Workshop Testing and Formal Methods (Week 6)
Note: this is the theory part of the Exam Software Specification and Testing held on October 23, 2014

```
module Workshop6

where
import Data.List
```

There are six questions.

Question 1 A property p on a domain D is a function $D \to \{\top, \bot\}$, where $\top$ and $\bot$ are the two truth values. A property p on D is stronger than a property q on D if it holds for all $x \in D$ that $p(x)$ implies $q(x)$. A property p on D is weaker than a property q on D if q is stronger than p. (You have seen these notions during the workshop sessions.)

Which of the following pairs of properties on the integers is stronger?

1. $\lambda x \mapsto x = 0$ and $\lambda x \mapsto x \leq 0$.
2. $\lambda x \mapsto x^2 \neq 4$ and $\lambda x \mapsto x \neq 2$.
3. $\lambda x \mapsto x^2 \geq 49$ and $\lambda x \mapsto x \geq 7 \vee x \leq -7$.
4. $\lambda x \mapsto x^2 \geq 0$ and $\lambda x \mapsto \bot$.
5. $\lambda x \mapsto x < 3$ and $\lambda x \mapsto x \neq 0$.

Question 2 Suppose function $f : a \to b$ has precondition $p : a \to \text{Bool}$ and postcondition $q : b \to \text{Bool}$. Suppose function $g : b \to c$ has precondition q and postcondition $r : c \to \text{Bool}$. Suppose the precondition of $g \cdot f$ is p. Derive from this a strongest postcondition for the function $g \cdot f$.

Question 3 Consider the following definition of expressions (you have seen this in the course):

```
data Expr = I Int | V String
          | Add Expr Expr
          | Subtr Expr Expr
          | Mult Expr Expr
```

To evaluate expressions, one needs values for the variables that occur in them. Write a function that collects these variables:

```
vars :: Expr -> [String]
vars ...
```

Question 4 Give all binary relations on the set $\{0, 1\}$ that are *not* transitive. First express these relations as sets of pairs, next draw pictures of them.

Question 5 Consider the following substitution procedure s on binary strings: replace each occurrence of 0 by 01 and each occurrence of 1 by 0. Start with 0, and repeat the procedure. This gives the following list of binary strings.

$$\begin{array}{lcl cl}
s^0(0) & & & = & 0 \\
s^1(0) & = & s(0) & = & 01 \\
s^2(0) & = & ss(0) & = & 010 \\
s^3(0) & = & sss(0) & = & 01001 \\
s^4(0) & = & ssss(0) & = & 01001010 \\
s^5(0) & = & sssss(0) & = & 0100101001001 \\
 & \vdots & & &
\end{array}$$

The Italian mathematician Leonardo Fibonacci, who lived around 1200, proposed this substitution scheme as a model of the breeding behaviour of rabbits. Let 0 represent a pair of mature rabbits and 1 a pair of young rabbits. Every month, each pair of mature rabbits produces a pair of young rabbits. This explains $0 \mapsto 01$. The young rabbits take one month to mature. This explains $1 \mapsto 0$.

Show by induction that the number of ones in $s^n(0)$ equals F_n and the number of zeros in $s^n(0)$ equals F_{n+1}. Here, F is the function for the Fibonacci numbers. Recall that the Fibonacci function F is defined by $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$.

See next page for an illustration.

The following implementation of the substitution operation is just for illustration:

```
s :: [Int] -> [Int]
s [] = []
s (0:xs) = 0:1:s xs
s (1:xs) = 0:  s xs

proc = iterate s [0]
```

This gives:

```
*Workshop6> take 7 proc
[[0],[0,1],[0,1,0],[0,1,0,0,1],[0,1,0,0,1,0,1,0],[0,1,0,0,1,0,1,0,0,1,0,0,1],
 [0,1,0,0,1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0]]
```

Question 6 Consider the following definition of binary trees with information at the internal nodes.

```
data Btree a = Leaf | B a (Btree a) (Btree a)
```

Consider the following insertion function.

```
insertT :: Ord a => a -> Btree a -> Btree a
insertT x Leaf = B x Leaf Leaf
insertT x (B y left right)
  | x < y     = B y (insertT x left) right
  | otherwise = B y left (insertT x right)
```

Call a binary tree *ordered* if either it is a single leaf, or its two subtrees are ordered, and moreover all items in its left subtree are less than or equal to the item at its root node, and the item at its root node is less than or equal to all items in its right subtree.

Show with induction on tree structure that if `tree` is ordered, then `insertT x tree` is also ordered.