Homework 9 (Final) (due June 5 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. Formal Proofs and Truth Tables (10 points)

Prove the following using logical equivalences and a truth table. $\neg(p \rightarrow (p \wedge \neg q)) \equiv p \wedge q$.

2. Logical Surjections (15 points)

For each of the following statements about functions: (1) translate the statement to predicate logic, (2) say whether the statement is true or false, and (3) prove that your answer is correct.

For the logic translations, let the domain of discourse be functions $f : A \rightarrow B$ of one variable. Recall that a function is a relation that maps each element in A to one element in B. You will be translating statements about *surjective* functions. We say a function $f : A \rightarrow B$ is surjective (onto) if and only if for every $b \in B$, there is an $a \in A$ such that $f(a) = b$. The set A is called the domain of f, and $f(A) = \{b \in B : \exists a \in A.f(a) = b\}$ is the range of f. Note that $f(A) \subseteq B$, and by definition, f is surjective onto its range.

Your translations should use only relational operators (e.g., composition $\circ$); the operator $|A|$ to represent the size, or cardinality, of a set A; the predicate $\mathsf{Surj}(f)$, which says whether a function f is surjective; and the predicates $=, \geq, \leq$ in regards to set equality and set sizes.

(a) [5 points] Not every function $f : A \rightarrow A$ whose domain and range are the same is surjective.

(b) [5 points] The composition of surjective functions $f : B \rightarrow C$ and $g : A \rightarrow B$ is surjective.

(c) [5 points] The domain of a surjective function is no larger than its range.

3. Define the Relationship (15 points)

For $m \in \mathbb{N} \setminus \{0\}$, define the relation C_m on $\mathbb{Z}$ by $(a, b) \in C_m$ iff $a \equiv b \pmod{m}$. State whether each of the following are True or False, and prove your answer.

(a) [5 points] If $(a, b) \in C_m$ then $(-a, b) \in C_m$.

(b) [5 points] $C_m^* = C_m$. Recall C_m^* is the reflexive transitive closure of C_m.

(c) [5 points] State whether or not C_m is reflexive, symmetric, antisymmetric, and transitive. Justify your answer briefly. (You may use any of your previous parts.)

4. Many Trees (10 points)

Let S be the set of all rooted binary trees:

Basis Step: $\bullet \in S$.

Recursive Step: If $L, R \in S$, then $\mathsf{Tree}(\bullet, L, R) \in S$.

Observe that the contents of the set S after n applications of the recursive rule can be represented as $S_n = S_{n-1} \cup \{\mathsf{Tree}(\bullet, L, R) : L, R \in S_{n-1}\}$, where $S_0 = \{\bullet\}$.

Next, let T be the following recursively defined function:

$$\begin{array}{rcll} T(0) & = & 1 & \\ T(n) & = & T(n-1)^2 + 1 & \text{for } n \geq 1 \end{array}$$

Prove that $T(n) = |S_n|$ where $|S_n|$ is the number of trees in the set S_n for all $n \geq 0$.

5. Logical Trees (15 points)

Let $\mathcal{V} = \{\ldots, a, b, c, \ldots\}$ be a fixed set of boolean variables. We then define the set **Expr** as follows:

Basis Step: For any $v \in \mathcal{V}$, $\mathsf{Var}(v) \in$ **Expr**.

Recursive Step: If $s, t \in$ **Expr**, then $\mathsf{Not}(s) \in$ **Expr**, $\mathsf{And}(s,t) \in$ **Expr**, and $\mathsf{Or}(s,t) \in$ **Expr**.

The set **Expr** represents parse trees of boolean algebra expressions that use only negation, conjunction, and disjunction. For example, $\mathsf{And}(\mathsf{Var}(a), \mathsf{Or}(\mathsf{Not}(\mathsf{Var}(b)), \mathsf{Var}(c)))$ represents the expression $a \cdot (b' + c)$.

Next, we define the function Eval_V that takes as input a parse tree from **Expr**, and returns the value of the boolean algebra expression that the tree represents when all variables in the set $V \subseteq \mathcal{V}$ have the value 1, and all variables not in V have the value 0.

$$\begin{array}{rcll}
\mathsf{Eval}_V(\mathsf{Var}(v)) & = & \text{if } v \in V \text{ then 1 else 0} & \text{for any } v \in \mathcal{V} \text{ and } V \subseteq \mathcal{V} \\
\mathsf{Eval}_V(\mathsf{Not}(s)) & = & \mathsf{Eval}_V(s)' & \text{for any } s \in \textbf{Expr} \\
\mathsf{Eval}_V(\mathsf{And}(s,t)) & = & \mathsf{Eval}_V(s) \cdot \mathsf{Eval}_V(t) & \text{for any } s, t \in \textbf{Expr} \\
\mathsf{Eval}_V(\mathsf{Or}(s,t)) & = & \mathsf{Eval}_V(s) + \mathsf{Eval}_V(t) & \text{for any } s, t \in \textbf{Expr}
\end{array}$$

For example, $\mathsf{Eval}_{\{a\}}(\mathsf{And}(\mathsf{Var}(a), \mathsf{Or}(\mathsf{Not}(\mathsf{Var}(b)), \mathsf{Var}(c)))) = 1 \cdot (0' + 0) = 1 \cdot (1 + 0) = 1$.

Finally, we define the functions f and g as follows:

$$\begin{array}{rcll}
f(\mathsf{Var}(v), 0) & = & \mathsf{Not}(\mathsf{Var}(v)) & \text{for any } v \in \mathcal{V} \\
f(\mathsf{Var}(v), 1) & = & \mathsf{Var}(v) & \text{for any } v \in \mathcal{V} \\
f(\mathsf{Not}(s), 0) & = & f(s, 1) & \text{for any } s \in \textbf{Expr} \\
f(\mathsf{Not}(s), 1) & = & f(s, 0) & \text{for any } s \in \textbf{Expr} \\
f(\mathsf{And}(s,t), 0) & = & \mathsf{Or}(f(s,0), f(t,0)) & \text{for any } s, t \in \textbf{Expr} \\
f(\mathsf{And}(s,t), 1) & = & \mathsf{And}(f(s,1), f(t,1)) & \text{for any } s, t \in \textbf{Expr} \\
f(\mathsf{Or}(s,t), 0) & = & \mathsf{And}(f(s,0), f(t,0)) & \text{for any } s, t \in \textbf{Expr} \\
f(\mathsf{Or}(s,t), 1) & = & \mathsf{Or}(f(s,1), f(t,1)) & \text{for any } s, t \in \textbf{Expr} \\
g(s) & = & f(s, 1) & \text{for any } s \in \textbf{Expr}
\end{array}$$

The function f takes as input a parse tree and a boolean value (0 or 1), and returns a parse tree. The function g is defined in terms of f; it takes a parse tree and returns a parse tree. For example, $g(\mathsf{Not}(\mathsf{Not}(\mathsf{Var}(a)))) = \mathsf{Var}(a)$. Understanding what these functions do is essential for solving this problem.

This problem asks you to prove two properties of the function g over parse trees, (b) and (c). Part (a) asks you to prove a lemma that will help you complete the proof in part (b). You can use the result from part (b) to prove part (c), but note that (c) is challenging. It is worth only 1 point, so do **not** attempt it unless you have finished the rest of the assignment and want a challenge!

(a) [10 points] Prove that $g(g(s)) = g(s)$ for every $s \in$ **Expr**.
Hint: You will need a stricter induction hypothesis. Recall how we obtained such a hypothesis for the Better Folded Than Reversed problem from HW8—expand the definition of g, generalize the resulting statement, and argue that it implies the desired result.

(b) [4 points] Let $F = \{t \in \textbf{Expr} : g(t) = t\}$. Prove that for every $s \in$ **Expr**, we have $g(s) \in F$.

(c) [1 point] Prove that for every $s \in$ **Expr**, there is a $t \in F$ such that $\mathsf{Eval}_V(s) = \mathsf{Eval}_V(t)$ for every $V \in \mathcal{P}(\mathcal{V})$.
(This problem is challenging; do not attempt it before finishing the rest of the assignment.)

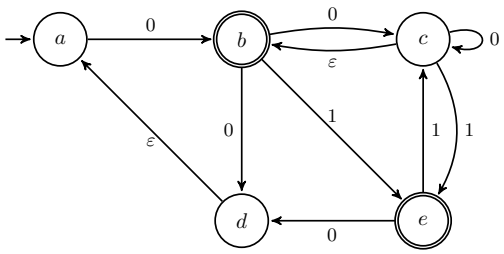
6. Grammatical Trees (10 points)

Write a context-free grammar that generates exactly the set F from Problem 5b. In other words, your grammar should generate the parse tree s if and only if $s \in F$.

Briefly explain why your solution is correct. If your grammar has more than one variable, write a sentence describing what sets of parse trees you expect each variable in your grammar to generate. No proof is needed.

7. Not So Nondeterministic [Online] (15 points)

Use the construction from lecture to convert the following NFA to a DFA. Label each state of the DFA using appropriate states of the original NFA.



Submit and check your answer here:

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

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

8. Regular or Not, That is the Final Question (10 points)

Let 1^* be the regular language of all strings consisting of only ones. Does there exist a set of strings $L \subseteq 1^*$ such that L is an irregular language? Prove that your answer is correct.