

Einführung in die funktionale Programmierung

Wintersemester 2021/2022

Aufgabenblatt Nr. 5

Abgabe: Montag 17. Januar 2021, Besprechung 21. Januar 2022

Das Aufgabenblatt 50 Punkte.

Aufgabe 1 (15 Punkte)

Überführen Sie die List Comprehension

```
[ x*3 | z <- [ [1..u] | u <- [3..30], even u], x <- z]
```

mit dem in der Vorlesung vorgestellten Verfahren in ZF-freies Haskell. Testen Sie die Gleichheit der Ergebnisse der Ausführung.

Aufgabe 2 (10 Punkte Bonus)

Der Datentyp `Expr a` von Aufgabenblatt 3 zur Darstellung von KFPT-Ausdrücken ist:

```
> data Expr a =
>   Var a                -- x      = Var "x"
> | App (Expr a) (Expr a) -- (e1 e2) = App e1 e2
> | Lam a (Expr a)        -- \x.e   = Lam "x" e
> | ListCons (Expr a) (Expr a) -- a:as  = ListCons a as
> | ListNil               -- []      = ListNil
> | BoolTrue              -- True    = BoolTrue
> | BoolFalse             -- False   = BoolFalse
> | CaseList (Expr a) (Expr a) (a,a,Expr a) -- case_List e of {[] -> e1; (x:xs) -> e2}
>                                           -- = CaseList e e1 ("x",xs",e2)
> | CaseBool (Expr a) (Expr a) (Expr a)    -- case_Bool e of {True -> e1; False -> e2}
>                                           -- = CaseBool e e1 e2
```

Sei der Typ für Variablen definiert als

```
> newtype Varname = Varname String
```

Definieren sie je eine Instanz der Typklasse `Show` für den Typ `Expr a` und den Typ `Varname`, sodass der Ausgabestring der `show`-Funktion für jeden Ausdruck vom Typ `Expr Varname` einen syntaktisch korrekten Haskell-Ausdruck gleicher Bedeutung darstellt (z.B. werden Abstraktionen als `\x -> e` dargestellt).

Da der Typ `Expr` polymorph über Variablennamen ist, müssen Sie bei der Instanzdefinition zusätzlich als Klassenbedingung fordern, dass bereits eine `Show`-Instanz für `a` existiert, d.h. die Instanzdefinition beginnt mit

```
instance Show a => Show (Expr a) where
```

Aufgabe 3 (35 Punkte)

Polymorphe binäre Bäume `B Baum` und `n`-äre Bäume `N Baum` seien in Haskell definiert als:

```
> data B Baum a = BBlatt a                -- Blatt mit Markierung
>                | BKnoten a (B Baum a) (B Baum a) -- Markierung, linker u. rechter Teilbaum
> deriving(Show)

> data N Baum a = NBlatt a                -- Blatt mit Markierung
>                | NKnoten a [N Baum a]    -- Markierung und Liste der Kinder
> deriving(Show)
```

- a) Definieren Sie in Haskell eine Konstruktorklasse `Ist Baum`, die Operationen für Baum-artige Datentypen überlädt. Als Klassenmethoden sollen dabei zur Verfügung stehen:

- `teilbaeume` liefert die Liste der Unterbäume der Wurzel.
- `istBlatt` testet, ob ein Baum nur aus einem Blatt besteht und liefert dementsprechend `True` oder `False`. (8 Punkte)

- b) Definieren Sie eine Unterklasse `IstMarkierter Baum` von `Ist Baum`, die Operatoren für Bäume mit Knotenmarkierungen überlädt. Die Klassenmethoden sind:

- `markierung` liefert die Beschriftung der Wurzel.
- `knoten` liefert alle Knotenmarkierungen eines Baums als Liste.
- `kanten` liefert alle Kanten des Baumes, wobei eine Kante als Paar von Knotenmarkierungen dargestellt wird.

Geben Sie dabei Default-Implementierungen für `knoten` und `kanten` innerhalb der Klassendefinition an. (12 Punkte)

- c) Implementieren Sie Instanzen der Klassen `Ist Baum` und `IstMarkierter Baum` jeweils für die Datentypen `B Baum` und `N Baum`. (10 Punkte)

Für die beiden folgenden Beispieltäume:

```
> bspB Baum = BKnoten 1 (BKnoten 2 (BBlatt 3) (BBlatt 4)) (BKnoten 5 (BBlatt 6) (BBlatt 7))
> bspN Baum = NKnoten 1 [NKnoten 2 [NBlatt 3,NBlatt 4, NBlatt 5]
>                ,NKnoten 6 [NBlatt 7, NKnoten 8 [NBlatt 9, NBlatt 10]]]
```

verdeutlichen die folgenden Beispielaufufe die geforderten Funktionalitäten:

```
*Main> teilbaeume bspB Baum
[BKnoten 2 (BBlatt 3) (BBlatt 4),BKnoten 5 (BBlatt 6) (BBlatt 7)]
*Main> teilbaeume bspN Baum
[NKnoten 2 [NBlatt 3,NBlatt 4,NBlatt 5],NKnoten 6 [NBlatt 7,NKnoten 8 [NBlatt 9,NBlatt 10]]]
*Main> knoten bspB Baum
[1,2,3,4,5,6,7]
*Main> knoten bspN Baum
[1,2,3,4,5,6,7,8,9,10]
*Main> kanten bspB Baum
[(1,2),(1,5),(2,3),(2,4),(5,6),(5,7)]
*Main> kanten bspN Baum
[(1,2),(1,6),(2,3),(2,4),(2,5),(6,7),(6,8),(8,9),(8,10)]
*Main> map markierung (teilbaeume bspN Baum)
[2,6]
*Main> map markierung (teilbaeume bspB Baum)
[2,5]
```