Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Fabian Huch

SS 23
16. 5. 2023

Functional Data Structures

Exercise Sheet 4

Exercise 4.1 List Elements in Interval

Write a function to in-order list all elements of a BST in a given interval. I.e. *in_range t u v* shall list all elements x with $u \le x \le v$. Write a recursive function that does not descend into subtrees that definitely contain no elements in the given range.

fun *in_range* :: *"'a::linorder tree ⇒ 'a ⇒ 'a ⇒ 'a list"*

Show that you list the right set of elements

lemma *"bst t ⟹ set (in_range t u v) = {x∈set_tree t. u≤x ∧ x≤v}"*

Show that your list is actually in-order

lemma *"bst t ⟹ in_range t u v = filter (λx. u≤x ∧ x≤v) (inorder t)"*

Exercise 4.2 Fist Isar Steps

Using Isar, show the following theorem over natural numbers:

theorem
  assumes *"x ≥ (1 :: nat)"*
  shows *"(x + x^2)^2 ≤ 4 * x^4"*

Hint: When phrasing intermediate goals, check your types. Use *sledgehammer* to fill in simple proof steps.

Exercise 4.3 Enumeration of Trees

Write a function that generates the set of all trees up to a given height. Show that only trees up to the specified height are contained.

fun *enum* :: *"nat ⇒ unit tree set"*
lemma *enum_sound*: *"t ∈ enum n ⟹ height t ≤ n"*

(Time permitting) Show the other direction, i.e. that all trees of the specified height are contained.

lemma *enum_complete*: *"height t ≤ n ⟹ t ∈ enum n"*

lemma *enum_correct*: *"enum h = {t. height t ≤ h}"*
  by (*auto simp*: *enum_complete enum_sound*)

Homework 4 Max Annotated Trees

Submission until Monday, May 22, 23:59pm.

In this homework, we will develop an augmented binary search tree that stores the maximum element in the tree at thee root. With this auxiliary information, it is easier to implement queres such as membership:

datatype *'a mtree = Leaf | Node ('a mtree) 'a 'a ('a mtree)*

Define a function to return the set of elements in such a tree

fun *set_mtree* :: *"'a mtree ⇒ 'a set"*

Define a function that charcterises the invariant on the tree: the search tree property and the correct maximum node labels. Note: you do not have to use the function *Max*.

fun *mbst* :: *"'a::linorder mtree ⇒ bool"*

To confirm that the invariant characterises what it is supposed to, define a function which computes the maximum value in an ordered tree. This function has to recurse on the given tree. You can assume the tree is properly ordered. Note that in addition to the linorder, we also require that the elements of the tree have a *0* element.

fun *max_val* :: *"'a::{linorder,zero} mtree ⇒ 'a"*

Show that this function returns the label of the root of a given tree, if the tree is *mbst*.

lemma *mbst_max*: *"mbst (Node l m a r) ⟹ max_val (Node l m a r) = m"*

Define the insert function for this tree. Note: it has to correctly update the node with the correct maximum labels (but must not use the *max_val* function, as that is only for the specification).

fun *mins* :: *"'a::linorder ⇒ 'a mtree ⇒ 'a mtree"*

Now show that *mins* preserves the invariant. Hint: you will need a lemma showing that mins actually inserts the element in the set of elements in the tree.

lemma *mins_mbst*: *"mbst t ⟹ mbst (mins x t)"*

Define the membership query function and show it correct. Note: the function has to exploit the augmented maximum value!

fun *misin* :: *“'a::linorder ⇒ 'a mtree ⇒ bool”*

lemma *misin_set*: *“mbst t ⟹ misin x t ⟷ x∈set_mtree t”*

Specify a function that lists the elements within a given range in a given augmented tree and show that it lists the right elements. Again, the function must exploit the augmented maximum values.

fun *mtree_in_range* :: *“'a::linorder mtree ⇒ 'a ⇒ 'a ⇒ 'a list”*

Show that the function lists the right set of elements

lemma *mbst_range*: *“mbst t ⟹ set (mtree_in_range t u v) = {x∈set_mtree t. u≤x ∧ x≤v}”*