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

SS 2022
6. 5. 2022

Functional Data Structures

Exercise Sheet 2

Exercise 2.1 Fold function

In the last homework, you implemented a fold function yourself. Now have a look at Isabelle/HOL's standard function *fold*.

thm *fold.simps*

Define a function to sum up a list once recursive and once using fold, and show that both are equal.

fun *listsum* :: *"nat list ⇒ nat"*
definition *listsum'* :: *"nat list ⇒ nat"*
lemma *"listsum xs = listsum' xs"*

Exercise 2.2 Folding over Trees

Define a datatype for binary trees that store data only at leafs.

datatype *'a ltree* =

Define a function that returns the list of elements resulting from an in-order traversal of the tree.

fun *inorder* :: *"'a ltree ⇒ 'a list"*

In order to fold over the elements of a tree, we could use *fold f (inorder t) s*.

Define a function *fold_ltree* that is recursive on the structure of the tree, and that returns the same result as *fold f (inorder t) s*.

fun *fold_ltree* :: *"('a ⇒ 's ⇒ 's) ⇒ 'a ltree ⇒ 's ⇒ 's"*
lemma *"fold f (inorder t) s = fold_ltree f t s"*

Define a function *mirror* that reverses the order of the leafs, i.e. that satisfies the following specification:

lemma *"inorder (mirror t) = rev (inorder t)"*

Exercise 2.3 Shuffle Product

A shuffle of two lists, xs and ys, is a list that contains exactly the elements of xs and ys s.t. every two elements $x \in xs$ (resp. ys) and $x' \in xs$ (resp. ys) occur in the shuffle in the same order they do in xs (resp. ys).

Define a function *shuffles* that returns a list of all shuffles of two given lists

fun *shuffles* :: "*'a list ⇒ 'a list ⇒ 'a list list*"

Show that the length of any shuffle of two lists is the sum of the length of the original lists.

lemma "*zs ∈ set (shuffles xs ys) ⟹ length zs = length xs + length ys*"

Homework 2.1 Tail-recursive replace

Submission until Thursday, May 12, 23:59pm.

We want to define a tail-recursive function that replaces all elements a with b in a list.

First, specify a non tail-recursive version, and prove it correct (this should not be difficult):

fun *replace* :: "*'a ⇒ 'a ⇒ 'a list ⇒ 'a list*"
lemma *replace_len*: "*length (replace a b xs) = length xs*"

lemma *replace_set*: "*a ≠ b ⟹ a ∉ set (replace a b xs)*"

lemma *replace_set2*: "*b ∈ set xs ⟹ b ∈ set (replace a b xs)*"

For the tail-recursive version, the recursive call *must* be the outermost function. We use an additional accumulator parameter which stores the list reversed, and reverse in the end.

Complete the definition and show it correct.

fun *replace_tr* :: "*'a ⇒ 'a ⇒ 'a list ⇒ 'a list ⇒ 'a list*" **where**
  "*replace_tr _ _ acc [] = rev acc*"
lemma *replace_tr_len*: "*length (replace_tr a b [] xs) = length xs*"

lemma *replace_tr_set*: "*a ≠ b ⟹ a ∉ set (replace_tr a b [] xs)*"

lemma *replace_tr_set2*: "*b ∈ set xs ⟹ b ∈ set (replace_tr a b [] xs)*"

Hint: Start by generalizing the lemmas first. If *auto* loops on a goal involving *set*, try *clarsimp* instead.

Homework 2.2 Converting Trees

Submission until Thursday, May 12, 23:59pm.

Define a function to convert *tree*s from the library to the *ltree* type from the tutorial.

Note that while the constructors from the library trees have the same names (*Leaf* and *Node*), we can use the $\langle ... \rangle$ syntax to disambiguate.

fun *to_tree* :: *"'a tree ⇒ 'a ltree"*

You do not need to specify an equation for $\langle\rangle$, but for all other trees, the in-order traversal should be kept:

lemma *to_tree_inorder*: *"t ≠ ⟨⟩ ⟹ Tree.inorder t = inorder (to_tree t)"*

Prove that lemma!

Hint: use *nitpick* to check for problems in your definition first.