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

SS 2022
29. 4. 2022

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 ⇒ 'a ⇒ 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 ≤ 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 Submission Instructions

Submissions are handled via https://do.proof.in.tum.de. Submit a theory file that runs in Isabelle **without errors**.

- Register an account in the system and send the tutor an e-mail. click here and fill in your details. Please don't put additional text in this mail.
- Select the competition "FDS 2022" and submit your solution following the instructions on the website (use the template file).
- The system will check that your solution can be loaded in Isabelle without any errors, and will then tell you how many of the checks you passed.
- You can upload multiple times; the last upload before the deadline is the one that will be graded.
- Only submissions with status "Passed" will be graded. If the submission seems to be rejected for reasons you cannot understand, and the `Check.thy` file runs locally without errors, please contact the tutor before the deadline.
- Partial credits may be given for:
  - finished intermediate lemmas
  - incomplete proofs, if they do not contain sorry and missing steps are extracted into succinct lemmas (which are assumed by using sorry).

  Mark all such incomplete proofs and sorried lemmas with `(*incomplete*)`, to help grading.
- We will be using a clone detection tool to compare solutions so please do not add any personal or identifying information to your homework solution theory files.

General hints:

- Define the functions as simply 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.

Homework 1.1 More Finger Exercise with Lists

Submission until Thursday, May 5, 23:59pm.

Define a function *fold_right* that iteratively applies a function to the elements of a list:

fun *fold_right* :: *"('a ⇒ 'b ⇒ 'b) ⇒ 'a list ⇒ 'b ⇒ 'b"*

More precisely, *fold_right f* $[x_1, x_2, \ldots, x_n]$ *a* should compute $f\ x_1\ (f\ x_2\ (\ldots\ (f\ x_n\ a)))$. The following evaluate to true, for instance:

value *"fold_right (+) [1,2,3] (4 :: nat) = 10"*
value *"fold_right (#) [a,b,c] [] = [a,b,c]"*

Prove that *fold_right* applied to the result of *map* can be contracted into a single

fold_right:

lemma *fold_map*: *"fold_right f (map g xs) a = fold_right (f o g) xs a"*

Here o is the regular composition operator on functions, i.e. $f\ o\ g = (\lambda x.\ f\ (g\ x))$.

Prove the following lemma on *fold_right* and *append*:

lemma *fold_append*: *"fold_right f (xs @ ys) a = fold_right f xs (fold_right f ys a)"*

For the remainder of the homework we will consider the special case where f is the addition operation on natural numbers. Prove that sums over natural numbers can be "pulled apart":

lemma *fold_append_plus*:
"fold_right (+) (xs @ ys) (0 :: nat) = fold_right (+) xs 0 + fold_right (+) ys 0"

The notation (+) is just a shorthand for $\lambda x\ y.\ x + y$.

Finally prove that calculating the sum from the right and from the left yields the same result:

theorem *fold_add_reverse*:
"fold_right (+) (reverse xs) (x :: nat) = fold_right (+) xs x"

Hint: You may need a lemma about *snoc* and *fold_right*.