Prof. Dr. K. Madlener
Dipl.-Inf. P. Michel
Dipl.-Inf. C. Feller

University of Kaiserslautern
Department of Computer Science
AG Grundlagen der Informatik

Exercise Sheet 3: Specification and Verification with Higher-Order Logic (Summer Term 2011)

Date: 03.05.2011

Exercise 1 Foundations

a) (Prepare!) What is the order of the following formulas?

- $\mathrm{Suc}(0) \neq 0$
- $\forall n.\ \mathrm{Suc}(n) \neq 0$
- $\forall n\ m.\ \mathrm{Suc}(n) = \mathrm{Suc}(m) \longrightarrow n = m$
- $\forall P.\ P(0) \wedge \Big(\forall n.\ P(n) \longrightarrow P(\mathrm{Suc}(n))\Big) \longrightarrow \forall n.\ P(n)$

b) (Prepare!) Determine which of these terms are syntactically correct. For the correct terms give possible types for all occurring variables and the complete term.

- $(\lambda x.\ x = a)\ b$
- $(\lambda x = x)$
- $(\lambda x.\ True) = (\lambda x.\ (f\ g\ x) = y)$
- $(x \longrightarrow x) = (b\ b)$

c) (Prepare!) Consider the following set of sets $U = \{\{1\}, \{1, 2\}\}$, which is not a universe. For each of the closure conditions violated by U, give an example set which should have been included in U.

d) (Prepare!) Consider the standard model $M = \langle (D_\alpha)_{\alpha \in \tau}, J \rangle$ for the set of types τ and constants defined in the lecture, where we consider the additional binary constant symbol $+ : ind \Rightarrow ind \Rightarrow ind$. The frame $(D_\alpha)_{\alpha \in \tau}$ is defined by $D_{bool} = \{T, F\}$, $D_{ind} = \mathbb{N}$ and $D_{\alpha \Rightarrow \beta} = D_\alpha \Rightarrow D_\beta$, i.e. the set of all functions from α to β. J interprets all constants as defined in the lecture and $+$ as the usual addition on natural numbers. Consider the following formula:

$$a = b \longrightarrow (\lambda x.x + a) = (\lambda x.b + x)$$

- Prove that the formula is satisfiable with regard to M, by giving an assignment under which the formula evaluates to T.
- Is the formula valid with regard to M?
- Is the formula valid in a standard sense?

Exercise 2 Sets as Functions in Isabelle/HOL

We have seen that the core of Isabelle/HOL does not contain types for sets. One way of introducing sets is to use functions to represent sets.

a) (Prepare!) Define the polymorphic type of sets as the type of unary predicate functions.

b) (Prepare!) Define the following constants for this type:

   1. The `empty` set.
   2. The `insert` function on sets.
   3. The `delete` function on sets.
   4. The `union` on sets.
   5. The `intersection` on sets.
   6. The set of all `even` integers.

Exercise 3 Datatypes and Properties in Isabelle/HOL

a) Define a datatype `'a tree` to represent binary trees. Leaves should be `Empty` and internal nodes should store a value of type `'a`.

b) Define the functions `preOrder`, `postOrder` and `inOrder` that traverse and convert a binary tree to a list in the respective order.

c) Define a function `mirror` that returns the mirror image of a binary tree.

d) Define the functions `root`, `leftmost` and `rightmost` on trees, which return the respective values for non-empty trees and are `undefined` otherwise.

e) Prove or disprove the following theorems:

   - `t` ≠ `Empty` ⟶ `last (inOrder t) = rightmost t`
   - `t` ≠ `Empty` ⟶ `hd (inOrder t) = leftmost t`
   - `t` ≠ `Empty` ⟶ `hd (preOrder t) = last (postOrder t)`
   - `t` ≠ `Empty` ⟶ `hd (preOrder t) = root t`
   - `t` ≠ `Empty` ⟶ `hd (inOrder t) = root t`
   - `t` ≠ `Empty` ⟶ `last (postOrder t) = root t`

f) Suppose that `xOrder` and `yOrder` are tree traversal functions chosen from `preOrder`, `postOrder`, and `inOrder`. Examine for which traversal functions the following formula holds:

```
xOrder (mirror xt) = rev (yOrder xt)
```

Exercise 4 A Simple Hilbert-Style Proof System (Optional)

This exercise is meant as additional training and to deepen your understanding of the calculus, proof systems, proof assistants and maybe most of all: ***functional programming****. We won't spend much time on it in the exercises though, so you can consider it* ***optional****.*

You can use whatever functional programming language you like. We used SML, but can easily give you support for Haskell, too. You can also do it in Isabelle/HOL itself, but to execute it you should use the code generation it offers.

We want to develop a system to represent and check proofs in the Hilbert-Calculus, like defined on Sheet 1. We use the following datatypes to represent formulas, proof states and proof commands:

```
datatype f =        (* formulas *)
    Var of string   (* variables *)
  | Neg of f        (* negation *)
  | Imp of f * f    (* implication *)

infix 3 -->         (* nicer syntax for implication *)
fun a --> b = Imp (a, b)

type proofState = f list  (* Hilbert Calculus proof state *)

datatype proofCommand = (* command to modify the proof state *)
    A1 of f * f         (* insert an instantiation of an axiom schema *)
  | A2 of f * f * f
  | A3 of f * f
  | MP of int * int     (* apply modus ponens to two elements of the proof state *)

type proof = proofCommand list
```

a) Define a function `applyCommand`, which applies a proof command to a proof state and returns the new proof state. If the proof command is not applicable, the function should raise an exception.

b) Define a function `apply`, which takes a list of assumptions (initial proof state) as well as a proof and returns the final proof state.