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

WS 2014/15
14. 10. 2013

Semantics of Programming Languages

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 1.1 Two to the power of n

Submission until Tuesday, October 21, 10:00am.

In this homework, you will formalize the function 2^n and the well-known law $2^{n+m} = 2^n * 2^m$.

First, define a function that computes 2 to the power of its argument, i.e., *pow2 n* = 2^n. Use the **fun**-command for a recursive definition, i.e., give equations for *pow2 0* and *pow2 (Suc n)*.

fun *pow2* :: *"nat ⇒ nat"* **where**

You may wish to use the **value** command to check that your definition is correct. For example, the following command should evaluate to *True*.

value *"pow2 3 = 8"*

Next, prove the well-known law that $2^{n+m} = 2^n * 2^m$. The proof is a straightforward induction over n. First write an informal proof by hand. Make clear over what

variable you do induction, and what the assumptions and goals are in each case of the induction. Moreover, make clear what assumptions (and basic laws about addition and multiplication) you use.

Next, prove the statement formally in Isabelle/HOL.

lemma *"pow2 (n+m) = pow2 n * pow2 m"*

Homework 1.2 Doubling a List

Submission until Tuesday, October 21, 10:00am.

Define a function

fun *double* :: *"'a list ⇒ 'a list"* **where**

such that *double* $[x_1, x_2, \ldots] = [x_1, x_1, x_2, x_2, \ldots]$. Prove

lemma *rev_double*: *"rev(double xs) = double(rev xs)"*

Hint: This may require an auxiliary lemma!