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

SS 23
2. 5. 2023

Functional Data Structures

Exercise Sheet 2

Exercise 2.1 Fold function

The fold function is a very generic function, that can be used to express multiple other interesting functions over lists.

Have a look at Isabelle/HOL's standard function *fold*.

thm *fold.simps*

Write a function to compute the sum of the elements of a list. Define two versions, one direct recursive definition, and one using fold. Show that both are equal.

fun *list_sum* :: *"nat list ⇒ nat"*
definition *list_sum'* :: *"nat list ⇒ nat"*

To use your definition in a proof, you need to use the theorem *list_sum'_def* explicitly.

lemma *"list_sum xs = list_sum' 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 Distinct lists

Submission until Monday, 8 May, 23:59pm.

Define a predicate *ldistinct* to characterize *distinct* lists, i.e., lists whose elements are pairwise disjoint. Use the contains function from the last sheet (contained in the Defs).

fun *ldistinct* :: *"'a list ⇒ bool"*

Show that a reversed list is distinct if and only if the original list is distinct.

lemma *ldistinct_rev*: *"ldistinct (rev xs) = ldistinct xs"*

Homework 2.2 More on fold

Submission until Monday, 8 May, 23:59pm.

Isabelle's fold function implements a left-fold. Additionally, Isabelle also provides a right-fold *foldr*.

Use both functions to specify the length of a list, and show them correct.

thm *fold.simps*
thm *foldr.simps*

definition *length_fold* :: *"'a list ⇒ nat"*
definition *length_foldr* :: *"'a list ⇒ nat"*

lemma *length_fold*: *"length_fold xs = length xs"*
lemma *length_foldr*: *"length_foldr xs = length xs"*

Homework 2.3 List Slices

Submission until Monday, 8 May, 23:59pm. Specify a function *slice xs s l*, that, for a list xs=[x_0,...,x_n] returns the slice starting at s with length l, i.e., [x_s,...,$x_{s+len-1}$].
If *s* or *len* is out of range, return a shorter (or the empty) list.

fun *slice* :: *"'a list ⇒ nat ⇒ nat ⇒ 'a list"*

Hint: Use pattern matching instead of *if*-expressions. For example, instead of writing *f x* = (*if x>0 then* ... *else* ...) you should define two equations *f 0* = ... and *f* (*Suc n*) = ...

Some test cases, which should all hold, i.e., yield *True*

value *"slice [0,1,2,3,4,5,6::int] 2 3 = [2,3,4]"*

(In range)

value *"slice [0,1,2,3,4,5,6::int] 2 10 = [2,3,4,5,6]"*

(Length out of range)

value *"slice [0,1,2,3,4,5,6::int] 10 10 = []"*

(Start index out of range)

Show that concatenation of two adjacent slices can be expressed as a single slice:

lemma *slice_append*: *"slice xs s l1 @ slice xs (s+l1) l2 = slice xs s (l1+l2)"*

Show that a slice of a distinct list is distinct.

lemma *ldistinct_slice*: *"ldistinct xs ⟹ ldistinct (slice xs s l)"*