Programmierung

03. Übungsblatt

Zeitraum: 25. – 29. April 2016

Übung 1 (AGS 12.1.41)

(a) Geben Sie eine Funktion `pack :: [Int] -> [Int]` an, welche in einer Liste eine Folge direkt aufeinanderfolgender Wiederholungen einer Zahl durch die Zahl selbst ersetzt. So soll z. B. `pack [1,5,5,5,2,7,7,1,1] = [1,5,2,7,1]` gelten.

(b) Gegeben sei der polymorphe algebraische Datentyp

```
data Tree a = Branch a (Tree a) (Tree a) | Leaf a
```

Geben Sie eine Funktion `depth :: Tree a -> Int` an, welche für einen Baum `t` die Länge des kürzesten Pfads von der Wurzel zu einem Blattknoten von `t` berechnet. Die Länge eines Pfads ist dabei die Anzahl der auf ihm vorkommenden Knoten, d.h. der Pfad von der Wurzel zur Wurzel selbst hat die Länge 1.

(c) Gegeben sei der algebraische Datentyp `Tree` aus Teilaufgabe **(b)**. Geben Sie eine Funktion `paths :: Tree a -> Tree [a]` an, die in einem Baum die Beschriftung jedes Knotens u durch die Liste der Knotenbeschriftungen auf dem Pfad vom Wurzelknoten zu u ersetzt. Ist der Wurzelknoten z. B. in `t` mit `5` beschriftet, so ist er in `paths t` mit `[5]` beschriftet, und ist sein erster Kindknoten in `t` mit `3` beschriftet, dann ist dieser in `paths t` mit `[5,3]` beschriftet.

Übung 2

Aus der Vorlesung kennen Sie die folgenden polymorphen Funktionen.

```
filter :: (a -> Bool) -> [a] -> [a]
map    :: (a -> b) -> [a] -> [b]
foldr  :: (a -> b -> b) -> b -> [a] -> b
```

Definieren Sie `map` und `filter` jeweils mithilfe von `foldr`!

Übung 3 (AGS 12.2.13)

(a) Gegeben seien folgende Terme über dem Rangalphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$:

$$t_1 = \sigma\big(\gamma(x_2),\ \sigma\big(\gamma(\alpha),\ x_3\qquad\big)\big),$$
$$t_2 = \sigma\big(x_1\quad,\ \sigma\big(\gamma(\alpha),\ \sigma(\alpha, x_1)\big)\big).$$

Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an. Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.

(b) Geben Sie zwei weitere Unifikatoren an.

Übung 4
Nutzen Sie den Unifikationsalgorithmus zur Bestimmung des Typs des Haskell-Terms

```
map snd [(5,'c'), (1,'b')]
```

mit

```
map :: (a -> b) -> [a] -> [b]
snd :: (p, q) -> q
```

Nehmen Sie dabei an, dass `[(5,'c'), (1,'b')] :: [(Int, Char)]`.

Zusatzaufgabe 1 (AGS 12.2.12)

(a) Gegeben seien folgende Terme über dem Rangalphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$:

$$t_1 = \sigma\Big(\sigma\big(x_1 \quad, \alpha\big), \sigma\big(\gamma(x_3), x_3\big)\Big),$$
$$t_2 = \sigma\Big(\sigma\big(\gamma(x_2), \alpha\big), \sigma\big(x_2 \quad, x_3\big)\Big).$$

Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an, wie in Aufg. 3. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.

(b) Geben Sie zwei weitere Unifikatoren an.

Zusatzaufgabe 2 (AGS 12.1.43 ⋆)
Sie dürfen alle üblichen Haskellfunktionen aus der Standardbibliothek `Prelude` verwenden.

(a) Sei `xs` eine Liste. Eine Teilliste von `xs` ist eine Liste, die durch Löschen von beliebig vielen Elementen aus `xs` entsteht. Die Teillisten von `[1, 2, 3]` sind beispielsweise `[]`, `[1]`, `[2]`, `[3]`, `[1, 2]`, `[1, 3]`, `[2, 3]` und `[1, 2, 3]`.
Schreiben Sie die Funktion `isSubseqOf :: Eq a => [a] -> [a] -> Bool`, die prüft, ob die erste übergebene Liste eine Teilliste der zweiten ist.

(b) Gegeben seien folgende Funktionen.

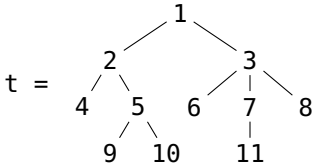
```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith f (x : xs) (y : ys) = f x y : zipWith f xs ys
zipWith _ _        _        = []

h :: [Int] -> [Int] -> [Int]
h xs ys = zipWith (\ x y -> 2 * x + y) xs ys
```

Berechnen Sie schrittweise `h [1, 2] [3, 4, 5]`. Sie dürfen dabei sinnvolle Abkürzungen einführen.

(c) Gegeben sei folgender algebraischer Datentyp für beliebig verzweigende Bäume.

```
data Tree a = Node a [Tree a]
```

Ein Pfad eines Baumes ist die Liste aller Knotenbeschriftungen auf dem Weg von der Wurzel des Baumes zu einem Knoten. (Daraus folgt, dass jeder Pfad mindestens ein Element enthält, nämlich die Beschriftung des Wurzelknotens.)
Implementieren Sie die Funktion `isPathOf :: Eq a => [a] -> Tree a -> Bool`, welche prüft, ob ein Baum einen gegebenen Pfad beinhaltet. Zum Beispiel:




```
isPathOf [] t = False
isPathOf [1, 2] t = True
isPathOf [1, 2, 9] t = False
isPathOf [1, 2, 5, 9] t = True
```

Zusatzaufgabe 3 (AGS 12.1.39 ⋆)
Wir wollen die Darstellung von arithmetischen Ausdrücken aus der Vorlesung nun um Variablen und um Exponentiation erweitern, d.h. wir betrachten den folgenden Datentyp.

```
data Expr = Lit Int
          | Var Char
          | Add Expr Expr
          | Mul Expr Expr
          | Exp Expr Int
type Assignment = Char -> Int
```

(a) Erweitern Sie die Auswertungsfunktion `eval` aus der Vorlesung um die Auswertung von Variablenbelegungen, d.h. zu einer Funktion `eval :: Expr -> Assignment -> Int`. Dabei soll eine Variable `Var 'x'` zu ihrem Wert unter der übergebenen Variablenbelegung evaluiert werden.

(b) Schreiben Sie eine Funktion `display :: Expr -> String`, welche den übergebenen Ausdruck als String darstellt!

(c) *(Zusatzaufgabe)* Schreiben Sie nun eine Funktion `diff :: Expr -> Char -> Expr`, so dass `diff e 'x'` eine (symbolische) Ableitung des Ausdrucks `e` nach `x` ist. So könnte der Aufruf von `diff e 'x'` mit

```
e = Add (Mul (Lit 5) (Exp (Var 'x') 2))
        (Mul (Var 'x') (Var 'y'))
```

beispielsweise den Wert

```
Add (Add (Mul (Lit 0) (Exp (Var 'x') 2))
         (Mul (Lit 5) (Mul (Mul (Lit 2) (Exp (Var 'x') 1))
                           (Lit 1))))
    (Add (Mul (Lit 1) (Var 'y')) (Mul (Var 'x') (Lit 0)))
```

ergeben.