Homework 8 (due May 27 2020)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough to convince someone who does not already know the answer. You may use results from lecture and previous homeworks without proof. See the syllabus for more details and for permitted resources and collaboration.*

1. Better Folded Than Reversed (20 points)

Consider the set of all linked lists over integers defined as follows:

Basis Step: $\mathsf{null} \in \mathbf{List}$.

Recursive Step: For any $x \in \mathbb{Z}$, if $L \in \mathbf{List}$, then $\mathsf{Node}(x, L) \in \mathbf{List}$.

For example, $\mathsf{Node}(3, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null}))) \in \mathbf{List}$ represents the list $[3, 1, 1]$.

Linked lists are a core data structure in functional programming languages, which also provide common functions for operating on lists, including fold_f, shift, and $\mathsf{reverse}$ defined as follows:

$$\begin{aligned} \mathsf{fold}_f(y, \mathsf{null}) &= y && \text{for any } y \in \mathbb{Z} \\ \mathsf{fold}_f(y, \mathsf{Node}(x, L)) &= f(x, \mathsf{fold}_f(y, L)) && \text{for any } L \in \mathbf{List}, x, y \in \mathbb{Z} \end{aligned}$$

$$\begin{aligned} \mathsf{shift}(\mathsf{null}, R) &= R && \text{for any } R \in \mathbf{List} \\ \mathsf{shift}(\mathsf{Node}(x, L), R) &= \mathsf{shift}(L, \mathsf{Node}(x, R)) && \text{for any } L, R \in \mathbf{List}, x \in \mathbb{Z} \end{aligned}$$

$$\mathsf{reverse}(L) = \mathsf{shift}(L, \mathsf{null}) \qquad \text{for any } L \in \mathbf{List}$$

The function shift prepends all elements in L to R, in the reverse order, and the function $\mathsf{reverse}$ uses shift to implement list reversal. For example:

$$\begin{aligned} \mathsf{reverse}(\mathsf{Node}(3, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null})))) &= \mathsf{shift}(\mathsf{Node}(3, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null}))), \mathsf{null}) \\ &= \mathsf{shift}(\mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null})), \mathsf{Node}(3, \mathsf{null})) \\ &= \mathsf{shift}(\mathsf{Node}(1, \mathsf{null}), \mathsf{Node}(1, \mathsf{Node}(3, \mathsf{null}))) \\ &= \mathsf{shift}(\mathsf{null}, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{Node}(3, \mathsf{null})))) \\ &= \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{Node}(3, \mathsf{null}))). \end{aligned}$$

As before, we would like to optimize our use of the list API to avoid creating intermediate lists. In particular, we want to replace $\mathsf{fold}_f(y, \mathsf{reverse}(L))$ with $\mathsf{fold}_f(y, L)$, whenever the results of these two expressions are guaranteed to be the same.

This problem asks you to show that (a) this optimization is incorrect for some functions f. Next, it asks you to (b) prove a lemma, which will then help you show that (c) the optimization is correct for every binary function f that satisfies two additional properties, commutativity and associativity. Recall that a binary function f is associative iff $f(a, f(b, c)) = f(f(a, b), c)$ for all $a, b, c \in \mathbb{Z}$, and f is commutative iff $f(a, b) = f(b, a)$ for all $a, b \in \mathbb{Z}$. For example, $f(a, b) = a + b$ is both commutative and associative, so you can apply the theorem you will prove in part (c) to replace $\mathsf{fold}_f(y, \mathsf{reverse}(L))$ with just $\mathsf{fold}_f(y, L)$.

(a) [2 points] Prove that $\mathsf{fold}_f(y, \mathsf{reverse}(L)) \neq \mathsf{fold}_f(y, L)$ for some $L \in \mathbf{List}$, $y \in \mathbb{Z}$, and binary function f over integers.

(b) [9 points] Prove that for all $L \in \mathbf{List}$, all $a, b \in \mathbb{Z}$, and all binary functions f over integers, if f is associative and commutative, then $\mathsf{fold}_f(f(a, b), L) = f(a, \mathsf{fold}_f(b, L))$.

(c) [9 points] Prove that for all $L \in \mathbf{List}$, all $y \in \mathbb{Z}$, and all binary functions f over integers, if f is associative and commutative, then $\mathsf{fold}_f(y, \mathsf{reverse}(L)) = \mathsf{fold}_f(y, L)$.

Hint: You will need a stricter induction hypothesis. To obtain the stricter hypothesis, observe that $\mathsf{fold}_f(y, \mathsf{reverse}(L)) = \mathsf{fold}_f(y, \mathsf{shift}(L, \mathsf{null}))$ and $\mathsf{fold}_f(y, L) = \mathsf{fold}_f(\mathsf{fold}_f(y, \mathsf{null}), L)$, so the problem is

asking you to prove that $\mathsf{fold}_f(y, \mathsf{shift}(L, \mathsf{null})) = \mathsf{fold}_f(\mathsf{fold}_f(y, \mathsf{null}), L)$ if f is associative and commutative. Generalize this statement to consider all lists R instead of just null, and argue that this generalization is a stricter hypothesis for a proof by structural induction. The result from part (b) can help you justify a step in your proof of the stricter hypothesis.

2. Functional Relationships (15 points)

A relation $R \subseteq A \times B$ is **functional** if and only if it satisfies the following property:

$$\forall (a, b_1) \in A \times B. \forall (a, b_2) \in A \times B. ((a, b_1) \in R \wedge (a, b_2) \in R) \rightarrow b_1 = b_2$$

A function $f : A \rightarrow B$ takes inputs from the set A and outputs an element in B, meaning for $a \in A$, $f(a) = b \in B$. From a function f we can define the functional relation $R_f = \{(a, f(a)) : a \in A\}$.

For each of the following questions about functional relations, answer the question (yes or no), and *prove* that your answer is correct. All of your proofs for this question should be short (one paragraph).

(a) [5 points] Does there exist a functional relation $R \subseteq A \times A$ such that R is transitive and R contains the tuples $(a, b) \in R$ and $(b, c) \in R$ for distinct $a, b, c \in A$?

(b) [5 points] We say that a functional relation is one-to-one if $\forall (a_1, f(a_1)) \in R_f.\ \forall (a_2, f(a_2) \in R_f.\ f(a_1) = f(a_2) \rightarrow a_1 = a_2$. Does there exist a function f over integers, $f : \mathbb{Z} \rightarrow \mathbb{Z}$, that defines an antisymmetric and one-to-one functional relation R_f?

(c) [5 points] Does there exist a function f over integers, $f : \mathbb{Z} \rightarrow \mathbb{Z}$, that defines a functional relation R_f such that $R_f^2 = R_f^1$ and $R_f^1 \cap R_f^0 = \emptyset$? (Recall that R^0 for every relation R on integers is $\{(a, a) : a \in \mathbb{Z}\}$.)

3. Scheduling with Precedence Constraints (20 points)

Suppose there is a set of jobs J, and each job takes some amount of time to process, denoted p_j (**processing time**) for $j \in J$. Given a set of machines that can process all the jobs, the goal is to create a schedule for assigning jobs to machines in order to complete the jobs as quickly as possible. A job cannot be moved from a machine once it is scheduled on a machine (this is called non-migratory), meaning a job cannot be processed partially on machine 1 and then moved to machine 2.

For instance, if we want to process jobs $J = \{j_1, j_2, j_3\}$ on 2 machines with $p_{j_1} = 1$, $p_{j_2} = 2$, and $p_{j_3} = 1$, then we can schedule j_1 then j_3 on machine 1 and j_2 on machine 2 to complete all the jobs in total time 2.

Sometimes certain jobs in J cannot be scheduled until other jobs have been completed. For example, this is extremely common in the parallelization of Deep Neural Network training. We say that there is a **precedence constraint** from j_1 to j_2, denoted $j_1 \prec j_2$, if no machine can start processing j_2 until j_1 is completely processed. We call j_1 a **predecessor** of j_2 and j_2 a **successor** of j_1. We model the set of jobs and its precedence constraints as a binary relation.

For instance, if we want to process jobs $J = \{j_1, j_2, j_3\}$ on 2 machines with $p_{j_1} = 1$, $p_{j_2} = 2$, and $p_{j_3} = 1$, and further $j_2 \prec j_3$, then if we still scheduled j_1 then j_3 on machine 1 and j_2 on machine 2, the completion time is $2 + 1 = 3$ because j_3 must wait until j_2 finished.

Let $J = \{j_1, j_2, j_3, j_4, j_5, j_6, j_7, j_8, j_9\}$ be a set of 9 jobs. We will refer to jobs by their index $1, \cdots, 9$. In this problem you will consider the following specification for precedence constraints on J. There are no other precedence constraints other than the ones listed below:

- Every job with even index has j_1 as a predecessor.
- Every job with index congruent to 2 mod 5 is a successor of j_3.
- The only predecessor of j_5 is the job whose index is the multiplicative inverse of 5 mod 9.
- j_3 and j_7 are predecessors of j_8.

- j_9 is the only successor of j_5.

(a) [10 points] Write down the relation R where $(j_i, j_k) \in R$ if and only if $j_i \prec j_k$.

(b) [5 points] Assume we can process J on 3 machines and all jobs have processing time $p_j = 1$. Give a schedule for the jobs that finishes as fast as possible (i.e., takes the least total time). You do not need to prove anything here. *Hint:* Write out the directed graph for the relation R. This will make it easier to see the answer.

(c) [5 points] Prove that your schedule in (b) is as fast as possible. *Hint:* Write out the elements of R^2, R^3, and R^k for $k \geq 4$. Then, use the elements of R^3 and R^k for $k \geq 4$ in your proof.

4. Designing DFAs [Online] (15 points)

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

(a) [5 points] Binary strings with at least two 1s.

(b) [5 points] Binary strings that have at least one 1 and an even number of 0s.

(c) [5 points] Binary strings such that none of their *runs* of 1s have odd length. A run of 1s is a substring consisting of all 1s (i.e. "$11\cdots1$") that is either at the beginning of the string, or at the end of the string, or in the middle of the string, surrounded by 0s. For example, the following strings are included in the language: "11", "110", "00001111", and "011001111011". But the following strings are *not* included in the language because they all contain at least one run of 1s with odd length: "111", "010", and "00110111".

Submit and check your answers here:

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

You **must also** submit screenshots of your answers in Gradescope, and mark them as the solution to this problem. You have only **5 chances** to submit a correct answer.

5. Designing NFAs [Online] (15 points)

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

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

(b) [8 points] The set of binary strings that contain 11 or do not contain 00.

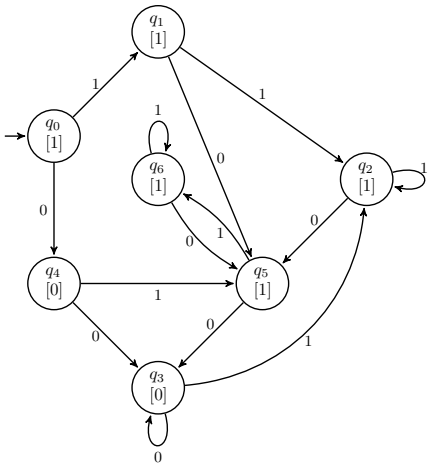
Submit and check your answers here:

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

You **must also** submit screenshots of your answers in Gradescope, and mark them as the solution to this problem. You have only **5 chances** to submit a correct answer.

6. DFA Minimization [Online] (15 points)

Consider the following automaton:



(a) [5 points] Use the algorithm for minimization that we discussed in class to minimize the above 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. See the solution to Problem 6 of Section 8 for an example of how to write down the steps and the explanations.

(b) [10 points]

Submit and check your minimized DFA here:
https://grinch.cs.washington.edu/cse311/fsm
You **must also** submit a screenshot of your DFA in Gradescope, and mark it as the solution to this problem. You have only **5 chances** to submit a correct answer.