CS:5810 Formal Methods in Software Engineering
Fall 2025

Homework 3

Due: Friday, Nov 21, at 11:59pm

This assignment is to be done *individually*.

This assignment is divided into two parts which can be done independently. It builds on material from Chapters 8, 11-13 of the textbook. Download the accompanying files `hw3a.dfy` and `hw3b.dfy`, type your name and your solutions in those files where indicated, and submit it on ICON.

Be sure to review the syllabus for details about this course's cheating policy.

This assignment will test your ability to write programs in the Dafny language, annotate them with contracts, and check their correctness. Use Dafny to verify that your programs are correct with respect to their specifications.

Part A

File `hw3a.dfy` contains a number of functions over ordered integer lists and of *ordered* trees. These as similar to binary search trees but maintain the invariant that the value stored in the root is smaller than or equal to all the values stored in its subtrees.

The file provides a number of functions over lists, most of which you are already familiar with. The new ones are the following:

- listCount, which takes a list of elements of some type and a value of that type, and returns the number of times that value occurs in the list;
- treeCount, which takes a binary tree and an integer value, and returns the number of times that value occurs in the tree;

You are mostly to annotate the provided code and use Dafny to prove the correctness of functions over ordered lists or trees. You are also to formulate and prove a couple of lemmas. This time, you do not have to turn induction off for their proof.

1. The given predicate isOrdered takes an integer list and returns true if and only if its elements are in non-decreasing order.

   State and prove lemma orderedListLemma, stating that every integer in an ordered list is smaller than or equal to the elements at higher positions in the list. Use the various list functions already provided as needed.

2. The provided function merge takes two ordered lists and merges them into an ordered list, returned as output.

   Add to merge the strongest contract that formalizes this behavior. Make sure the contract fully captures the specification above and is verified by Dafny.[1]

   Hint: In this problem and later ones, if your postconditions mention the returned value several times, it is cleaner to write a single (conjunctive) postcondition that first gives a name to that value, as in:

   ```
   ensures var l := merge(l1, l2);
           P1(l) && P2(l) && ...
   ```

   where P1, P2, and so on are the various postconditions.

3. Ghost function treeCount is useful in later specifications in the code. It turns out, however, that Dafny needs a little help with its properties.

   Add a suitable postcondition that relates the output of treeCount to the set of elements in the input tree.

4. The essential property of being a ordered tree can be expressed equivalently by saying the root of a non-empty ordered tree is smaller than or equal to the roots of its non-empty subtrees, if any. Ghost predicate isOrderedTree is meant to capture this property.

   Define isOrderedTree accordingly (see isOrderedList).

5. The provided function insert is similar to the insert function in insertion sort except that it inserts a value into an ordered tree. Specifically, it takes a tree ordered in the sense above and an integer x, and returns an ordered tree that contains an additional occurrence of x as well as all the occurrences of the other values in the input tree.

   Add a contract to insert that fully captures its functionality and make sure Dafny is able to verify it.[2] Use helper ghost functions as needed to keep the contract tidy.

6. The provided function toTree takes an integer list and returns an ordered tree containing all and only the integer occurrences in the list.

   Add a contract to toTree that fully captures its functionality and make sure Dafny is able to verify it.

7. The provided function toList takes an ordered tree and returns an ordered list containing all and only the integer occurrences in the tree.

   (a) Provide an implementation of this function.

   (b) Add a contract to toList that fully captures its functionality, making sure Dafny is able to verify it.

   Hint: merge is your friend.

[1] If Dafny is not able to verify the contract, chances are that it is either incorrect or not strong enough.

[2] In this case too, if Dafny fails to prove your contract, chances are that it is incorrect or not strong enough.

8. **Optional, Extra credit**
   Using the provided ghost predicate isBalanced, formalize and prove a lemma stating that applying function insert to a balanced ordered tree results in a balanced tree. Use auxiliary lemmas as needed.

Part B

File `hw3b.dfy` contains a number of methods that already have or must be provided with an iterative implementation based on a while loop. You are to specify the correctness of these methods using suitable variants and invariants for their loop.

1. Provide an iterative implementation of the given method IntLog which takes a positive integer n and returns its integer binary logarithm, that is, the largest natural number l such as $2^l \leq n$. In addition to the implementation, annotate the code with specs as follows.

   (a) Provide a contract for the method that fully captures its functionality.

   (b) A single while loop is enough for this method. Specify a variant function and invariants for the while loop as needed for Dafny to be able to verify the contract.

   Use the provided function `pow2` as needed.

   Hint: Verification will be easier if you implement IntLog without using division.

2. Consider the given Boolean method isPrime which is meant to take as input a natural number n greater than 1 and return true if and only if n is prime.

   (a) Define a ghost predicate prime that captures the property of being a prime natural number: i.e., not being divisible by any of its predecessors greater than 1.

   (b) Fully specify the functionality of isPrime with the aid of prime. Add preconditions, postconditions, variants and loop invariants as needed to verify the correctness of isPrime.

3. The provided method isPrefix takes an array a and an array b with the same type of elements, with b at least as long as a, and returns true if and only if every element of a, if any, is equal to the corresponding element of b.

   (a) Fully specify the functionality of isPrefix as described above with a contract.

   (b) Annotate the loop of isPrefix with invariants and a variant function as needed to allow Dafny to verify the contract.