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

WS 2020/21
2.11.2020

Semantics of Programming Languages

Exercise Sheet 1

Before beginning to solve the exercises, open a new theory file named `Ex01.thy` and add the the following three lines at the beginning 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 1.1 More Finger Exercise with Lists

Submission until Sunday, November 8, 23:59.

Submission Instructions
Submissions are handled via https://competition.isabelle.systems/.

- Register an account in the system and send the tutor an e-mail with your username.
- Select the competition "Semantics 2020/21" and submit your solution following the instructions on the website.
- The system will check that your solution can be loaded in Isabelle2020 without any errors and reports how many of the main theorems you were able to prove.
- You can upload multiple times; the last upload before the deadline is the one that 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.
- Definitions from the tutorial (such as the *snoc* function) are made available in the submission file.

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 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.

Define a function *repeat* that repeats a value n times in a list.

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

The following evaluate to true, for instance:

value *"repeat 5 (0::nat) = [0, 0, 0, 0, 0]"*
value *"repeat 3 (1::nat) = [1, 1, 1]"*

Prove that the size of the resulting list is n:

theorem *rep_len*: *"length (repeat n a) = n"*

Finally, prove the following lemma connecting *reverse* and *repeat*:

theorem *rep_rev*: *"reverse (repeat n a) = repeat n a"*

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