Prof. Dr. A. Poetzsch-Heffter
Dipl.-Inf. P. Michel
Dipl.-Inf. C. Feller

University of Kaiserslautern
Department of Computer Science
Software Technology Group

# Exercise Sheet 3: Specification and Verification with Higher-Order Logic (Summer Term 2012)

Date: 25.04.2012

## Exercise 1 Datatypes and Properties in Isabelle/HOL

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

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

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

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

e) Prove or disprove the following theorems:

- `t` $\neq$ `Empty` $\longrightarrow$ `last (inOrder t) = rightmost t`
- `t` $\neq$ `Empty` $\longrightarrow$ `hd (inOrder t) = leftmost t`
- `t` $\neq$ `Empty` $\longrightarrow$ `hd (preOrder t) = last (postOrder t)`
- `t` $\neq$ `Empty` $\longrightarrow$ `hd (preOrder t) = root t`
- `t` $\neq$ `Empty` $\longrightarrow$ `hd (inOrder t) = root t`
- `t` $\neq$ `Empty` $\longrightarrow$ `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 2 Sets as Functions in Isabelle/HOL

Consider the following type synonym: `type_synonym 'a myset = "'a ⇒ bool"`

The idea behind this definition is that sets can be represented by their characteristic function, i.e., the function which decides for each element if it is in the set or not.

(Prepare!) Define the following constants for our new type:

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

# Exercise 3 Using and Extending the Simple Theorem Prover

You can find the `SimpleTheoremProver2.thy` used in the lecture on our website. We want to use this simple prover to prove the theorem from Exercise 1a on Sheet 1.

a) (Prepare!) Download the file and run it through Isabelle/HOL. Think about the existing proof in the file and how it is done.

b) (Prepare!) Extend the data type for formulas to support the logical operators `And` and `Or`. Remember to also adjust the functions which are using the data type!

c) (Prepare!) The theory so far has only four rules. Add further rule definitions from the natural deduction calculus, such that you can do the proof of the sequent from Exercise 1a.

d) (Prepare!) Proof the sequent: $\vdash (a \vee (b \wedge c)) \rightarrow ((a \vee b) \wedge (a \vee c))$

### Tips and Tricks to make your life easier

- You can define abbreviations in Isabelle/HOL, which will be used for both input and output. For example, you might want to define abbreviations for normal and schematic variables like this:

```
abbreviation a where "a ≡ Var ''a''"
abbreviation b where "b ≡ Var ''b''"
abbreviation c where "c ≡ Var ''c''"

abbreviation A where "A ≡ SVar ''A''"
abbreviation B where "B ≡ SVar ''B''"
abbreviation C where "C ≡ SVar ''C''"
```

  You can then use the abbreviations in the theory but internally they will still be applications of `Var` and `SVar`, respectively, without the need to unfold a definition!

  Isabelle/HOL will also use them for its output, so that's nicer as well!

- You probably will use "copy & paste" a lot in this exercise. You can replace a string with something else in a marked block (or starting from the cursor if nothing is marked), using META % (press ESC once, then press %).

  Emacs will prompt you for the string to search, then for the string to replace it by. He will then highlight the first appearence and ask what to do. You can get a help window within Emacs itself at this point, but we just print it here for your convenience:

```
Type Space or 'y' to replace one match, Delete or 'n' to skip to next,
RET or 'q' to exit, Period to replace one match and exit,
Comma to replace but not move point immediately,
C-r to enter recursive edit (C-M-c to get out again),
C-w to delete match and recursive edit,
C-l to clear the screen, redisplay, and offer same replacement again,
! to replace all remaining matches with no more questions,
^ to move point back to previous match,
E to edit the replacement string
```

  The most commonly used keys are 'y' and 'n' to replace or not replace an occurence, 'q' to quit and '!" to replace *all* occurrences in the file.