Haskell intro

Assignement0

Kai Arne Schroeder Myklebust, Silvan Robert Adrian

Handed in: September 12, 2018

Contents

1 Design/Implementation

2 Code Assessment

A Code Listing

```haskell
-- This is a skeleton file for you to edit

{-# OPTIONS_GHC -W #-}  -- Just in case you forgot...

module Arithmetic
  (
  showExp,
  evalSimple,
  extendEnv,
  evalFull,
```

```
11    evalErr,
12    showCompact,
13    evalEager,
14    evalLazy
15    )
16
17  where
18
19  import Definitions
20  import Data.Either
21
22  showExpStr :: Exp -> Exp -> String -> String
23  showExpStr a b s = "(" ++ showExp a ++ s ++ showExp b ++ ")"
24
25  showExp :: Exp -> String
26  showExp (Cst as) =
27    if head(show as) == '-' then "(" ++ show as ++ ")" else show as
28  showExp (Add a b) = showExpStr a b " + "
29  showExp (Sub a b) = showExpStr a b " - "
30  showExp (Mul a b) = showExpStr a b " * "
31  showExp (Div a b) = showExpStr a b " / "
32  showExp (Pow a b) = showExpStr a b "^"
33  showExp _ = error "is not supported"
34
35  evalSimple :: Exp -> Integer
36  evalSimple (Cst a) = a
37  evalSimple (Add a b) = evalSimple a + evalSimple b
38  evalSimple (Sub a b) = evalSimple a - evalSimple b
39  evalSimple (Mul a b) = evalSimple a * evalSimple b
40  evalSimple (Div a b) = evalSimple a `div` evalSimple b
41  evalSimple (Pow a b)
42    | evalSimple b < 0 = error "Negative exponent"
43    | evalSimple b == 0 = 1
44    | otherwise = evalSimple a * evalSimple(Pow a (Sub b (Cst 1)))
45  evalSimple _ = error "is not supported"
46  -- evalSimple (Pow a b) = evalSimple a ^ evalSimple b
47
48  extendEnv :: VName -> Integer -> Env -> Env
49  extendEnv v n r a = if v == a then Just n else r a
50
51  intTest :: Maybe Integer -> Integer
```

```haskell
intTest (Just i) = i
intTest _ = error "Value is unbound"

summ :: VName -> Integer -> Integer -> Exp -> Env -> Integer
summ v a b c r = if a > b then 0 else
  evalFull c r + summ v (a+1) b c (extendEnv v (a+1) r)

evalFull :: Exp -> Env -> Integer
evalFull (Cst a) _ = a
evalFull (Add a b) r = evalFull a r + evalFull b r
evalFull (Sub a b) r = evalFull a r - evalFull b r
evalFull (Mul a b) r = evalFull a r * evalFull b r
evalFull (Div a b) r = evalFull a r `div` evalFull b r
evalFull (Pow a b) r = evalFull a r ^ evalFull b r
evalFull (If a b c) r =
  if evalFull a r /= 0 then evalFull b r else evalFull c r
evalFull (Var v) r = intTest(r v)
evalFull (Let a b c) r = evalFull c (extendEnv a (evalFull b r)
  ↪  r)
evalFull (Sum v a b c) r =
  summ v (evalFull a r) (evalFull b r) c (extendEnv v (evalFull a
  ↪  r) r)

-- summ' :: VName -> Integer -> Integer -> Exp -> Env -> Integer
-- summ' v a b c r = if a > b then Right 0 else
--   Right (evalErr c r) + Right (summ' v (a+1) b c (extendEnv v
  ↪  (a+1) r))

intTestErr :: Maybe Integer -> VName -> Either ArithError Integer
intTestErr (Just i) _ = Right i
intTestErr _ v = Left (EBadVar v)

evalErr :: Exp -> Env -> Either ArithError Integer
evalErr (Cst a) _ = Right a
evalErr (Add a b) r = evalEither (evalErr a r) (+) (evalErr b r)
evalErr (Sub a b) r = evalEither (evalErr a r) (-) (evalErr b r)
evalErr (Mul a b) r = evalEither (evalErr a r) (*) (evalErr b r)
evalErr (Div a b) r = if isRight (evalErr b r)
                        then if fromRight' (evalErr b r) /= 0
                          then evalEither (evalErr a r) div
  ↪  (evalErr b r)
```

```
89                            else Left EDivZero
90                          else evalErr b r
91  evalErr (Pow a b) r = if isRight (evalErr b r)
92                          then if fromRight' (evalErr b r) >= 0
93                            then evalEither (evalErr a r) (^)
    ↪  (evalErr b r)
94                            else Left ENegPower
95                          else evalErr b r
96  evalErr (If a b c) r = if isRight (evalErr a r)
97                            then if fromRight' (evalErr a r) /= 0
98                              then evalErr b r
99                              else evalErr c r
100                         else evalErr a r
101 evalErr (Var v) r = intTestErr (r v) v
102 evalErr (Let a b c) r = if isRight (evalErr b r)
103                           then evalErr c (extendEnv a
    ↪  (fromRight'(evalErr b r)) r)
104                           else evalErr b r
105
106 evalErr (Sum v a b c) r = if isRight (evalErr a r)
107                             then if isRight (evalErr b r)
108                               then Right (summ v (fromRight'
    ↪  (evalErr a r)) (fromRight' (evalErr b r)) c (extendEnv v
    ↪  (fromRight'(evalErr a r)) r))
109                               else evalErr b r
110                             else evalErr a r
111
112 evalEither :: Either a b -> (b -> b -> b) -> Either a b -> Either
    ↪  a b
113 evalEither a b c = if isRight a
114                         then if isRight c
115                           then Right ( b (fromRight' a)
    ↪  (fromRight' c))
116                           else c
117                         else a
118
119 -- use own implementation of fromRight from Data.Either but not
    ↪  returning a
120 -- default value, which is not needed for the assignment
121 fromRight' :: Either a b -> b
122 fromRight' (Right c) = c
```

```haskell
fromRight' _ = error "No value"

-- optional parts (if not attempted, leave them unmodified)

showCompact :: Exp -> String
showCompact = undefined

evalEager :: Exp -> Env -> Either ArithError Integer
evalEager = undefined

evalLazy :: Exp -> Env -> Either ArithError Integer
evalLazy = undefined
```