Philipps Universität Marburg

Übungen zu „Konzepte von Programmiersprachen“, WS 2010/11

Prof. Dr. R. Loogen · Fachbereich Mathematik und Informatik · Marburg

Nr. 4, Abgabe: Dienstag, 16. November 2010 vor der Vorlesung

8. Ausdrucksvereinfachung — 3 Punkte

Gegeben sei der Typ `Expr` zur Definition von einfachen arithmetischen Ausdrücken:

```
data Expr = Var String | Val Int | Add Expr Expr | Mult Expr Expr
          deriving Show
```

Definieren Sie eine Funktion `simplify :: Expr -> Expr`, die in einem Ausdruck konstante Teilausdrücke durch ihren Wert ersetzt.

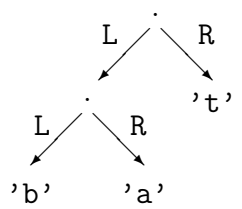
Beispiel:
```
simplify (Add (Mult (Val 5) (Val 3)) (Var "x"))
=>* (Add (Val 15) (Var "x"))
```

9. Huffman-Kodierung/Dekodierung — 6 Punkte

Gegeben seien die folgenden Typfestlegungen:

```
data BinTree  a = Leaf a |
                  Node (BinTree a) (BinTree a)
                  deriving Show
data Bit        = L | R deriving Show
type Table a    = [ (a, [Bit]) ]
```

Beispielbaum mit Pfadmarkierung:



Geben Sie Haskell-Definitionen der folgenden Funktionen an:

(a) `codeTable :: BinTree a -> Table a` wandelt einen Baum vom Typ `BinTree a` in eine Tabelle vom Typ `Table a` um, in der zu jedem Eintrag vom Typ `a` der Pfad von der Wurzel zu der Position des Eintrags beschrieben wird. / 2

Beispiel:
```
codeTable (Node (Node (Leaf 'b') (Leaf 'a')) (Leaf 't'))
=>* [('b',[L,L]),('a',[L,R]),('t',[R])]
```

(b) `code :: Eq a => Table a -> [a] -> [Bit]` wandelt mit einer Tabelle eine Elementliste in eine Bitliste um. / 2

Beispiel:
```
code [('b',[L,L]),('a',[L,R]),('t',[R])] "tabat"
=>* [R,L,R,L,L,L,R,R]
```

(c) `decode :: BinTree a -> [Bit] -> [a]` bestimmt mit einem Baum vom Typ `BinTree a` zu einer Bitfolge die ursprüngliche Liste. / 2

Beispiel:
```
decode  (Node (Node (Leaf 'b') (Leaf 'a')) (Leaf 't'))
        [R,L,R,L,L,L,R,R]   =>* "tabat"
```

10. QuickCheck, Listeninduktion — 3 Punkte

Gegeben seien die folgenden Gleichungen für beliebige endliche Listen `xs, xs1, xs2` über einem Elementtyp `a`:

```
                xs ++ []  =  xs
reverse (xs1 ++ xs2)  =  (reverse xs2) ++ (reverse xs1)
```

(a) Testen Sie die Gültigkeit der Gleichungen für Listen ganzer Zahlen mit QuickCheck. / 1

(b) Beweisen Sie die Korrektheit der Gleichungen mittels Listeninduktion. / 2