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

**WS 2015/16**
**13. 10. 2015**

# 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** List-Sum

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

Mail a theory file named FirstnameLastname01.thy (replace with your name!) which runs in Isabelle-2015 **without errors** to *lammichatindottumdotde*.

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!
- Nitpick, Quickcheck, and Sledgehammer are your friends!
- All proofs should be straightforward, and take only a few lines.

Define a function that adds up the elements of a list:

**fun** *lsum* :: *"nat list ⇒ nat"*

Prove that reversing the list does not affect its sum.

**lemma** *"lsum (reverse xs) = lsum xs"*

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

Write a function that, for argument $n$ generates the list $[1..n]$:

**fun** *nlist* :: *"nat ⇒ nat list"*

Prove the well-known Gauss-formula:

**lemma** *"lsum (nlist n) = n * (n+1) div 2"*