

Einführung in die funktionale Programmierung

Wintersemester 2025/2026

Aufgabenblatt Nr. 5

Abgabe: Dienstag 9. Januar 2026

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:

```
1 newtype Varname = Varname String
2 instance Show Varname where
3   show (Varname a) = a
4
5 instance Show a => Show (Expr a) where
6   show (Var x)      = show x
7   .....
```

Lösung:

```

1 newtype Varname = Varname String
2 instance Show Varname where
3     show (Varname a) = a
4
5 instance Show a => Show (Expr a) where
6     show (Var x)           = show x
7     show (App e1 e2)       = "(" ++ show e1 ++ " " ++ show e2 ++ ")"
8     show (Lam x e)         = "(" ++ "\\" ++ show x ++ " -> " ++ show e ++ ")"
9     show (ListCons a as)   = "(" ++ show a ++ ":" ++ show as ++ ")"
10    show ListNil            = "[]"
11    show BoolTrue           = "True"
12    show BoolFalse          = "False"
13    show (CaseList e e1 (x,xs,e2)) = "(case " ++ show e ++ " of [] -> " ++
        show e1 ++ "; (" ++ show x ++ ":" ++ show xs ++ ") -> " ++ show e2 ++
        ")"
14    show (CaseBool e e1 e2) = "(case " ++ show e ++ " of True -> " ++
        show e1 ++ "; False -> " ++ show e2 ++ ")"

```

Aufgabe 2 (35 Punkte)

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

```

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

> data N Baum a = N Blatt a                -- Blatt mit Markierung
>               | N Knoten 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)

Lösung:

```

1  class IstBaum b where
2      teilbaeume :: (b a) -> [(b a)]
3      istBlatt   :: (b a) -> Bool
4
5  class IstBaum b => IstMarkierterBaum b where
6      markierung :: (b a) -> a
7      knoten     :: (b a) -> [a]
8      kanten     :: (b a) -> [(a,a)]
9      knoten k = [markierung k] ++ concat (map (\t -> knoten t) (teilbaeume k))
10     kanten k = map (\t -> (markierung k,markierung t)) (teilbaeume k) ++
11               concat (map (\t -> kanten t) (teilbaeume k))
12
13 instance IstBaum BBaum where
14     teilbaeume (BBlatt _)      = []
15     teilbaeume (BKnoten _ b c) = [b,c]
16
17     istBlatt (BBlatt _) = True
18     istBlatt _          = False
19
20 instance IstBaum NBaum where
21     teilbaeume (NBlatt _)      = []
22     teilbaeume (NKnoten _ xxs) = xxs
23
24     istBlatt (NBlatt _) = True
25     istBlatt _          = False
26
27 -- Die Defaultimplementierungen von knoten und kanten müssen nicht
28 -- überschrieben werden.
29
30 instance IstMarkierterBaum BBaum where
31     markierung (BBlatt l)      = l
32     markierung (BKnoten a _ _) = a
33
34 instance IstMarkierterBaum NBaum where
35     markierung (NBlatt l)      = l
36     markierung (NKnoten a _)   = a

```

Für die beiden folgenden Beispielsäume:

```

> 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 Beispielaufufe 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]
```