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

SS 24
19. 4. 2024

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 Registration

Submissions are handled via https://do.proof.in.tum.de. 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.

Homework Submission

- Use the template from the competition "FDS 2024". **Do not** change the *existing* code of the template (except for the `sorry`s and `undefined`s), only add your solution (you can add other definitions, lemmas, etc. as well, but do not name any lemma `test`).
- Submit your solution following the instructions on the website.
- The system will check that your solution can be loaded in Isabelle2024-RC1 **without any errors**.
- You can upload multiple times; the last upload before the deadline is the one that will be graded.
- The submission system will give you feedback which checks were passed. Some checks are listed multiple times for weighting.

Homework Guidelines

- Only submissions with status "Passed" will be graded. If you have any problems uploading, or if the submission seems to be rejected for reasons you cannot understand, please contact the tutor *before the deadline*. **Make sure that the submission (and check file) runs through locally without errors.**
- Partial credits may be given for:
  - nearly correct definitions
  - 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).
- To claim partial credit (e.g., if you made progress in a proof but didn't finish it), **Mark it as `(*incomplete*)`**.
- We will be using a clone detection tool to compare solutions so please do not add any personal or identifying information.

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 Sum of odd numbers

Submission until Thursday, 25 April, 23:59pm.

Define a recursive function *oddsum* which computes the sum of the first n numbers. Your definition should have equations for *oddsum 0* and *oddsum* (*Suc n*).

fun *oddsum* :: *"nat ⇒ nat"*

For example, the following should evaluate to *True*:

value *"oddsum 3 = 5 + 3 + 1 + 0"*
value *"oddsum 7 = 49"*
value *"oddsum 1 = 1"*

You might want to test your implementation on more inputs using the **value** command. Then prove that the square of a natural number n can be computed as the sum of the first n odd numbers:

lemma *oddsum_is_square*: *"oddsum n = n * n"*

Finally prove the following property of *oddsum*:

lemma *oddsum_mult*: *"oddsum (n*m) = oddsum n * oddsum m"*

Homework 1.2 Spreading elements

Submission until Thursday, 25 April, 23:59pm.

Define a function *spread* that spreads an element among a list. This is, *spread a xs* adds the element *a* behind every element of *xs*.

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

The following evaluates to true, for instance:

value *"spread (0::nat) [1,2,3] = [1,0,2,0,3,0]"*

Prove that spreading an element amongst a list *xs* adds exactly *length xs* copies of the element to the list.

lemma *count_spread*: *"count (spread a xs) a = count xs a + length xs"*

Finally, prove the following lemma connecting *reverse*, *snoc* and *spread*:

lemma *spread_reverse_snoc*: *"snoc (reverse (spread a xs)) a = a # spread a (reverse xs)"*

Hint: You may need an auxiliary lemma about *spread* and *snoc*.