Theory of Programming and Types, Exercise 3

Victor Cacciari Miraldo

No Institute Given

Prelude

```
{-# LANGUAGE GADTs #-}
{-# LANGUAGE KindSignatures #-}
{-# LANGUAGE RankNTypes #-}
{-# LANGUAGE FlexibleContexts #-}
{-# LANGUAGE TypeOperators #-}
{-# LANGUAGE TypeFamilies #-}
module Exercise where

import qualified Generics.Regular.Base as R
```

1 Exercise 1

- Kind Tree = $* \to * \to *$
- Kind GList = $(* \to *) \to * \to *$
- Kind Bush = $* \to *$
- Kind HFix = $((* \to *) \to * \to *) \to * \to *$
- Kind Exists = $* \to *$
- Kind Exp = $*$

2 Exercise 2

2.1 A

```
data Exp where
  EBool :: Bool → Exp
  EInt :: Int → Exp
  Gt :: Exp → Exp → Exp
  Add :: Exp → Exp → Exp
  IsZero :: Exp → Exp
  Succ :: Exp → Exp
  If :: Exp → Exp → Exp → Exp

pair :: a → b → (a, b)
pair a b = (a, b)
```

```haskell
evalInt :: Exp → Maybe Int
evalInt e = eval e ≫= either return (const Nothing)

evalBool :: Exp → Maybe Bool
evalBool e = eval e ≫= either (const Nothing) return

eval :: Exp → Maybe (Either Int Bool)
eval (EBool b) = return (Right b)
eval (EInt n) = return (Left n)
eval (Add a b) = evalInt a ≫= λav → evalInt b
                                ≫= return ∘ Left ∘ (av+)
eval (Gt a b) = evalInt a ≫= λav → evalInt b
                              ≫= return ∘ Right ∘ (av ⩾)
eval (IsZero a) = evalInt a ≫= return ∘ Right ∘ (≡ 0)
eval (Succ a) = evalInt a ≫= return ∘ Left ∘ (+1)
eval (If c t e)
  = do
    b ← evalBool c
    if b then eval t else eval e

testExp :: Exp
testExp = If (IsZero (Succ (EInt 3))) (EInt 10) (EInt 15)
```

2.2 B

```haskell
data ExpF a
  = EFInt Int
  | EFBool Bool
  | EGt a a
  | EAdd a a
  | EIsZero a
  | ESucc a
  | EIf a a a

evalAlg :: ExpF (Maybe (Either Int Bool)) → Maybe (Either Int Bool)
evalAlg (EFInt n) = Just (Left n)
evalAlg (EFBool b) = Just (Right b)
evalAlg (EGt (Just (Left n)) (Just (Left m))) = Just (Right (n ⩾ m))
evalAlg (EAdd (Just (Left n)) (Just (Left m))) = Just (Left (m + n))
evalAlg (EIsZero (Just (Left n))) = Just (Right (n ≡ 0))
evalAlg (ESucc (Just (Left n))) = Just (Left (n + 1))
evalAlg (EIf (Just (Right b)) t e) = if b then t else e
evalAlg _ = Nothing
```

2.3 C

```haskell
instance Functor ExpF where
  fmap f (EGt a b) = EGt (f a) (f b)
  fmap f (EAdd a b) = EAdd (f a) (f b)
  fmap f (EIsZero x) = EIsZero (f x)
  fmap f (ESucc x) = ESucc (f x)
  fmap f (EIf c t e) = EIf (f c) (f t) (f e)
  fmap f (EFInt n) = EFInt n
  fmap f (EFBool b) = EFBool b
```

```haskell
newtype Fix f = In
```

2.4 D

```haskell
data HFix f a = HIn
```

2.5 E

```haskell
newtype Id a = Id
```

3 Exercise 3

```haskell
class Children f where
  getChildren :: f a -> [a]
```

```haskell
instance Children (R.K v) where
  getChildren (R.K a) = []
```

```haskell
instance Children R.I where
  getChildren (R.I r) = [r]
```

```haskell
instance Children R.U where
  getChildren R.U = []
```

```haskell
instance (Children f, Children g) => Children (f R.:+: g) where
  getChildren (R.L fr) = getChildren fr
  getChildren (R.R gr) = getChildren gr
```

```haskell
instance (Children f, Children g) => Children (f R.:*: g) where
  getChildren (fr R.:*: gr) = getChildren fr ++ getChildren gr
```

```haskell
instance (Children f) => Children (R.C c f) where
  getChildren (R.C fr) = getChildren fr
```

-- List instance of PF

$$
\begin{array}{l}
\textbf{type instance}\ \mathit{R.PF}\ [a] = \mathit{R.U}\ \mathit{R.}:+:((\mathit{R.K}\ a)\ \mathit{R.}:*:\mathit{R.I})\\
\textbf{instance}\ \mathit{R.Regular}\ [a]\ \textbf{where}\\
\quad \mathit{from}\ [\,] = \mathit{R.L}\ \mathit{R.U}\\
\quad \mathit{from}\ (a:\mathit{as}) = \mathit{R.R}\ ((\mathit{R.K}\ a)\ \mathit{R.}:*:(\mathit{R.I}\ \mathit{as}))\\
\\
\quad \mathit{to}\ (\mathit{R.L}\ \mathit{R.U}) = [\,]\\
\quad \mathit{to}\ (\mathit{R.R}\ ((\mathit{R.K}\ a)\ \mathit{R.}:*:(\mathit{R.I}\ \mathit{as}))) = a:\mathit{as}
\end{array}
$$

$$
\begin{array}{l}
\mathit{children} :: (\mathit{R.Regular}\ r, \mathit{Children}\ (\mathit{R.PF}\ r)) \Rightarrow r \rightarrow [r]\\
\mathit{children} = \mathit{getChildren} \circ \mathit{R.from}
\end{array}
$$

$$
\begin{array}{l}
\mathit{example3} :: \mathit{Bool}\\
\mathit{example3} = (\mathit{children}\ [1,2] \equiv [[2]])
\end{array}
$$

4 Exercise 4

$$
\begin{array}{l}
\mathit{parents} :: (\mathit{R.Regular}\ r, \mathit{Children}\ (\mathit{R.PF}\ r)) \Rightarrow r \rightarrow [r]\\
\mathit{parents}\ r = \textbf{case}\ \mathit{children}\ r\ \textbf{of}\\
\qquad\qquad\qquad [\,] \rightarrow [\,]\\
\qquad\qquad\qquad \mathit{cs} \rightarrow r:(\mathit{concatMap}\ \mathit{parents}\ \mathit{cs})
\end{array}
$$

$$
\begin{array}{l}
\mathit{example4} :: \mathit{Bool}\\
\mathit{example4} = (\mathit{parents}\ [1,2,3] \equiv [[1,2,3],[2,3],[3]])
\end{array}
$$