Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

SS 2018
13. 4. 2018

Functional Data Structures

Exercise Sheet 1

Before beginning to solve the exercises, open a new theory file named `ex01.thy` and write the the following three lines at the top of this file.

theory *ex01*
imports *Main*
begin

Exercise 1.1 Calculating with natural numbers

Use the **value** command to turn Isabelle into a fancy calculator and evaluate the following natural number expressions:

"$2 + (2::nat)$" "$(2::nat) * (5 + 3)$" "$(3::nat) * 4 - 2 * (7 + 1)$"

Can you explain the last result?

Exercise 1.2 Natural number laws

Formulate and prove the well-known laws of commutativity and associativity for addition of natural numbers.

Exercise 1.3 Counting elements of a list

Define a function which counts the number of occurrences of a particular element in a list.

fun *count* :: "$'a\ list \Rightarrow 'a \Rightarrow nat$"

Test your definition of *count* on some examples and prove that the results are indeed correct.

Prove the following inequality (and additional lemmas, if necessary) about the relation between *count* and *length*, the function returning the length of a list.

theorem "$count\ xs\ x \leq length\ xs$"

Exercise 1.4 Adding elements to the end of a list

Recall the definition of lists from the lecture. Define a function *snoc* that appends an element at the right end of a list. Do not use the existing append operator @ for lists.

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

Convince yourself on some test cases that your definition of *snoc* behaves as expected, for example run:

value *"snoc [] c"*

Also prove that your test cases are indeed correct, for instance show:

lemma *"snoc [] c = [c]"*

Next define a function *reverse* that reverses the order of elements in a list. (Do not use the existing function *rev* from the library.) Hint: Define the reverse of $x \# xs$ using the *snoc* function.

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

Demonstrate that your definition is correct by running some test cases, and proving that those test cases are correct. For example:

value *"reverse [a, b, c]"*
lemma *"reverse [a, b, c] = [c, b, a]"*

Prove the following theorem. Hint: You need to find an additional lemma relating *reverse* and *snoc* to prove it.

theorem *"reverse (reverse xs) = xs"*

Homework 1 Sum of Values in List

Submission until Friday, Apr 20, 11:59am.

Submit your solution via https://vmnipkow3.in.tum.de. Submit a theory file that runs in Isabelle-2017 **without errors**.

General hints:

- If you cannot prove a lemma, that you need for a subsequent proof, assume this lemma by using sorry.
- Define the functions as simple as possible. In particular, do not try to make them tail recursive by introducing extra accumulator parameters — this will complicate the proofs!
- All proofs should be straightforward, and take only a few lines.

Specify a function to sum up all elements in a list of integers. The empty list has sum 0. Note that *int* is the type of (mathematical) integer numbers.

fun *listsum* :: *"int list ⇒ int"* **where**

Test cases:

value *"listsum [1,2,3] = 6"*
value *"listsum [] = 0"*
value *"listsum [1,−2,3] = 2"*

Prove that filtering out zeroes does not affect the sum.

Note: Isabelle might print $[x \leftarrow l\ .\ x \neq (0{::}'a)]$ instead of *filter* $(\lambda x.\ x{\neq}0)$ *l*, which is just a list comprehension syntax, that gets expanded during parsing, i.e., before Isabelle's core sees the term.

lemma *listsum_filter_z*: *"listsum (filter (λx. x≠0) l) = listsum l"*

Show that reversing a list does not affect the sum.

HINT: You'll need an auxiliary lemma relating *listsum* and append (*xs* @ *ys*).

Note that we use the *rev* function from the HOL list library here, which is the same as *reverse* specified in the tutorial, but comes with more lemmas etc..

lemma *listsum_rev*: *"listsum (rev xs) = listsum xs"*

Specify and prove the following: When filtering out negative elements from a list, its sum does not decrease.

lemma *"phrase this lemma yourself"*

Write a function to flatten a list of lists, i.e., concatenate all lists in the given list

fun *flatten* :: *"'a list list ⇒ 'a list"* **where**
  "flatten [] = []"
| *"flatten (l#ls) = l @ flatten ls"*

Test cases:

value *"flatten [[1,2,3],[2]] = [1,2,3,2::int]"*
value *"flatten [[1,2,3],[],[2]] = [1,2,3,2::int]"*

Show that the sum of the flattened list equals the sum of the sums of all element lists.
Hint: Auxiliary lemma!

lemma *"listsum (flatten xs) = listsum (map listsum xs)"*