

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp

Exercises *Functional Programming* – Sheet 5

Solutions will be collected until Wednesday, June 01, 2005 in the exercise course.

Exercises can be solved both in English and in German.

Exercise 1 (2 + 1 + 4 points)

We define a monad `MyMaybe` as follows:

```
data MyMaybe a = Value a | Error

instance Monad MyMaybe where
  return      = Value
  Error  >>= q = Error
  Value x >>= q = q x

instance Show a => Show (MyMaybe a) where
  show Error = "An error occurred!"
  show (Value x) = show x
```

- (a) Show that the three monad-laws hold for the `MyMaybe` monad.
- (b) In the following Haskell code the `MyMaybe`-monad is used in the evaluation of terms (i.e., of objects from the data structure `Term`). Here, a return value of `Error` means that a division by zero error occurred.

```
data Term = Con Float | Div Term Term

eval :: Term -> MyMaybe Float
eval (Con x) = Value x
eval (Div t u) = do x <- eval t
                   y <- eval u
                   if y /= 0 then return (x/y) else Error
```

Extend the data structure `Term` by a data constructor `Sqrt :: Term -> Term` for representing square roots. Extend the function `eval` to handle terms containing `Sqrt`, too. Here, you can use the pre-defined function `sqrt :: Float -> Float`. Trying to compute the square root of a negative number should also return `Error` to indicate an error. For example, the evaluation of the Haskell term `show (eval (Sqrt (Con 25.0)))` should yield the string `"5.0"` while `show (eval (Sqrt (Con (-25.0))))` should return `"An error occurred!"`.

- (c) Replace the data structure `MyMaybe` in the algorithm `eval` by a new data structure that can distinguish between errors resulting from division by zero and errors resulting from computing the square root of a negative number. For example, the evaluation of `show (eval (sqrt (Con (-25.0))))` should yield the string `"Cannot take square root of a negative number!"` while `show (eval (Div (Con 1.0) (Con 0.0)))` should return the string `"Cannot divide by zero!"`.

Make your new data structure an instance of the classes `Monad` and `Show` and change the implementation of `eval` as little as possible.

Exercise 2 (3 points)

Prove that if all \sqsubseteq_{D_i} are reflexive orders, then $\sqsubseteq_{D_1 \times \dots \times D_n}$ is a reflexive order.

Exercise 3 (1+2+2 points)

Let \sqsubseteq_{D_1} and \sqsubseteq_{D_2} be orders on D_1 and D_2 respectively. A function $f : D_1 \rightarrow D_2$ is said to be monotonic if and only if $f(d) \sqsubseteq_{D_2} f(d')$ for all $d \sqsubseteq_{D_1} d'$.

Find all monotonic extensions of the following total functions (given their usual interpretation):

- (a) $\neg : \mathbb{B} \rightarrow \mathbb{B}$
- (b) $\wedge, \vee : \mathbb{B}^2 \rightarrow \mathbb{B}$
- (c) $+, * : \mathbb{N}^2 \rightarrow \mathbb{N}$

Exercise 4 (2+1 points)

- (a) Present the order $(\mathbb{B}_\perp \rightarrow \mathbb{B}_\perp, \sqsubseteq)$ graphically. How many elements does it have? How many elements are maximal?
- (b) How many elements does the order $(\mathbb{B}_\perp \rightarrow \mathbb{B}_\perp^2, \sqsubseteq)$ have?

Exercise 5 (1+3 points)

Consider the function `minus` defined as follows:

```
minus :: (Int, Int) -> Int
minus (0, y) = 0
minus (x, 0) = x
minus (x, y) = minus (x-1, y-1)
```

As an interpretation of `minus` we have the function $f : \mathbb{Z}_\perp \times \mathbb{Z}_\perp \rightarrow \mathbb{Z}_\perp$ with

$$f(x, y) = \begin{cases} 0, & \text{if } x = 0 \vee (x > 0 \wedge (x \leq y \vee y < 0)) \\ x - y, & \text{if } y \geq 0 \wedge (x < 0 \vee x > y) \\ \perp_{\mathbb{Z}}, & \text{otherwise} \end{cases}$$

Prove or disprove the following statements.

- (a) The function f is strict.
- (b) The function f is monotonic.