Exercise Sheet 3: Specification and Verification with Higher-Order Logic (Summer Term 2010)

Date: 26.04.2010

Exercise 1 Foundations

a) (Prepare!) What is the order of the following formulas?
   • Suc(0) ≠ 0
   • ∀n. Suc(n) ≠ 0
   • ∀nm. Suc(n) = Suc(m) −→ n = m
   • ∀P. P(0) ∧ (∀n. P(n) −→ P(Suc(n))) −→ ∀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.
   • (λx. x = a) b
   • (λx = x)
   • (λx. True) = (λx. (f g x) = y)
   • (x −→ 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 = ((D_α)_{α ∈ τ}, J) for the set of types τ and constants defined in the lecture, where we consider the additional binary constant symbol + : ind ⇒ ind ⇒ ind. The frame (D_α)_{α ∈ τ} is defined by D Bool = {T, F}, D ind = N and D α⇒β = D α ⇒ D β, 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 −→ (λx.x + a) = (λx.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?
Exercises 2 and 4 will only be discussed in the exercise meeting, if there are questions or problems with their solutions. Like any other exercises they are relevant for the exam and you should be able to solve them. In the meeting we will discuss exercise 1 and concentrate on solving exercise 3 together. You can of course work on the other exercises in the meeting as well and ask for assistance when necessary.

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.

Define the polymorphic type of sets as the type of unary predicate functions. Now define the following constants:

a) The empty set.
b) The insert function on sets.
c) The delete function on sets.
d) The union on sets.
e) The intersection on sets.
f) 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:
   - \( t \neq \text{Empty} \implies \text{last (inOrder } t) = \text{rightmost } t \)
   - \( t \neq \text{Empty} \implies \text{hd (inOrder } t) = \text{leftmost } t \)
   - \( t \neq \text{Empty} \implies \text{hd (preOrder } t) = \text{last (postOrder } t) \)
   - \( t \neq \text{Empty} \implies \text{hd (preOrder } t) = \text{root } t \)
   - \( t \neq \text{Empty} \implies \text{hd (inOrder } t) = \text{root } t \)
   - \( t \neq \text{Empty} \implies \text{last (postOrder } t) = \text{root } t \)
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:
   \[ \text{xOrder (mirror } xt) = \text{rev (yOrder } xt) \]
Exercise 4 A Simple Hilbert-Style Proof System

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:

```haskell
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.