Probeklausur: Funktionale Programmierung im WS 2014/15

---

1) Um es als echten Test zur Klausurvorbereitung zu nutzen, sollte man die Aufgaben in 90 Minuten bearbeiten ohne sie sich vorher angesehen zu haben.
2) Man sollte auch nur die Hilfsmittel einsetzen, die in der Klausur erlaubt sein werden, nämlich ein **einseitig, handschriftlich** gefülltes A4-Blatt mit Fakten und Formeln eigener Wahl.
3) Alle Lösungen sind kurz (stichpunktartig), aber inhaltlich ausreichend zu kommentieren!

Aufgabe 1: **Algebraische Typen** 8 **Punkte**

Gegeben ist der folgende Datentyp für (unechte) Binärbäume mit `Int`-Markierungen:
`data Baum = Nil | Node Int Baum Baum`
a) Definieren Sie eine Funktion `deepestNeg::Baum->Int` welche die maximale Tiefe eines Knotens mit negativer Markierung bestimmt, wobei das Ergebnis `-1` sein soll, wenn kein Knoten eine negative Markierung hat.
b) Definieren Sie eine Funktion `highestSingle::Baum->Int` welche die minimale Tiefe eines Knotens bestimmt, dessen Elternknoten nur ein Kind hat (das zweite ist ein Nil), wobei das Ergebnis `0` sein soll, wenn es keinen solchen Knoten gibt.
c) Definieren Sie eine Funktion `biggestJump::Baum->Int`, welche die größte Differenz aus den Markierungen eines Knotens v und eines Nachfahrens von v bestimmt, wobei das Ergebnis `0` sein soll, wenn der Baum nur aus der Wurzel besteht.
Achtung: Auch die Enkel und Urenkel gehören zu den Nachfahren!
d) Analysieren Sie die Laufzeit von `biggestJump` für Bäume mit n Knoten in der Form $\Theta(f(n))$.

Aufgabe 2: **Strukturelle Induktion** 6 **Punkte**

Gegeben sind die Funktionen `g n m = n*n+m` und `square n = n*n` sowie die bekannten Definitionen

```
map f [ ]         = [ ]                    -- (1)
map f (x:xs)      = f x :  map f xs        -- (2)
foldr f e [ ]     = e                      -- (3)
foldr f e (x:xs)  = f x (foldr f e xs)     -- (4)
sum xs            = foldr (+) 0 xs         -- (5)
```

Beweisen Sie, dass für beliebige `Int` -Listen `xs` die Identität
`sum(map square xs)=foldr g 0 xs` gilt.

Aufgabe 3: **Lazy evaluation** $3+3$ **Punkte** $+$ 3 **Zusatzpunkte**

Betrachten Sie die Funktionen `fstCommon1, fstCommon2::[Int]->[Int]->[Int]->Int` mit

```
fstCommon1 xs ys zs = head[x|x<-[x|x<-xs,y<-ys,x==y],z<-zs,x==z]
fstCommon2 xs ys zs = head[x|x<-[x|x<-xs, elem x ys],z<-zs,x==z]
elem x [ ] = False
elem x (y:ys) = (x==y) || elem x ys
```

a) Wie viele Vergleichsschritte erfordert die Auswertung von

`fstCommon1 [1..10] [2..10] [3..10]` ? (mit kurzer Begründung)

b) Analysieren Sie die (worst case) Laufzeit von `fstCommon1` für drei Listen, die jeweils die Länge n haben. Gefragt sind obere und untere Schranken, also möglicht eine Funktion der Form $\Theta(f(n))$.

c) Zusatzaufgabe: Analysieren Sie die (worst case) Laufzeit von `fstCommon2` für drei Listen, die jeweils die Länge n haben (auch hier ein $\Theta(f(n))$).

Aufgabe 4: **Typbestimmung** $3+3$ **Punkte**

Bestimmen Sie den allgemeist möglichen Typ für die folgenden Ausdrücke und begründen Sie kurz Ihre Antworten:

a)
```
f x y z = head [(s,t) | s <- x, t <- (y:x), 2*t == s, z s ]
```

b)
```
g  x y z
   |x == [y]   = z
   |otherwise  = x:z
```

Aufgabe 5: **λ-Kalkül** $2+4$ **Punkte**

a) Bestimmen sie für den λ-Ausdruck $E = (\lambda x\, y.x\, y\, z)(\lambda x.x\, y)(\lambda z.z\, x)$ die Mengen $FV(E)$ und $BV(E)$ und zeigen Sie für jedeS Auftreten von Variablen in E an, ob sie in E frei oder gebunden sind.

b) Reduzieren Sie den Ausdruck so weit wie möglich.

Aufgabe 6: **Codierungen** 4 **Punkte**

Welcher der folgenden Codes $C_1, C_2, C_3 \subseteq \{0,1,2\}^*$ ist eindeutig decodierbar und welcher nicht ? (mit kurzer Begründung!)

$C_1 = \{01, 012, 0111, 10, 210\}$
$C_2 = \{01, 02, 10, 210\}$
$C_3 = \{01, 0122, 10, 210\}$