# Real World Haskell

*Blatt 8*

Julian Fleischer, Alexander Steen

Dienstag, den 06.08.2013

**Aufgabe 1** (**Existentielle Datentypen**)

```
{-# LANGUAGE Haskell2010, ExistentialQuantification #-}

import Prelude hiding (lookup)

data Map k v = forall m. MkMap {
    _self :: m
  , _insert :: m -> k -> v -> m
  , _lookup :: m -> k -> Maybe v
  , _delete :: m -> k -> m
}

insert (MkMap this ins lup del) k v
    = MkMap (ins this k v) ins lup del
delete (MkMap this ins lup del) key
    = MkMap (this `del` key) ins lup del
lookup (MkMap this ins lup del) key
    = 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**)

```
{-# LANGUAGE Haskell2010, GADTs, DataKinds, KindSignatures #-}

data Nat :: * where
    Z :: Nat
    S :: Nat -> Nat

data Vect :: * -> Nat -> * where
    Empty :: Vect a Z
    VCons :: a -> Vect a n -> Vect a (S n)
```

Welche Erweiterungen werden für welchen Teil der obigen Deklaration benötigt?

Deklarieren 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**)

```
{-# LANGUAGE Haskell2010, FlexibleInstances #-}

class Tautology a where
    isTautology :: a -> Bool

...

main = do
    print $ isTautology (\toBe -> toBe || not toBe)
    print $ isTautology (\x -> x && x)
    print $ isTautology (\x y -> (not x && not y) == not (x && y))
    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**)

```
{-# LANGUAGE Haskell2010 #-}

data Z
data S n

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

**Aufgabe 5** (**Strict State Threads**)

Machen Sie sich mit der `ST` Monade[1] 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`