Homework 4

Please submit your solution via email to the instructor with CC to ly603@nyu.edu.

The deadline for Homework 4 is March 4.

Problem 1 Search Tree Sets (25 Points)

In this exercise you will learn how to use abstraction functions to check conformance of concrete models against abstract models. Consider the following abstract model describing a data structure that implements set containers:

```
module homework/setContainer [Data]

sig Set { content: set Data }

fact Canonical { all s,s': Set | s.content = s'.content => s = s' }

pred insert [s,s': Set, d: Data] { s'.content = s.content + d }
pred delete [s,s': Set, d: Data] { s'.content = s.content - d }
pred contains [s: Set, d: Data] { d in s.content }
```

The relation `content` relates to each `Set` container s a set of data elements `Data` that are stored in the container. Each container supports three operations: insertion and deletion of data elements, and a membership test. The goal of this exercise is to check conformance of a more concrete implementation of set containers against this abstract model. The concrete implementation of set containers that we are considering is based on binary search trees. A stub of the corresponding module is as follows:

```
module homework/searchTreeSet [Data]

open util/ordering [Data]

sig Tree {
  root: lone Data,
  left: Data ->lone Data,
  right: Data ->lone Data
}

pred isTree [t: Tree] { ...}
pred isSortedTree [t: Tree] { ...}

pred insert [t,t': Tree, d: Data] { ...}
pred delete [t,t': Tree, d: Data] { ...}
pred contains [t, d: Data] { ...}
```

(a) Write a predicate `isTree` that takes a `Tree t` as argument and holds true if and only if the relations `t.left` and `t.right` form a (possibly empty) binary tree that takes atoms from `Data` as nodes and is rooted in `t.root`. Write a simulation predicate to generate some non-trivial trees. *Hint: make sure that each `Tree` defines a single tree and not a forest.* **(4 Points)**

(b) Write a predicate `isSortedTree` that takes a `Tree t` as argument and holds true if and only if t is a binary tree that is sorted according to the ordering imposed on `Data`. Recall that a tree is sorted if for every node n in the tree, all elements in the left subtree of n are smaller than n, and all elements in the right subtree are larger than n. Note that this definition does not allow duplicate elements in a tree, which is fine since we are interested in implementing sets. Write a simulation predicate to generate some interesting sorted binary trees. *Hint: the module `util/ordering` defines the necessary predicates to compare `Data` atoms. Note that the ordering that is imposed by `util/ordering` always corresponds to the lexicographic ordering on the names of atoms. For instance, if `Data0` and `Data1` are atoms in `Data`, then `Data0` will be smaller than `Data1`.* **(4 Points)**

(c) Write the predicates `insert`, `delete`, and `contains` for insertion, deletion, and membership test on search tree sets. The insert and delete operations should model the exact behavior you would expect from the concrete implementation of these operations on binary search trees. In particular, the resulting tree of each operation should only deviate from the input tree by constantly many changes in the tree structure. Simulate the operations for some non-trivial cases. *Hint: you can use (reflexive) transitive closure to model a traversal of the tree that searches for a particular element.*

(10 Points)

(d) Write two assertions `checkInsert` and `checkDelete` expressing that for each of the two operations `insert` and `delete`, if the pre-state `t` of the operation is a sorted binary tree, then the post-state `t'` is again a sorted binary tree. Check that both of your assertions hold in your model. **(3 Points)**

(e) Write a new module `homework/checkSearchTreeSet` that checks conformance of search tree sets against the abstract model of set containers. Your module should contain an abstraction function `alpha` relating `Trees` to `Sets` and three assertions expressing conformance of the operations `insert`, `delete`, and `contains` on `Trees` with their counterparts on `Sets`. Check that all your assertions hold. *Hint: Do not forget that in order to check conformance, you have to assume that the pre-state trees of your operations are sorted binary trees.* **(4 Points)**