

Programmierung WS 18

Hausaufgaben - Blatt 9

Julian Giesen (MNR 388487)
Levin Gäher (MNR 395035)
Gruppe 12

HA 2

```
-- a)
data VariableName = X | Y deriving Show

getValue :: VariableName -> Int
getValue X = 5
getValue Y = 13

-- b)
data Expression = Constant Int | Variable VariableName | Add Expression Expression |
  Multiply Expression Expression deriving Show

-- c)
evaluate :: Expression -> Int
evaluate (Constant c) = c
evaluate (Variable v) = getValue v
evaluate (Add ex1 ex2) = (evaluate ex1) + (evaluate ex2)
evaluate (Multiply ex1 ex2) = (evaluate ex1) * (evaluate ex2)

-- d)
tryOptimize :: Expression -> Expression
tryOptimize (Add (Constant c1) (Constant c2)) = Constant (c1 + c2)
tryOptimize (Multiply (Constant c1) (Constant c2)) = Constant (c1 * c2)
tryOptimize ex = ex

-- e)
evaluatePartially :: Expression -> Expression
evaluatePartially (Add ex1 ex2) = tryOptimize (Add (evaluatePartially ex1)
  (evaluatePartially ex2))
evaluatePartially (Multiply ex1 ex2) = tryOptimize (Multiply (evaluatePartially ex1)
  (evaluatePartially ex2))
evaluatePartially ex = ex

-- Example Provided
exampleExpression = Add
  ( Add
    ( Constant 20)
    ( Constant 17))
  ( Add
    ( Variable X )
    ( Multiply
      ( Add
        ( Constant 14)
        ( Constant 7))
      ( Constant 2)))
```

HA 4

$f :: a \rightarrow a \rightarrow [a] \rightarrow [a]$ $g :: [a] \rightarrow \text{Int} \rightarrow [\text{Int}] \rightarrow [\text{Int}]$ $h :: (a \rightarrow a) \rightarrow a \rightarrow [a] \rightarrow [a]$ // List of f applies to ys and then x at the end

HA 6

```
-- a)
data Optional a = Empty | Present a deriving Show

mapOptional :: (a -> b) -> Optional a -> Optional b
mapOptional _ Empty = Empty
mapOptional f (Present o) = Present (f o)

-- b)
filterOptional :: (a -> Bool) -> Optional a -> Optional a
filterOptional f Empty = Empty
filterOptional f (Present o) = if f o then Present o else Empty

-- c)
foldOptional :: (a -> b) -> b -> Optional a -> b
foldOptional _ x Empty = x
foldOptional f _ (Present o) = f o

-- d)
data Product = Article String Int deriving Show

isHumanEatable :: Product -> Bool
isHumanEatable (Article "Dog Food" _) = False
isHumanEatable _ = True

adjustPrice :: Product -> Product
adjustPrice (Article n p) = if p < 1000 then Article n (p*2) else Article n p

stringify :: Product -> String
stringify (Article n p) = "The Article named '" ++ n ++ "' costs " ++ show(p) ++ "
    Cents"

-- e)
filterHumanEatable :: Product -> Optional Product
filterHumanEatable a = filterOptional isHumanEatable (Present a)

adjustPrice0 :: Optional Product -> Optional Product
adjustPrice0 a = mapOptional adjustPrice a

stringify0 :: Optional Product -> String
stringify0 a = foldOptional stringify "This article is unavailable" a

toPriceTag :: Product -> String
toPriceTag a = stringify0 (adjustPrice0 (filterHumanEatable a))
```

HA 8