Homework 7 (due May 20 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. All the Recursive Strings (15 points)

For each of the following, write a recursive definition of the set of strings satisfying the given properties. Your basis step must explicitly enumerate a finite number of initial elements. Use the smallest basis set and the smallest number of recursive rules possible. *Briefly* justify that your solution is correct; no proof is required. You do not have to justify why your answer is the smallest in either way.

(a) [5 points] Binary strings that start with 0 and have odd length.

(b) [5 points] Binary strings with an even number of 1s.

(c) [5 points] Binary strings $x \in \{0,1\}^*$ whose length $\mathsf{len}(x)$ satisfies $\mathsf{len}(x) \equiv 2 \pmod 3$.

2. An Uphill Manhattan Walk (12 points)

Let S be a subset of $\mathbb{Z} \times \mathbb{Z} = \mathbb{Z}^2$ defined as:

Basis Step: $(0,0) \in S$

Recursive Step: If $(a,b) \in S$ then $(a, b+1) \in S$, $(a+1, b+1) \in S$, and $(a+2, b+1) \in S$.

Prove that $\forall (a,b) \in S, a \leq 2b$.

3. Expressive Trees (25 points)

Let x be a variable and define the set **Expr** as follows:

Basis Step: $\mathsf{Var}(x) \in$ **Expr**, $\mathsf{Int}(4) \in$ **Expr**, and $\mathsf{Int}(9) \in$ **Expr**.

Recursive Step: For any $s, t \in$ **Expr**, we have $\mathsf{Add}(s,t) \in$ **Expr** and $\mathsf{Sub}(s,t) \in$ **Expr**.

The set **Expr** represents parse trees of arithmetic expressions over the variable x and integers 4 and 9 that use only addition and subtraction. For example, $\mathsf{Add}(\mathsf{Var}(x), \mathsf{Int}(4))$ represents the expression $x + 4$.

We also define the function Eval_v that takes a parse tree (an element of **Expr**) and returns the value of the expression that the tree represents when x has the value $v \in \mathbb{Z}$:

$$\begin{array}{rcll} \mathsf{Eval}_v(\mathsf{Int}(w)) & = & w & \text{for any } w \in \{4, 9\} \\ \mathsf{Eval}_v(\mathsf{Var}(x)) & = & v & \\ \mathsf{Eval}_v(\mathsf{Add}(s,t)) & = & \mathsf{Eval}_v(s) + \mathsf{Eval}_v(t) & \text{for any } s, t \in \textbf{Expr} \\ \mathsf{Eval}_v(\mathsf{Sub}(s,t)) & = & \mathsf{Eval}_v(s) - \mathsf{Eval}_v(t) & \text{for any } s, t \in \textbf{Expr} \end{array}$$

For example, $\mathsf{Eval}_{307}(\mathsf{Add}(\mathsf{Var}(x), \mathsf{Int}(4)) = \mathsf{Eval}_{307}(\mathsf{Var}(x)) + \mathsf{Eval}_{307}(\mathsf{Int}(4)) = 307 + 4 = 311$.

This problem asks you to prove two properties of parse trees, (a) and (c), and based on those proofs, state a theorem about them (d). Part (b) asks you to prove a lemma that will help you complete the proof in part (c).

(a) [8 points] Prove that for any $e \in$ **Expr**, there are integers A and B such that for any integer v, $\mathsf{Eval}_v(e) = A \cdot v + B$.

(b) [8 points] Consider the recursive function Mul that constructs a parse tree given an integer $n > 0$ and a parse tree $e \in$ **Expr**:

$$\begin{array}{rcll} \mathsf{Mul}(1, e) & = & e & \text{for any } e \in \textbf{Expr} \\ \mathsf{Mul}(n, e) & = & \mathsf{Add}(e, \mathsf{Mul}(n-1, e)) & \text{for any } n \geq 2, e \in \textbf{Expr} \end{array}$$

Prove that $\mathsf{Eval}_v(\mathsf{Mul}(n, e)) = n \cdot \mathsf{Eval}_v(e)$ for every parse tree $e \in$ **Expr**, integer $n > 0$, and integer v.

(c) [8 points] Prove that for any integers A and B, there is a parse tree $e \in$ **Expr** such that for any integer v, $\mathsf{Eval}_v(e) = A \cdot v + B$. *Hint:* You may want to use the result from (b) for this proof.

(d) [1 point] What is the strongest theorem you can state about the relationship between the set of parse trees **Expr** and arithmetic expressions of the form $Ax + B$, where A and B are integers?

4. Better Folded Than Appended (20 points)

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

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

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

For example, $\mathsf{Node}(3, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null}))) \in$ **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 append and fold_f, defined as follows:

$$\begin{array}{rcll} \mathsf{append}(\mathsf{null}, R) & = & R & \text{for any } R \in \textbf{List} \\ \mathsf{append}(\mathsf{Node}(x, L), R) & = & \mathsf{Node}(x, \mathsf{append}(L, R)) & \text{for any } L, R \in \textbf{List}, x \in \mathbb{Z} \end{array}$$

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

Here, append concatenates two lists L and R to create a new list that has all the elements of L, followed by all the elements of R. The expression $\mathsf{fold}_f(y, L)$ computes the result of repeatedly applying a binary function $f(a, b)$, where a and b are integer arguments, to the elements of a list L and a value y. For example, if L is the list containing the number $x_1, \ldots, x_n$, then $\mathsf{fold}_f(y, L) = f(x_1, f(x_2, f(\ldots f(x_n, y)\ldots)))$.

When writing programs that use these operations, we want to avoid creating intermediate lists to get the best performance. For example, instead of writing $\mathsf{fold}_f(y, \mathsf{append}(L, R))$, we prefer to write $\mathsf{fold}_f(\mathsf{fold}_f(y, R), L)$. This problem asks you to prove that this optimization is correct but one that looks very similar is not!

(a) [2 points] Use the definitions to calculate $\mathsf{append}(L, R)$ where $L = \mathsf{Node}(7, \mathsf{Node}(5, \mathsf{null}))$ and $R = \mathsf{Node}(3, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null})))$. Show your work.

(b) [2 points] Let f be the function defined by $f(a, b) = a + b$. Use the definitions to calculate $\mathsf{fold}_f(0, L)$ where $L = \mathsf{Node}(3, \mathsf{Node}(1, \mathsf{Node}(1, \mathsf{null})))$. Show your work.

(c) [11 points] Prove that $\mathsf{fold}_f(y, \mathsf{append}(L, R)) = \mathsf{fold}_f(\mathsf{fold}_f(y, R), L)$ for all $L, R \in$ **List**, all $y \in \mathbb{Z}$, and all binary functions f over integers.

(d) [5 points] Prove that $\mathsf{fold}_f(y, \mathsf{append}(L, R)) \neq \mathsf{fold}_f(\mathsf{fold}_f(y, L), R)$ for some $L, R \in$ **List**, $y \in \mathbb{Z}$, and binary function f over integers.

5. Constructing Regular Expressions [Online] (16 points)

For each of the following, construct regular expressions that match the given set of strings:

(a) [4 points] Binary strings where **every** occurrence of a 1 is immediately followed by a 0.

(b) [4 points] Binary strings where **no** occurrence of a 00 is immediately followed by a 1.

(c) [4 points] The set of all binary strings that contain at least one 1 and at most two 0's.

(d) [4 points] The set of all binary strings that begin with a 1 and have length congruent to 2 mod 4

Submit and check your answers to this question here:
https://grinch.cs.washington.edu/cse311/regex
You **must also** submit a screenshot of your submitted regexes in Gradescope, and mark it as the solution to this problem. Think carefully about your answer to make sure it is correct before submitting. Do not include blanks since these count as characters. You have only **5 chances** to submit a correct answer.

6. All the Grammatical Strings (12 points)

For each of the following, construct context-free grammars that generate the given set of strings. If your grammar has more than one variable, we will ask you to write a sentence describing what sets of strings you expect each variable in your grammar to generate.

For example, if your grammar were:

$$\begin{aligned}\mathbf{S} &\to \mathbf{E} \mid \mathbf{O}\\ \mathbf{E} &\to \mathbf{EE} \mid \mathbf{CC}\\ \mathbf{O} &\to \mathbf{EC}\\ \mathbf{C} &\to 0 \mid 1\end{aligned}$$

We would expect you to say "E generates (non-empty) even length binary strings; O generates odd length binary strings; C generates binary strings of length one."

(a) [4 points] The same set of strings as in 5c—the set of all binary strings that contain at least one 1 and at most two 0's.

(b) [4 points] $\{1^m 0^n 1^{m+n} : m, n \geq 0\}$

(c) [4 points] Binary strings with an odd number of 0s.