ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 7: Monads and Repetition

Submission deadline: —

Assignment 1: Monads

Recall the `Monad` typeclass.

```
class Monad m where
   return :: a -> m a
   (>>=)  :: m a -> (a -> m b) -> m b
```

A `Monad` class instance is *valid* iff it satisfies the following three *monad laws*.

1. Left identity: `return x >>= f  =  f x`
2. Right identity: `m >>= return  =  m`
3. Associativity: `(m >>= f) >>= g  =  m >>= (\x -> f x >>= g)`

Consider the following type of binary trees.

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```

(a) Give a `Monad` class instance for `Tree`.

(b) Prove that your `Monad` class instance is valid; i.e., satisfies the monad laws.

Note: The *typeclassopedia*[1] provides a very good ressource for learning more about standard typeclasses like the `Monad` typeclass. Moreover, as it happens, the typeclassopedia will be discussed by the *Zurich Haskell user group*[2] here at ETH in CNB 100.5 on Thu, 12th of April 2012. If you are interested in learning more about Haskell and functional programming in general, then you are more than welcome :-)

[1] `http://www.haskell.org/haskellwiki/Typeclassopedia#Monad`
[2] `http://www.meetup.com/HaskellerZ/`

Assignment 2: Evaluation

Simplify the following terms stepwise using (i) lazy and (ii) eager evaluation reduction.

(a) $(\lambda x.\, \lambda y.\, x\, y)\, (\lambda z.\, y\, z)$

(b) $(\lambda f.\, \lambda x.\, x\, f)\, (\lambda r.\, r)\, ((\lambda y.\, \lambda z.\, y\, z)\, (\lambda a.\, a))$

Note: You can download a mini Haskell interpreter from the course web page that outputs stepwise reductions of terms (lazy and eager).

Assignment 3: Typing

(a) Recall the following functions from the Haskell libraries.

```
(:)    :: a -> [a] -> [a]
map    :: (a -> b) -> [a] -> [b]
inits  :: [a] -> [[a]]
1      :: Num a => a
(<)    :: Ord a => a -> a -> Bool
```

State the most general type of each of the following expressions.

1. `(\x y z -> (y x, x z))`
2. `(\x y -> x (\z -> y))`
3. `map (1 :)`
4. `(\x ys -> inits (map (x <) ys))`

(b) Recall the proof rules for the mini-Haskell type system:

$$\frac{}{\ldots, x : a, \ldots \vdash x :: a} \qquad \frac{}{A \vdash n :: \mathit{Int}} \qquad \frac{}{A \vdash \textbf{True} :: \mathit{Bool}} \qquad \frac{}{A \vdash \textbf{False} :: \mathit{Bool}}$$

$$\frac{A, x : \sigma \vdash t :: \tau}{A \vdash (\lambda x.\, t) :: \sigma \to \tau}\ x \notin A \qquad \frac{A \vdash t_1 :: \sigma \to \tau \quad A \vdash t_2 :: \sigma}{A \vdash (t_1\, t_2) :: \tau}$$

$$\frac{A \vdash t_1 :: \mathit{Int} \quad A \vdash t_2 :: \mathit{Int}}{A \vdash (t_1 + t_2) :: \mathit{Int}} \qquad \frac{A \vdash t_1 :: \tau_1 \quad A \vdash t_2 :: \tau_2}{A \vdash (t_1, t_2) :: (\tau_1, \tau_2)}$$

Formally prove the following typing judgement using the typing rules above:

$$\vdash (\lambda y.\, (\textbf{True}, \lambda x.\, 1 + x\, y)) :: a \to (\mathit{Bool}, (a \to \mathit{Int}) \to \mathit{Int})$$

Assignment 4: Proof by Induction

Consider the following functions.

```
addShifted :: [Int] -> Int -> [Int]
addShifted []     i = [i]                          -- addShifted.1
addShifted (x:xs) i = (x + i):addShifted xs x      -- addShifted.2

sum :: [Int] -> Int
sum []     = 0                                     -- sum.1
sum (x:xs) = x + sum xs                            -- sum.2
```

Prove that $\forall xs :: \texttt{[Int]}.\ \forall i :: \texttt{Int}.\ \texttt{sum}\ (\texttt{addShifted}\ xs\ i) = i + 2 * \texttt{sum}\ xs$.

Assignment 5: List Functions

(a) Define a Haskell function `match :: String -> Bool` that checks whether the parentheses match. You may assume that `()` and `{}` are the only parentheses. Note that a string can contain arbitrary letters of type `Char`.

Example: `match "(a{b} cd)()ef" = True` and `match "(xy{z}}" = False`

Hint: Use a stack and represent this stack as a list.

(b) Write a function `risers :: Ord a => [a] -> [[a]]` that splits a list `xs :: [a]` into the list of longest non-empty monotonically rising subsequences of `xs`.

Example: `risers [1,3,3,4,1,0,2,6] = [[1,3,3,4],[1],[0,2,6]]`

Assignment 6: Symbolic Differentiation

In this assignment, you have to implement a Haskell function that symbolically computes the first derivative of a functional expression. Functional expressions $f(x)$ are interpreted in the common way and may contain the variable symbol x, the constant e, the real numbers (represented as values of type `Double`), addition, multiplication, division, potentiation $(f(x)^{g(x)})$, exponentiation $(e^{f(x)})$, and the natural logarithm.

(a) Find a suitable representation in Haskell for functional expressions $f : \mathbb{R} \to \mathbb{R}$.

(b) Implement a function that computes the first derivative of a functional expression.

Recall the derivation rules ($f(x)$, $g(x)$, and $h(x)$ are functional expressions):

$$
\begin{aligned}
x' &= 1.0 & c' &= 0.0 \text{ for } c \in \mathbb{R} \\
\big(g(x) + h(x)\big)' &= g'(x) + h'(x) & \big(g(x) \cdot h(x)\big)' &= g'(x) \cdot h(x) + g(x) \cdot h'(x) \\
\big(g(h(x))\big)' &= g'(h(x)) \cdot h'(x) & \big(e^{f(x)}\big)' &= f'(x) \cdot e^{f(x)} \\
\big(\ln(f(x))\big)' &= \tfrac{f'(x)}{f(x)}
\end{aligned}
$$

and remember the equality $g(x)^{h(x)} = e^{h(x) \cdot \ln(g(x))}$.

Note: Simplification rules need not be implemented.