technische universität dortmund

FLDIT

Lehrstuhl 1
Logik in der Informatik

Prof. Dr. Peter Padawitz

Jos Kusiek (*jos.kusiek@tu-dortmund.de*) Wintersemester 2017/2018

Übungen zu Funktionaler Programmierung
Übungsblatt 9

Ausgabe: 15.12.2017, **Abgabe:** 22.12.2017 – 16:00 Uhr, **Block:** 5

Aufgabe 9.1 (4 Punkte)

a) Zeigen Sie, dass die Rekursionsgleichung `fib` eine Funktion definiert. Definieren Sie dazu eine Schrittfunktion Φ analog zu der auf Folie 146.

b) Beweisen Sie durch Induktion, dass lfp(Φ) keine natürliche Zahl auf $\perp$ abbildet.

Aufgabe 9.2 (3 Punkte)
Definieren Sie folgende Haskell-Funktionen.

a) `isCyclic :: Eq a => Graph a -> Bool` – Erkennt, ob ein Graph zyklisch ist. Sie können hier den transitiven Abschluss nutzen.
Beispiele:
`isCyclic graph1` $\rightsquigarrow$ `True`
`isCyclic graph2` $\rightsquigarrow$ `False`

b) `depthFirst :: Eq a => a -> Graph a -> [a]` – Ähnlich wie `preorder` und `postorder` auf Bäumen, sollen die Knoten des Graphen in einer bestimmten Reihenfolge als Liste ausgegeben werden. Die Funktion erhält einen Startknoten und gibt dann weitere Knoten durch Tiefensuche aus.

Aufgabe 9.3 (2 Punkte) *Kinds*
Bestimmen Sie den Kind folgender Typkonstruktoren.

a)
```
class C f where
  comp :: f b c -> f a b -> f a c
```
Bestimmen Sie den Kind von `f`.

b)
```
data T f g = T (f String Int) (g Bool)
```
Bestimmen Sie den Kind von `T`.

Aufgabe 9.4 (3 Punkte) *Typfamilien*
Gegeben sei folgende Typklasse:

```
class Listable l where
  type Item l :: *
  toList :: l -> [Item l]
```

Die Funktion `toList` wandelt eine Eingabe in eine Liste um. Überladen Sie die Funktion für die angegebenen Typen.

a) `Colist a` – Gibt die zugehörige Liste aus.

b) `data Map a b = Map [(a,b)]` – Ein Datentyp für eine Assoziationsliste (siehe Folie 53). Es sollen in der Liste nur die Werte ausgegeben werden. Die Argumente bzw. Schlüssel entfallen.

c) `Nat` – Die bijektive Funktion der Isomorphie von `Nat` und `[()]`.

Hinweis: Die Typklasse macht gebrauch von einer Typfamilie. Um Typfamilien zu nutzen, muss die Spracherweiterung *TypeFamilies* aktiviert werden. Fügen Sie das Pragma

```
{-# LANGUAGE TypeFamilies #-}
```

an den Kopf Ihrer Haskell-Datei ein.