Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lars Noschinski, Dr. Jasmin Blanchette, Dmitriy Traytel

WS 2012/13
29.01.2013
keine Abgabe

Einführung in die Informatik 2
14. Übung

Aufgabe G14.1 QuickCheck

Gegeben sei eine Ihnen unbekannte Implementierung der Funktion

```
delete :: Eq a => a -> [a] -> [a]
```

Angeblich entfernt `delete x xs` sämtliche Vorkommen von `x` aus der Liste `xs`. Überprüfen Sie diese behauptung, indem Sie eine vollständige Sammlung von QuickCheck-Tests für `delete` definieren. Eine Sammlung von Tests ist *vollständig*, wenn im Falle eines Fehlers in der Implementierung es Eingaben für die Tests gibt, die den Fehler aufdecken (d.h. der Test schlägt fehl). Vergleichen Sie mehrere Test-Sammlungen in der Gruppe, prüfen Sie die Vollständigkeit und diskutieren Sie Vor- und Nachteile dieser Vorschläge.

Aufgabe G14.2 Endrekursive Funktionen

Wir betrachten die Funktion `concat :: [[a]] -> [a]`, die eine Liste von Listen nimmt und diese konkateniert:

```
concat [[1,2],[],[5,6],[7]] = [1,2,5,6,7]
```

1. Geben Sie eine endrekursive Implementierung von `concat` an.
2. Eine naive Implementierung der Funktion `concat :: [[a]] -> [a]` sieht wie folgt aus:

   ```
   concat' :: [[a]] -> [a]
   concat' [] = []
   concat' (xs:xss) = xs ++ concat' xss
   ```

   Vergleichen Sie das Auswertungsverhalten der beiden Implementierungen. Welche Implementierung ist in Haskell besser? (Wie sähe es mit einer strikten Auswertungsstrategie aus?)

Aufgabe G14.3 Reduktion

Werten Sie die folgenden Ausdrücke mit Haskells Auswertungsstrategie vollständig aus:

```
map (*2) (1 : threes) !! 1
(\f -> \x -> x + f 2) (\y -> y * 2) (3 + 1)
filter (/=3) threes
```

Welche Auswertungen terminieren nicht? Dabei seien die verwendeten Funktionen wie folgt definiert:

```
map _ [] = []
map f (x:xs) = f x : map f xs

filter _ [] = []
filter f (x:xs) | f x = x : filter f xs
                | otherwise = filter f xs

(x:xs) !! n | n == 0 = x
            | otherwise = xs !! (n - 1)

threes = 3 : threes
```

Hinweis: Die Hausaufgaben auf diesem Blatt sind alleine zur Wiederholung gedacht und werden nicht korrigiert. Ab Montagabend wird die Musterlösung auf der Website bereitstehen, so dass Sie gegebenenfalls in Ihrer Tutorgruppe noch Fragen stellen können.

Aufgabe H14.1 Typen

Geben Sie den allgemeinsten Typen der folgenden Ausdrücke an:

1. `map reverse`
2. `map (: [])`
3. `map ([] :)`
4. `\g f x y -> g (f x) (f y)`
5. `\xs f -> concat (map f xs)`

Aufgabe H14.2 Induktion

Beweisen Sie die Gleichung

```
tmap f (mirror t) = mirror (tmap f t)
```

per Induktion über Bäume. Benutzen Sie das aus der Vorlesung bekannte "induction template" (Folie 234). Geben Sie zusätzlich die Gleichungen, die Sie als Induktionshypothesen (IH1 und IH2) benutzen, explizit an.

Die Funktion `mirror :: Tree a -> Tree a` ist wie folgt definiert:

```
tmap f Empty = Empty                              --tmap_Empty
tmap f (Node x l r) =                             --tmap_Node
  Node (f x) (tmap f l) (tmap f r)

mirror Empty = Empty                              --mirror_Empty
mirror (Node x l r) =                             --mirror_Node
```

```
Node x (mirror r) (mirror l)
```

Verwenden Sie in jedem Schritt *nur eine* dieser Gleichungen oder die Induktionshypothesen und vergessen Sie nicht, den Namen der angewendeten Gleichung anzugeben.

Aufgabe H14.3 Filtern und Mappen

Die Funktion `filterMap :: (a -> Maybe b) -> [a] -> [b]` ist eine Variante von `map`, die auch Elemente aus einer Liste löschen kann: Bildet die übergebenen Funktion ein Element der Eingabeliste auf `Nothing` ab, so wird das Element entfernt. Wird es auf `Just` x abgebildet, wird x in die Ausgabeliste übernommen.

Beispiele:

```
filterMap (\x -> Just (x+1)) [1,2,3] ==  [2,3,4]
filterMap (\x -> Nothing) [4,5,6] == []
filterMap (\x -> lookup x [(1,"a"),(2,"b")]) [3,2,1] == ["b","a"]
```