Dr. Steffen Jost
Leah Neukirchen

Ludwig-Maximilians-Universität München
Institut für Informatik
23. Mai 2018

06. Übung zur Vorlesung

Programmierung und Modellierung

Hinweis: Aufgrund des Feiertages entfallen die Übungen am Mittwoch, Donnerstag & Freitag (30.5.–1.6.18). Das Übungsblatt erscheint ab 4.6.18 immer am Dienstag.

A6-1 *Listenverarbeitung höherer Ordnung*

a) Ersetzen Sie die List-Comprehension in der folgenden Definition durch den Einsatz der Funktionen `map` und `filter` aus der Standardbibliothek:

```
foo1 f p xs = [f x | x <- xs, x >= 0, p x]
```

b) Die Funktion `dropWhile :: (a -> Bool) -> [a] -> [a]` aus der Standardbibliothek entfernt so lange Element vom Anfang einer Liste, so lange diese das gegebene Prädikat erfüllen. Implementieren Sie diese Funktion unter den Namen `myDropWhile` selbst mit Rekursion, ohne die Verwendung von Bibliotheksfunktionen.
Beispiel: `dropWhile (<4) [1,3,4,5,3,1] == [4,5,3,1]`

c) Die Funktion `all :: (a -> Bool) -> [a] -> Bool` aus der Standardbibliothek gibt nur dann `True` zurück, wenn alle Element der Liste das übergebene Prädikat erfüllen. Implementieren Sie die Funktion unter dem Namen `myAll` selbst ohne direkte Rekursion, sondern unter Verwendung Funktionen höherer Ordnung aus der Standardbibliothek.

A6-2 *Funktionen höherer Ordnung*

a) Implementieren Sie folgende Funktionen, analog zu (`un`)-`curry` und (`un`)-`curry3`:

```
curry4   :: ((a, b, c, d) -> e) -> a -> b -> c -> d -> e
uncurry4 :: (a -> b -> c -> d -> e) -> (a, b, c, d) -> e
```

Hinweis: Die Typsignaturen lassen bei der Implementation einer totalen Funktion hier schon keine Wahl mehr zu, wenn man auf Schummeleien wie `undefined`, `error` oder endlose Rekursion verzichtet.

b) Diskutieren Sie anhand des Typs den Unterschied zwischen folgenden drei Funktionen:

i) `uncurry3 foldr`
ii) `uncurry (uncurry foldr)`
iii) `(uncurry . uncurry) foldr`

A6-3 ***Compose*** Alois Dimpfelmoser möchte eine Funktion programmieren, welche die Summe der Quadrate aller geraden Zahlen aus einer Liste berechnet. Weil Alois der pointfree-Stil so gut gefällt (komischerweise sogar besser als List-Comprehension), hat er unter Verwendung von `compose` aus Folie 6.26 folgendes dazu implementiert:

```
geradequadratsumme = compose [sum, map (^2), filter even]
```

Leider mag GHC diese Definition nicht! Helfen Sie dem armen Alois!

Wo liegen der/die Fehler? Wie lautet die richtige Definition im pointfree-Stil?
Hinweis: Die Verwendung von `compose` könnte hier der falsche Ansatz sein.

H6-1 ***I need a Dollar*** (2 Punkte; Abgabe: `H6-1.txt` oder `H6-1.pdf`)
Ein klammer Kleptomane hat alle `$` geklaut! Fügen Sie in die nachfolgenden Haskell-Ausdrücke wieder `$` ein, so dass jeder Ausdruck zu `42` auswertet!

Hinweis: Es ist ausschließlich `$` einzufügen; sonst nichts, auch keine Klammern! Die Infix-Funktion `($)` wurde auf Folie 6.29 besprochen. Wem unklar ist, wie die Aufgabe anzugehen ist, kann die ersten beiden Teilaufgaben auch zuerst durch Einfügen von runden Klammern lösen, und diese dann erst anschließend wieder durch `$` ersetzen; bei den letzten beiden Teilaufgaben geht das aber nicht mehr — warum? (Unterschied zwischen den ersten beiden Teilaufgaben und den letzten beiden Teilaufgaben mit 1–3 Sätzen erklären.)

a) `gcd 210 28 * 2 * 3`

b) `sum filter odd [2..8] ++ [6..12]`

c) `(3)(*14)`

d) `(foldr) (6) (6) [(-),(*),(-),(+)]`

H6-2 ***Falten*** (2 Punkte; Datei `H6-2.hs` als Lösung abgeben)
Definieren Sie die Funktionen `myLength :: [a] -> Int` und `myReverse :: [a] -> [a]` ohne direkte Rekursion, sondern nur unter Verwendung von `foldl`. Die Funktionen sollen so funktionieren wie `length` und `reverse` aus der Standardbibliothek.
Zur Erinnerung: `foldl :: (b -> a -> b) -> b -> [a] -> b`

H6-3 ***Monoid für MyList*** (2 Punkte; Datei `H6-3.hs` als Lösung abgeben)
Implementieren Sie die Instanzdeklaration `Monoid (MyList a)` für den Datentyp aus H4-3 und A5-1:

```
data MyList a = Leer | Element a (MyList a)     deriving Show
```

Achten Sie darauf, dass Ihre Instanzdeklaration alle geforderten Gesetze für `Monoid` erfüllt!

H6-4 ***Einige Funktor*** (2 Punkte; Datei `H6-4.hs` als Lösung abgeben)
Implementieren Sie die Instanzdeklaration `Functor Einige` für folgenden Datentypen:

```
data Einige a = Keins | Eins a | Zwei a a | Drei a a a
  deriving (Eq, Ord, Show)
```

Hinweise:

- Wir überlegen uns zuerst, welchen konkreten Typ die geforderte Funktion `fmap` dabei hat: `fmap :: (a -> b) -> (Einige a) -> (Einige b)`
- In der Instanzdeklaration ist zu beachten, dass `Functor` als Argument einen Typkonstruktor mit Kind `* -> *` erwartet. (Was heisst das hier?)
- Der Datentyp `Einige` kann als Erweiterung von `Maybe` aufgefasst werden, d.h. wir können uns an dessen Implementation in Foliensatz 7 gut orientieren.

Abgabe: Lösungen zu den Hausaufgaben können bis Samstag, den 2.6.18, mit UniWorX nur als `.zip` abgegeben werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bis zu 3 Studierende können gemeinsam als Gruppe abgeben. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ss-2018/promo/`).