Prof. Dr. Manfred Schmidt-Schauß
Institut für Informatik

Fachbereich Informatik und Mathematik/ Institut für Informatik
Johann Wolfgang Goethe-Universität Frankfurt am Main

Einführung in die funktionale Programmierung

Wintersemester 2023/2024

Aufgabenblatt Nr. 5

Abgabe: Dienstag 9. Januar 2024

Das Aufgabenblatt hat 50 Punkte.

Aufgabe 1 (15 Punkte)

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 wie unten gezeigt.

Hinweise zur Lösung:

```haskell
newtype Varname = Varname String
instance Show Varname where
  show (Varname a) = a

instance Show a => Show (Expr a) where
  show (Var x)          = show x
    ........................
```

Lösung:

```haskell
newtype Varname = Varname String
instance Show Varname where
  show (Varname a) = a

instance Show a => Show (Expr a) where
  show (Var x)          = show x
  show (App e1 e2)      = "(" ++ show e1 ++ " " ++ show e2 ++ ")"
  show (Lam x e)        = "(" ++ "\\" ++ show x ++ " -> " ++ show e ++ ")"
  show (ListCons a as) = "(" ++ show a ++ ":" ++ show as ++ ")"
  show ListNil          = "[]"
  show BoolTrue         = "True"
  show BoolFalse        = "False"
  show (CaseList e e1 (x,xs,e2)) = "(case " ++ show e ++ " of [] -> " ++
      show e1 ++ "; (" ++ show x ++ ":" ++ show xs ++ ") -> " ++ show e2 ++
       ")"
  show (CaseBool e e1 e2)        = "(case " ++ show e ++ " of True -> " ++
      show e1 ++ "; False -> " ++ show e2 ++ ")"
```

Aufgabe 2 (35 Punkte)

Polymorphe binäre Bäume `BBaum` und n-äre Bäume `NBaum` seien in Haskell definiert als:

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

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

a) Definieren Sie in Haskell eine Konstruktorklasse `IstBaum`, 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 `IstMarkierterBaum` von `IstBaum`, 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 `IstBaum` und `IstMarkierterBaum` jeweils für die Datentypen `BBaum` und `NBaum`. (10 Punkte)

Lösung:

```haskell
class IstBaum b where
  teilbaeume :: (b a) -> [(b a)]
  istBlatt   :: (b a) -> Bool

class IstBaum b => IstMarkierterBaum b where
  markierung :: (b a) -> a
  knoten     :: (b a) -> [a]
  kanten     :: (b a) -> [(a,a)]
  knoten k = [markierung k] ++ concat (map (\t -> knoten t) (teilbaeume k))
  kanten k = map (\t -> (markierung k,markierung t)) (teilbaeume k) ++
      concat (map (\t -> kanten t) (teilbaeume k))

instance IstBaum BBaum where
  teilbaeume (BBlatt _)      = []
  teilbaeume (BKnoten _ b c) = [b,c]

  istBlatt (BBlatt _) = True
  istBlatt _          = False

instance IstBaum NBaum where
  teilbaeume (NBlatt _)      = []
  teilbaeume (NKnoten _ xxs) = xxs

  istBlatt (NBlatt _) = True
  istBlatt _          = False

-- Die Defaultimplementierungen von knoten und kanten müssen nicht
    überschrieben werden.

instance IstMarkierterBaum BBaum where
  markierung (BBlatt l)      = l
  markierung (BKnoten a _ _) = a

instance IstMarkierterBaum NBaum where
  markierung (NBlatt l)    = l
  markierung (NKnoten a _) = a
```

Für die beiden folgenden Beispielbäume:

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

verdeutlichen die folgenden Beispielaufrufe die geforderten Funktionalitäten:

```
*Main> teilbaeume bspBBaum
[BKnoten 2 (BBlatt 3) (BBlatt 4),BKnoten 5 (BBlatt 6) (BBlatt 7)]
*Main> teilbaeume bspNBaum
[NKnoten 2 [NBlatt 3,NBlatt 4,NBlatt 5],NKnoten 6 [NBlatt 7,NKnoten 8 [NBlatt 9,NBlatt 10]]]
*Main> knoten bspBBaum
[1,2,3,4,5,6,7]
*Main> knoten bspNBaum
[1,2,3,4,5,6,7,8,9,10]
*Main> kanten bspBBaum
[(1,2),(1,5),(2,3),(2,4),(5,6),(5,7)]
*Main> kanten bspNBaum
[(1,2),(1,6),(2,3),(2,4),(2,5),(6,7),(6,8),(8,9),(8,10)]
```

```
*Main> map markierung (teilbaeume bspNBaum)
[2,6]
*Main> map markierung (teilbaeume bspBBaum)
[2,5]
```