

Real World Haskell

Blatt 8

Julian Fleischer, Alexander Steen

Dienstag, den 06.08.2013

Aufgabe 1 (Existentielle Datentypen)

```
1 {-# LANGUAGE Haskell2010, ExistentialQuantification #-}
2
3 import Prelude hiding (lookup)
4
5 data Map k v = forall m. MkMap {
6   _self :: m
7   , _insert :: m -> k -> v -> m
8   , _lookup :: m -> k -> Maybe v
9   , _delete :: m -> k -> m
10 }
11
12 insert (MkMap this ins lup del) k v
13   = MkMap (ins this k v) ins lup del
14 delete (MkMap this ins lup del) key
15   = MkMap (this 'del' key) ins lup del
16 lookup (MkMap this ins lup del) key
17   = this 'lup' key
```

Implementieren Sie `newListMap :: Map k v`, die eine Map mit Hilfe einer assoziativen `[(k, v)]` verwaltet (Alternativ können Sie auch `newTreeMap` mit einer `TreeMap` aus `Data.Map` oder `newHashMap` mit einer `HashMap` aus `Data.HashMap` implementieren – oder alle drei!)

Aufgabe 2 (Phantom Typen)

```
1 {-# LANGUAGE Haskell2010, GADTs, DataKinds, KindSignatures #-}
2
3 data Nat :: * where
4   Z :: Nat
5   S :: Nat -> Nat
6
7 data Vect :: * -> Nat -> * where
8   Empty :: Vect a Z
9   VCons :: a -> Vect a n -> Vect a (S n)
```

Welche Erweiterungen werden für welchen Teil der obigen Deklaration benötigt?

Deklariieren Sie `safeHead` und `safeTail` die die sicheren Varianten von `head` und `tail` für Vektoren sind. Welche Typen haben diese Funktionen?

Aufgabe 3 (Haskell 98 Style Typklassen)

```
1 {-# LANGUAGE Haskell2010, FlexibleInstances #-}
2
3 class Tautology a where
4     isTautology :: a -> Bool
5
6 ...
7
8 main = do
9     print $ isTautology (\toBe -> toBe || not toBe)
10    print $ isTautology (\x -> x && x)
11    print $ isTautology (\x y -> (not x && not y) == not (x && y))
12    print $ isTautology (\x y -> (not x && not y) == not (x || y))
```

Schreiben Sie adequate Instanzen der Typklasse `Tautology`. Die Funktion `isTautology` bekommt eine Funktion mit Ergebnis-Typ `Bool` und prüft, ob diese für alle Eingabewerte `True` zurückliefert (ob die logische Aussage eine Tautologie ist).

Aufgabe 4 (Matrixenmultiplikation)

```
1 {-# LANGUAGE Haskell2010 #-}
2
3 data Z
4 data S n
5
6 newtype Mat a m n = Mat [[a]]
```

Schreiben Sie die obige Definition unter Verwendung von `XDataKinds` und dem Datentyp `Nat` im Stile des Beispielcodes der 2. Aufgabe um.

Schreiben Sie eine typsichere Matrizen-Multiplikation, die die Dimensionen der Eingabe-Matrizen und Ausgabe-Matrizen im Typen berücksichtigt!

Sie könne als zugrunde liegende Datenstruktur auch Arrays statt Listen verwenden. Siehe `Data.Array`.

Aufgabe 5 (Strict State Threads)

Machen Sie sich mit der `ST` Monade¹ vertraut (`Control.Monad.ST`). Schreiben Sie Aufgabe 5 unter Verwendung von `ST` und einem veränderbaren (*mutable*) `Array`² um!

Welchen Sinn hat der existenzielle Typ `s` im Typ von `runST`?

¹<http://hackage.haskell.org/packages/archive/base/4.5.0.0/doc/html/Control-Monad-ST.html>

²<http://hackage.haskell.org/packages/archive/array/0.4.0.1/doc/html/Data-Array-ST.html>