*$ and $y$ a subsequence of $x^R$

## 7. Ruled With an Iron List (38 points)

Recall the definition of linked lists of numbers from lecture:

**Bases Step**: $\texttt{null} \in$ **List**
**Recursive Step**: For any $x \in \mathbb{R}$, if $L \in$ **List**, then $\texttt{Node}(x, L) \in$ **List**.

The following parts also refer to the functions concat and rev, which concatenate and reverse these lists, respectively. Those functions were formally defined in HW7 problem 7 (“Up the Ladder to the Proof”).

(a) [20 Points] Prove that $\text{rev}(\text{concat}(L, R)) = \text{concat}(\text{rev}(R), \text{rev}(L))$, for all lists $L$ and $R$, by structural induction on $L$. You may use, without proof, the following facts about concat:

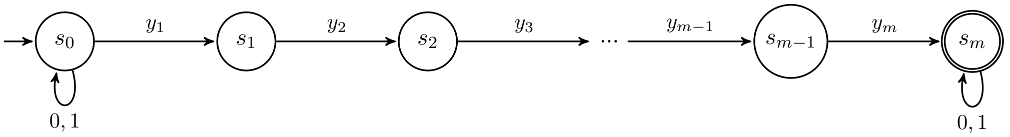
- Identity: $\text{concat}(A, \texttt{null}) = A$ for all $A \in$ **List**
- Associativity: $\text{concat}(\text{concat}(A, B), C) = \text{concat}(A, \text{concat}(B, C))$ for all $A, B, C \in$ **List**

(b) [16 Points] Use part (a) to prove that $\text{rev}(\text{rev}(L)) = L$ for all lists $L$ by structural induction.

(c) [2 Points] Use part (b) to prove that, if $L = \text{rev}(R)$, then $R = \text{rev}(L)$.

## 8. Extra Credit: Strings to Mind (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:



(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.