Vrije Universiteit Amsterdam

Bachelor Thesis

Logical Verification of AVL Trees in Lean

Author: Sofia Konovalova (2635220)

1st supervisor: Jasmin Blanchette
daily supervisor: Jannis Limperg

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

April 17, 2021

Contents

1 AVL Trees

1.1 Binary Search Trees

While Lean does have a definition of a binary search tree, it is limited to the inductive definition `bin_tree`, in which a node only contains its left and right subtree. For a search tree, it is important that at the very least a tree node has a key and maybe some value.

```
universe u

inductive btree (α : Type u)
| empty {} : btree
| node (l : btree) (k : nat) (a : α) (r : btree) : btree
```

Figure 1: Inductive definition of a binary tree

Figure 1 shows the inductive definition of a binary tree. The definition has two constructors: an empty tree, and a node. A node contains a left and right subtrees, a key and some data of type α. In a binary search tree, lookup and insertion operations are recursive. Therefore, the definitions in Lean are recursive as well.

```
def lookup (x : nat) : btree α → option α
| btree.empty := none
| (btree.node l k a r) :=
  if x < k then lookup l
  else if x > k then lookup r
  else a
```

Figure 2: Lookup operation

Figure 2 shows the lookup operation for a binary search tree. The two argumentsthere is probably a different word for this are the key to search for and the tree itself. This definition will return an `option`. If the tree that the key is being search for is empty, than `none` is returned. If the key does exist, then the comparison to the node key is done. If the input key is smaller than that of the current node, then the definition is called recursively on the left subtree of the node; if the input key is larger, then on the right. If it is neither larger or smaller, then we know that we have found the right key, and we can return the value of the node.

```
def insert (x : nat) (a : α) : btree α → btree α
| btree.empty := btree.node btree.empty x a btree.empty
| (btree.node l k a' r) :=
  if x < k then btree.node (insert l) k a' r
  else if x > k then btree.node l k a' (insert r)
  else btree.node l x a r
```

Figure 3: Simple insert operation

An insertion operation in a binary search tree is similar to lookup. The three arguments for this definition are the new key and the new value of the node, and the tree to insert this information in. If the new key is smaller than the key of the current node, then the insertion operation is done recursively on the left subtree; if larger, then on the right. If you cannot compare any further, then this means that the keys are the same. If you are inserting into a tree with an existing key, then only the value gets updated. If the insertion is done into an empty subtree, then a new node is created with empty left and right subtrees, with the new key and new node.

1.2 Rotation

Appendices