3. Übung zur Vorlesung "Fortgeschrittene funktionale Programmierung"

Listen und Bäume

Boris Dudelsack

25. Oktober 2016

Aufgabe 1: Refactoring der Graphik

```haskell
type Graphic = [Object]

single :: Object → Graphic
single o = [o]

(<>) :: Graphic → Graphic → Graphic
(<>) [] g = g
(<>) g [] = g
(<>) g (o:os) = (o : g) <> os

objToSVG :: Object → String
objToSVG(Rect (Point x1 y1) (Point x2 y2) s) = "<rect x=\"" ++ show x1 ++ "\" y=\"" ++ show y1
     ++ "\" width=\""
     ++ show x2 ++ "\" height=\"" ++ show y2 ++ "\" " ++ styleToAttr s ++ "/>"
objToSVG(Circle (Point x y) r s) = "<circle cx=\"" ++ show x ++ "\" cy=\"" ++ show y ++ "\" r
    =\"" ++ show r ++ "\" " ++ styleToAttr s ++ "/>"

toSVG :: Graphic → String
toSVG g = "<svg version=\"1.1\" xmlns=\"http://www.w3.org/2000/svg\">\n" ++ toSVG_ g ++ "\n</
    svg>";

toSVG_ :: Graphic → String
toSVG_ [] = ""
toSVG_ (o:[]) = objToSVG o
toSVG_ (o:g) = objToSVG o ++ "\n" ++ toSVG_ g

rectangle :: Double → Double → Graphic
rectangle d1 d2 = single (Rect (Point 0.0 0.0) (Point d1 d2) defaultStyle)

circle :: Double → Graphic
circle r = single (Circle (Point (0.0 + r) (0.0 + r)) r defaultStyle)

colored' :: Color → Object → Object
colored' c (Rect p1 p2 s) = Rect p1 p2 (Style c)
colored' c (Circle p r s) = Circle p r (Style c)

colored :: Color → Graphic → Graphic
colored c g = map (colored' c) g
```

Aufgabe 2: XML-Datenstruktur

```haskell
data XML = XText String
    | XNode String [Attr] [XML]
    deriving Show

data Attr = String := String
    deriving Show

-- wandelt ein Attribute in die entsprechende String- Darstellung um
attrToString :: Attr → String
attrToString (k := v) = "\"" ++ k ++ "\"=\"" ++ v ++ "\""

-- wandelt eine Liste von Attributen in die entsprechende String- Darstellung um
attrsToString :: [Attr] → String
attrsToString a = unwords (map attrToString a)

-- wandelt ein XML-Dokument in die entsprechende String- Darstellung um
xmlToString :: XML → String
xmlToString (XText s) = s
xmlToString (XNode tag attrs xml) = "<" ++ unwords [tag, attrsToString attrs] ++ ">" ++
    concatMap xmlToString xml ++  "</" ++ tag ++ ">"

-- wandelt den Style in die String- Darstellung um
styleToAttr' :: Style → String
styleToAttr' (Style c) = "fill: " ++ colorToStr c ++ "; stroke: " ++ colorToStr c ++ ";"

-- wandelt ein Grafikobjekt in die SVG- Darstellung um
objToSVG' :: Object → XML
objToSVG' (Rect (Point x1 y1) (Point x2 y2) s) = XNode "rect" ["x" := show x1, "y" := show y1,
     "width" := show x2, "height" := show y2, "style" := styleToAttr' s] []
objToSVG' (Circle (Point x y) r s) = XNode "circle" ["cx" := show x, "cy" := show y, "r" :=
    show r, "style" := styleToAttr' s] []

-- wandelt eine Graphik in die SVG- Darstellung um
toSVG' :: Graphic → XML
toSVG' g = XNode "svg" ["version" := "1.1", "xmlns" := "http://www.w3.org/2000/svg"] (map
    objToSVG' g)
```

Aufgabe 3: Binärbäume

```haskell
data BinTree a = Empty | Node a (BinTree a) (BinTree a) deriving (Show)

-- berechnet Summe aller Werte in einem mit Zahlen beschrifteten Baum
sumTree :: BinTree Int → Int
sumTree Empty = 0
sumTree (Node v x y) = v + sumTree x + sumTree y

-- liefert alle Werte eines Baumes in einer Liste zurueck
values :: BinTree a → [a]
values Empty = []
values (Node v x y) = v : values x ++ values y

-- wendet eine gegebene Funktion auf alle Werte im Baum an
mapTree :: (a → b) → BinTree a → BinTree b
mapTree f Empty = Empty
mapTree f (Node v x y) = Node (f v) (mapTree f x) (mapTree f y)
```