technische universität dortmund

FLDIT

Prof. Dr. Peter Padawitz

Lehrstuhl 1
Logik in der Informatik

Jos Kusiek (*jos.kusiek@tu-dortmund.de*)
Lukasz Czajka (*lukasz.czajka@tu-dortmund.de*)

Wintersemester 2018/2019

Übungen zu Funktionaler Programmierung
Übungsblatt 9

Ausgabe: 7.12.2018, **Abgabe:** 14.12.2018 – 16:00 Uhr, **Block:** 3

Aufgabe 9.1 (10 Punkte) *Ausgabe*
Instanziieren Sie den Datentyp `BExp` für die Klasse `Show`. Beachten Sie Präzedenzen. Definieren Sie dazu die Funktion `showsPrec`. Orientieren Sie sich dabei an der entsprechenden Instanz für den Datentyp `Exp`. Benutzen Sie für die Ausgabe der Konstruktoren folgende Zeichenketten und Präzedenzen.

Konstruktor	Zeichen	Präzedenz
`True_`	`true`	—
`False_`	`false`	—
`BVar`	Name als String, z. B.: `"x"`	—
`Or`	`|`	2
`And`	`&`	3
`Not`	`!`	10 (Präfix)
`:=`	`=`	4
`:<=`	`<=`	4

Beispiele:

`bexpr1` ⇝ `"b"|!"b"`
`bexpr2` ⇝ `("x"|false)&"y"`
`bexpr3` ⇝ `"b"&"x"<="x"+10`

Aufgabe 9.2 (4 Punkte) *Bäume mit beliebigem Ausgrad*
Definieren Sie folgende Funktionen für Bäume mit beliebigem Ausgrad (`Tree`) *ohne* Faltung. Sie dürfen die Funktion `foldTree` nicht benutzen.

a) `productA :: Tree Int -> Int` – Produkt aller Zahlen in einem Baum.
Beispiel: `productA tree1` ⇝ `-1440`

b) `preorderT :: Tree a -> [a]` – Wie `preorder` (Folie 140) ohne Faltung.
Beispiel: `preorderT tree1` ⇝ `[1,2,2,3,-1,-2,4,-3,5]`

Aufgabe 9.3 (4 Punkte) *Baumfaltungen*
Definieren Sie folgende Funktionen als Baumfaltungen (`foldBin` bzw. `foldTree`).

a) `sizeBintree :: Bintree a -> Int` – Gibt die Anzahl der Knoten in einem binären Baum (`Bintree`) wieder.
Beispiel: `sizeBintree btree1` ⇝ `6`

b) `labelA :: Tree a -> Node -> a` – Wie `label` (Folie 136) mit Faltung.
Beispiel: `label tree1 [0,0,1]` ⇝ `-1`

Aufgabe 9.4 (6 Punkte) *Arithmetische Ausdrücke kompilieren*
Geben Sie die Kommandosequenz für die Auswertung `execute (foldArith codeAlg expr) ([],vars)` an. Zu jedem Kommando soll auch der Stapelinhalt (Stack) nach der Ausführung angegeben werden.
Dabei sei der Ausdruck `expr` und die Belegungsfunktion `vars` wie folgt definiert:

```
expr :: Exp String
expr = Sum [5 :* Var "x", 3 :* (Var "y" :^ 2), Con 10]

vars :: Store String
vars "x" = 7
vars "y" = 5
```

Sie können mit `fst` den finalen Stapelinhalt und damit das Ergebnis der Ausführung erhalten.
`fst (execute (exp2code expr) ([],vars))` $\rightsquigarrow$ `[120]`