PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

Übungen zur „Praktischen Informatik III“, WS 2006/07

Nr. 5, Abgabe: 21. November 2006 vor der Vorlesung

Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich per E-Mail an Ihren Tutor oder Ihre Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

12. `map - filter - foldr - zipWith` — 5 Punkte

Definieren Sie die folgenden Funktionen unter Verwendung vordefinierter Funktionen höherer Ordnung:

(a) `pairAndOne :: Num a => [a] -> [(a,a)]` paart jedes Element einer Liste von Zahlen mit dem um eins inkrementierten Element.
Beispiel: `pairAndOne [1,2,3]` $\Rightarrow^*$ `[(1,2), (2,3), (3,4)]`

(b) `remove :: Eq a => a -> [a] -> [a]` löscht in einer Liste alle Vorkommen eines Elementes.
Beispiel: `remove 'u' "substitute"` $\Rightarrow^*$ `"sbstitte"`

(c) `numTwins :: Eq a => [a] -> Int` zählt in einer Liste, wie oft benachbarte Elemente gleich sind. Beispiel: `numTwins "Hallo, Otto!"` $\Rightarrow^*$ `2`

(d) `multNeighbours :: Num a => [a] -> [a]` multipliziert je zwei aufeinanderfolgende Zahlen einer Liste. Beispiel: `multNeighbours [1,2,3,4]` $\Rightarrow^*$ `[2,6,12]`

(e) `ordered :: Ord a => [a] -> Bool` testet, ob eine Liste aufsteigend sortiert ist.
Beispiele: `ordered [1..4]` $\Rightarrow^*$ `True`, `ordered [4,3..1]` $\Rightarrow^*$ `False`

13. `curry` – `uncurry` — 2 Punkte

Definieren Sie unter Verwendung von `curry`, `uncurry` und der Funktionskomposition eine Funktion `sprod :: Num a => [a] -> [a] -> a` die das Skalarprodukt zweier als Zahlenlisten dargestellter Vektoren berechnet.

14. Funktionen höherer Ordnung über algebraischen Datenstrukturen — 5 Punkte

Gegeben seien die nebenstehenden Definitionen zur Modellierung natürlicher Zahlen in Haskell.

```
data Nat = Zero  |  Succ Nat
foldN :: a -> (a->a) -> Nat -> a
foldN z s Zero     = z
foldN z s (Succ n) = s (foldN z s n)
```

(a) Definieren Sie *als Instanz von* `foldN` die binären Operationen / 3

```
addN, mulN, powerN :: Nat -> Nat -> Nat
```

zur Addition, Multiplikation und Exponentiation natürlicher Zahlen.

(b) Definieren Sie *als Instanz von* `foldN` die wie folgt durch Pattern Matching definierte Vorgängerfunktion auf natürlichen Zahlen. / 2

```
data Maybe a = Nothing | Just a
subN          :: Nat -> Maybe Nat
subN Zero     =  Nothing
subN (Succ n) =  Just n
```