Prof. Dr. A. Poetzsch-Heffter
Dipl.-Inform. J. O. Blech
Dipl.-Inform. M. J. Gawkowski

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Handout 2: Specification and Verification Using Higher-Order Logic (summer term 2008)

22. April 2007

Exercise 1 Binary Trees

Define a datatype `'a bintree` for binary trees. Both leaf and internal nodes shall store information.

Based on this datatype the following exercises shall be solved:

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

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

c) Suppose that `xOrder` and `yOrder` are tree traversal functions chosen from `preOrder`, `postOrder`, and `inOrder`. Formulate and prove some non-trivial valid properties of the form:

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

Exercise 2 Properties of Binary Trees

This exercise is based on the definition of binary trees from the previous exercise.

Define the functions `root`, `leftmost`, and `rightmost`, that return the root, leftmost, and rightmost element of a binary tree respectively.

Prove or disprove (by counterexample) the theorems that follow. You may have to prove some lemmata first.

a)
```
theorem "last (inOrder xt) = rightmost xt"
 (*<*) oops (*>*)
```

b)
```
theorem "hd (inOrder xt) = leftmost xt"
 (*<*) oops (*>*)
```

c)
```
theorem "hd (preOrder xt) = last (postOrder xt)"
(*<*) oops (*>*)
```

d)
```
theorem "hd (preOrder xt) = root xt"
(*<*) oops (*>*)
```

e)
```
theorem "hd (inOrder xt) = root xt"
(*<*) oops (*>*)
```

f)
```
theorem "last (postOrder xt) = root xt"
(*<*) oops (*>*)
```

Exercise 3 Folding Lists

Make yourself familiar with the definitions of `foldl`, `foldr`, `map`, and the following `sum` function.

```
consts
  sum :: "nat list => nat"
primrec
  "sum [] = 0"
  "sum (x # xs) = x + sum xs"
```

Show the following lemmata. You may have to instantiate `h` and `b`.

```
lemma sum_foldr: "sum xs = foldr (op +) xs 0"
(*<*) oops (*>*)

lemma length_foldr: "length xs = foldr (% x res. 1 + res) xs 0"
(*<*) oops (*>*)

lemma "sum (map (% x. x + 3) xs) = foldr h xs b"
(*<*) oops (*>*)

lemma "foldr g (map f xs) a = foldr h xs b"
(*<*) oops (*>*)
```