CS 13: Mathematical Foundations of Computer Science Fall 2023

Written Homework 02 (due Monday, October 30)

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. You may use results from lecture and previous homeworks without proof. Your solutions* must *be written in LaTeX using our homework template.* ***No solution to a single part may be more than one page.***

0. Lists without Lisp! (50 points)

Define two new programs on Lists. concat : (List, List) → List and rev : List → List:

$$\begin{aligned} \text{concat}(\texttt{[]}, R) &= R & \text{rev}(\texttt{[]}) &= \texttt{[]} \\ \text{concat}(x :: L, R) &= x :: \text{concat}(L, R) & \text{rev}(x :: X) &= \text{concat}(\text{rev}(X), x :: \texttt{[]}) \end{aligned}$$

In this question, you will prove that $\text{rev}(\text{concat}(A, B)) = \text{concat}(\text{rev}(B), \text{rev}(A))$ for all lists A and B.

(a) [15 Points] Prove that concat is symmetric across `[]`. That is, prove that for all lists L,

$$\text{concat}(L, \texttt{[]}) = \text{concat}(\texttt{[]}, L)$$

(b) [15 Points] Prove that for all lists A, B, C, concat is associative. That is:

$$\text{concat}(\text{concat}(A, B), C) = \text{concat}(A, \text{concat}(B, C))$$

(c) [20 Points] In this part, you will (finally!) prove that $\text{rev}(\text{concat}(A, B)) = \text{concat}(\text{rev}(B), \text{rev}(A))$ for all lists A and B.

To do this, you should first let B be an arbitrary list, and then go by structural induction on A. You should find both of the previous parts useful (and you may use them as lemmas even if you haven't proven them correctly.)

1. Proving BST Insertion Works! (50 points)

Consider the following definition of **Tree**:

$$\textbf{Tree} = \texttt{Nil} \mid \texttt{Tree}(\textbf{Integer}, \textbf{Tree}, \textbf{Tree})$$

Then, the standard *BST insertion* function can be written as the following:

$$\begin{aligned} \text{insert}(v, \texttt{Nil}) &= \texttt{Tree}(v, \texttt{Nil}, \texttt{Nil}) \\ \text{insert}(v, \texttt{Tree}(x, L, R))) &= \texttt{if } v < x \texttt{ then } \texttt{Tree}(x, \text{insert}(v, L), R) \texttt{ else } \texttt{Tree}(x, L, \text{insert}(v, R)) \end{aligned}$$

(a) [25 Points]

Next, define a program less which checks if an entire tree is less than a provided integer:

$$\begin{aligned} \text{less}(v, \texttt{Nil}) &= \texttt{true} \\ \text{less}(v, \texttt{Tree}(x, L, R)) &= x < v \texttt{ and } \text{less}(v, L) \texttt{ and } \text{less}(v, R) \end{aligned}$$

Prove that for all $b \in \mathbb{Z}$, $v \in \mathbb{Z}$ and all trees T, if $\text{less}(b, T)$, and $b > v$, then $\text{less}(b, \text{insert}(v, T))$. You may assume that all elements inserted into the tree are *unique*.

In English, this means that, given a upper bound on the elements in a BST, if you insert something that meets that upper bound, it is still a upper bound.

(b) [25 Points] Consider a similar function greater which can be used to establish an *lower bound* on the elements in the tree:

$$\begin{aligned}
\text{greater}(v, \texttt{Nil}) &= \texttt{true} \\
\text{greater}(v, \texttt{Tree}(x, L, R)) &= x > v \texttt{ and } \text{greater}(v, L) \texttt{ and } \text{greater}(v, R)
\end{aligned}$$

You may assume the symmetric theorem for greater that you proved in (a).

Now, consider a function that checks if a tree is a BST:

$$\begin{aligned}
\text{isBST}(\texttt{Nil}) &= \texttt{true} \\
\text{isBST}(\texttt{Tree}(x, L, R)) &= \text{less}(x, L) \texttt{ and } \text{isBST}(L) \texttt{ and } \text{greater}(x, R) \texttt{ and } \text{isBST}(R)
\end{aligned}$$

Using part (a) and the symmetric theorem for greater, prove that for all trees T and all $v \in \mathbb{Z}$, if isBST(T), then isBST(insert(v, T)). That is, prove that insertion into a BST preserves the BST property!