Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Roßkopf

SS 24
26. 4. 2024

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 Association Lists

Submission until Thursday, May 02, 23:59pm.

An association list is a list of pairs. An entry (*k*,*v*) means that key *k* is associated to value *v*.

For an association list *xs*, the *collect k xs* operation returns a list of all values associated to key *k*, in the order stored in the list. Specify the function collect by a set of recursion equations:

fun *collect* :: *"'a ⇒ ('a × 'b) list ⇒ 'b list"*

Test cases

```
definition ctest :: "(int * int) list" where "ctest ≡ [
    (2,3),(2,5),(2,7),(2,9),
    (3,2),(3,4),(3,5),(3,7),(3,8),
    (4,3),(4,5),(4,7),(4,9),
    (5,2),(5,3),(5,4),(5,6),(5,7),(5,8),(5,9),
    (6,5),(6,7),
    (7,2),(7,3),(7,4),(7,5),(7,6),(7,8),(7,9),
    (8,3),(8,5),(8,7),(8,9),
    (9,2),(9,4),(9,5),(9,7),(9,8)
  ]"

value "collect 3 ctest = [2,4,5,7,8]"
value "collect 1 ctest = []"
```

An experienced functional programmer might also write this function as

$map\ snd\ (filter\ (\lambda kv.\ fst\ kv = x)\ ys)$

Show that this specifies the same function:

lemma *collect_alt*: "$collect\ x\ ys = map\ snd\ (filter\ (\lambda kv.\ fst\ kv = x)\ ys)$"

When the lists get bigger, efficiency might be a concern. To avoid stack overflows, you might want to specify a tail-recursive version of *collect*. The first parameter is the accumulator, that accumulates the elements to be returned, and is returned at the end.

Note: To avoid appending to the accumulator, we accumulate the elements in reverse order, and reverse the accumulator at the end.

Complete the second equation!

fun *collect_tr* :: "$'a\ list \Rightarrow 'b \Rightarrow ('b \times 'a)\ list \Rightarrow 'a\ list$" **where**
"$collect\_tr\ acc\ x\ [] = rev\ acc$"

Show correctness of your tail-recursive version. Hint: Generalization!

lemma *collect_tr_collect*: "$collect\_tr\ []\ x\ ys = collect\ x\ ys$"

Homework 2.2 Perfectly Balanced Trees

Submission until Thursday, May 02, 23:59pm.

Recall the tree datatype $'a\ ltree$ from the tutorial. Define functions to return the height (A leaf has height 0) and the number of leafs:

fun *lheight* :: "$'a\ ltree \Rightarrow nat$"
fun *num_leafs* :: "$'a\ ltree \Rightarrow nat$"

A tree is perfectly balanced iff, for each node, the left and right subtree have the same height. Specify a function to check that a tree is perfectly balanced.

fun *perfect* :: "$'a\ ltree \Rightarrow bool$"

Show that, for a perfectly balanced tree with height h and number of leafs l, we have $l = 2^h$:

lemma *perfect_num_leafs_height*: "$perfect\ t \Longrightarrow num\_leafs\ t = 2\hat{}lheight\ t$"

Homework 2.3 Shuffles

Submission until Thursday, May 02, 23:59pm.

In the tutorial you have defined the function *shuffles*. Show that the elements in a shuffle are exactly the elements of the input lists.

lemma *set_shuffles*: "$zs \in set\ (shuffles\ xs\ ys) \Longrightarrow set\ zs = set\ xs \cup set\ ys$"