LiU-FP2016, Linköping, 23–27 May 2016

Problem Set 5: Monads

Henrik Nilsson

---

1. Verify that `Maybe a` indeed is a monad by verifying the monad laws for `mbReturn` and `mbSeq` from Lecture 7.

2. (a) Write a Haskell program that asks a user for his/her name and then greets the user by name ten times. Use `do`-notation.

   (b) Reimplement the tree numbering program from Lecture 7 using the `ST` monad to keep a counter in an imperative variable (an `STRef`). The function `numberTree` should still have the type `Tree a -> Tree Int`: the use of imperative effects for implementing the function should be hidden. You can find the functionality you need in the modules `Control.Monad.St` and `Data.STRef`.

3. Below are the type signatures for a number of monad utility functions from the Haskell prelude and the module `Monad`. Define these utilities in terms of the basic monad operations. (If it is not reasonably clear from the type signatures what the intended meaning of each function is, ask!)

```
sequence  :: Monad m => [m a] -> m [a]
sequence_ :: Monad m => [m a] -> m ()
mapM      :: Monad m => (a -> m b) -> [a] -> m [b]
mapM_     :: Monad m => (a -> m b) -> [a] -> m ()
when      :: Monad m => Bool -> m () -> m ()
foldM     :: Monad m => (a -> b -> m a) -> a -> [b] -> m a
liftM     :: Monad m => (a -> b) -> (m a -> m b)
```