

HUF zu FUN

(Häufig unbeantwortete Fragen zur Vorlesung PI3)

Berthold Hoffmann

Universität Bremen and DFKI Bremen
Cartesium 2.48, Tel. 218-64 222
hof@informatik.uni-bremen.de

In diesem Text stehen einige Fragen, die man sich zu den Folien der Veranstaltung *Funktionales Programmieren* (Praktische Informatik 3) im Winter 2010/2011 stellen könnte.

Alle TeilnehmerInnen der Veranstaltung sind eingeladen, die Fragen zu beantworten. Das sollte bei regelmäßigem Besuch der Veranstaltung nicht schwer fallen, ist aber völlig freiwillig.

Vielleicht stellen sich Ihnen auch noch weitere Fragen, die Sie dringlicher, interessanter und schwieriger finden? Dann können sie natürlich auch im Tutorium besprochen werden. Oder Sie klären dies in meiner *allgemeinen Fragestunde* zu PI3, Donnerstags 9–11 im Cartesium 2.48 (oder nach Vereinbarung per Email).

Die hier gestellten Fragen könnten auch Gegenstand der *Fachgespräche* am Ende des Semester sein.

1 Fragen zur Vorlesung am 27. 10. 2010

1. Definieren Sie die Funktionen *fac* und *repeat* als Methoden in Java.
 - Sind die Methoden referentiell transparent? Weshalb (nicht) ?
 - Welche Definition ist: kürzer? schöner? ... (Weshalb?)
2. Für welche Teilmenge von *Int* ist die Funktion *repeat* definiert?
3. Das funktionale Programmieren *unmittelbar praktisch* angewendet werden kann, sehen Sie an folgender Aufgabe: *Clara Clever* meint, sie könne jedes Aufgabenblatt in PI3 “locker” so bearbeiten, dass sie jeweils $h\%$ der zu erzielenden Punkte bekommt. Insgesamt möchte Sie $z\%$ der insgesamt zu erzielenden Punkte erreichen – danach richtet sich ja die Note. (Nur mit $z \geq 50\%$ bekäme sie einen Schein.) Clara möchte ausrechnen, wie viele der $n - 1$ in die Wertung eingehenden Aufgaben der insgesamt $n = 11$ gestellten Aufgaben sie bearbeiten muss, um $z\%$ der Punktzahl zu erreichen. Dann kann sie auch gleich eine Haskell-Funktion schreiben mit der Signatur *minAuf*: $Int \rightarrow Int \rightarrow Int$, die dann wie folgt ausgewertet:

```
minAuf 100 50  ~>  5
minAuf 100 51  ~>  6
minAuf 99  50  ~>  6
minAuf 90  50  ~>  6
minAuf 75  50  ~>  7
minAuf 50  50  ~> 10
minAuf 90  90  ~> 10
minAuf 50 100  ~> 20
minAuf 50 (-20) ~> -4
minAuf (-20) 50 ~> -24
```

Wäre dies eine echte Aufgabe in PI3, wäre Claras Hoffnung auf $h = 100\%$ zu optimistisch. Weshalb wohl?

4. Was ist *syntaktischer Zucker*? Dies steht nicht auf den Folien, Christoph hat aber zugegeben, diesen Begriff zweimal erwähnt zu haben. (Wenn es jetzt noch nicht klar ist, worum es sich dabei handelt, ist das nicht so schlimm, denn Christoph wird diesen Begriff noch öfters erwähnen.)

2 Fragen zur Vorlesung am 3. 11. 2010

1. Definieren Sie die Funktion *max* mit einer bedingten Definition. (wie *f* auf Folien 12-13)
2. *Abseits muss nicht sein!* Es vermeidet aber die in anderen Sprachen allgegenwärtigen geschweiften Klammern “{”, “}” und das Semikolon “;”. Allerdings gibt es diese Zeichen in HASKELL auch. In “*D*₁;*D*₁” trennt ein Semikolon zwei Definitionen voneinander, und ein Deklarationsblock “\{*D*₁;...;*D*_{*n*} \}” fasst Deklarationen zusammen. Damit können Sie ganze HASKELL-Programme auf eine Zeile schreiben, ohne dass etwas ins Abseits gestellt wird.
Wie geht das mit der Funktion *f* von Folie 13? ¹
3. Benennen Sie die Namen in einer von beiden Definitionen der Funktion *f* so um, dass keiner der Namen mehr einen anderen überlagert. (Folie 13)
4. Wenn man schreibt “**data** *Month* =...; **deriving** (*Eq,Ord*)”, sind die Werte des Datentypen *Days* total geordnet. Es gilt

$$Mon < Tue < Wed < Thu < Fri < Sat < Sun.^2$$

Können Sie die Funktion *weekend* einfacher definieren, wenn Sie die Ordnungsrelation (\leq):: *Date* \rightarrow *Date* \rightarrow *Bool* ausnutzen?

Ein einziger Vergleich müsste reichen! (Folie 19)

5. Welche Signatur hat der Wertkonstruktor *Date* auf der rechten Seite der Definition des gleichnamigen Datentyps auf Folie 24?
6. Welche Signatur haben die Wertkonstruktoren *C*_{*i*} auf Folie 27.?

¹ Das ist nicht als Ansporn gemeint, möglichst kurze Programme mit immer längeren Zeilen zu schreiben. Es soll erklären, wie der *ghci* ein Programm “versteht”. Wenn die Abseitsregel nicht eingehalten wird, kann es sein, dass Fehlermeldungen auf Zeichen “{”, “}” oder “;” Bezug nehmen, die im Programm gar nicht stehen.

² Was es allgemein mit “**deriving**” auf sich hat, wird in einer der nächsten Vorlesungen erklärt. Hier sei nur gesagt, dass *Eq* die Typklasse der mit den Operationen $=, \neq$ vergleichbaren Datentypen ist, und *Ord* die Typklasse der mit den Operationen $<, \leq, >, \geq$ total geordneten Datentypen. Mit **deriving** (*Eq,Ord*) werden diese Vergleichsoperationen für Werte des Typs *Date* automatisch definiert (abgeleitet).

3 Fragen zur Vorlesung am 10. 11. 2010

1. Definieren Sie die Funktion $mult:: Nat \rightarrow Nat \rightarrow Nat$ zur *Multiplikation* natürlicher Zahlen.
2. Definieren Sie die Funktion $fac:: Nat \rightarrow Nat$ zur Berechnung der *Fakultät* einer natürlichen Zahl.
Ist die Fakultäts-Funktion partiell, wie die auf Folie 12 zur Vorlesung vom 29.10?
3. Welche der drei Eigenschaften von algebraischen Datentypen auf Folie 4 garantiert dass

$$\text{Cons } a : s \neq \text{Empty} \quad (\text{für beliebige } Char \ a \text{ und } MyString \ s)$$

4. Definieren Sie eine Funktion $hd:: MyString \rightarrow Char$, die das erste Zeichen eines *MyString* zurückgibt.
5. Die Funktionen len , cat , rev auf Folie 5 haben einige Eigenschaften (*properties*), die mit Gleichungen spezifiziert werden können, z. B.:

$$\begin{aligned} len(x) &= len(\text{rev } x) && \text{für alle } x \in MyString \\ cat \ x \ (cat \ y \ z) &= cat \ (cat \ x \ y) \ z && \text{für alle } x, y, z \in MyString \end{aligned}$$

Definieren Sie mindestens drei weitere Eigenschaften dieser Funktionen.

6. Definieren Sie eine Funktion $width:: Btree \rightarrow Int$, die zurückgibt, wieviele Knoten (*BNode* ...) ein Baum enthält.
7. Definieren Sie eine Funktion $mirror:: Btree \rightarrow BTree$, die die Zweige jedes Knotens in einem Baum vertauscht.
Definieren Sie mindestens eine Eigenschaft von *mirror* und beweisen Sie sie durch strukturelle Induktion.
8. Definieren Sie eine Variante *BTree'* von Binärbäumen, in denen die Zahlen den Blättern und nicht den Knoten zugeordnet sind. Definieren Sie eine Höhenfunktion *height'* und eine Breitenfunktion *width'* für diesen Typ.

4 Fragen zur Vorlesung am 17. 11. 2010

1. Mit welchen Typen kann der Typ *List a* von Folie 5 noch instanziiert werden? (Außer mit *Bool*, *Int* oder *Char*?) Nennen Sie mindestens drei weitere Typen.
Geben Sie jeweils zwei typkorrekte Terme für jede Ihrer *List*-Instanzen an!
2. Geben Sie einen Term an, der den Typ $[[a]]$ hat.³
3. Definieren Sie den Zusammenhang zwischen den Funktionen *head* und *tail* mit einer Gleichung. (Folie 12)
4. Definieren Sie den Zusammenhang zwischen den Funktionen *init* und *last* mit einer Gleichung. (Folie 12)
5. Definieren Sie den Zusammenhang zwischen den Funktionen *take*, *drop* und *splitAt* mit einer Gleichung. (Folie 12)
6. Werten Sie den Ausdruck *perms* [1,2,3,4] aus – von Hand!
7. In der Funktion *qsort* auf Folie 20 wird die Liste drei mal durchlaufen, um die kleineren, gleich großen und größeren Elemente zu bestimmen.
 - (a) Definieren Sie eine Variante *qsort'*, die die Liste in zwei Durchläufen aufteilt.
 - (b) Definieren Sie eine Variante *qsort''*, die die Liste in einem Durchlauf aufteilt – dies geht aber nicht mit einer Listenkomprehension.

³ Ihre Antworten auf die letzten beiden Fragen können Sie gut überprüfen, indem Sie im *ghci* mit `:type t` den Typ eines Terms *t* erfragen.

5 Fragen zur Vorlesung am 24. 11. 2009

Vergessen Sie die Eigenschaften nicht!

1. Die Funktion $iter\ n\ f\ x$ hat eine Cousine $fixPoint\ f\ x$, die die Funktion f solange auf den Ausgangswert x anwendet, bis $f\ x == x$. (Folie 4)
Definieren Sie die Cousine.
2. Die *Identitätsfunktion* $id :: a \rightarrow a$ mit $id\ x = x$ ist nützlicher als man denkt. Formulieren Sie mit Gleichungen die Eigenschaft, dass id für die Funktionskomposition (\circ) *linksneutral* und *rechtsneutral* ist.
3. Geben Sie eine Gleichung an, die eine offensichtliche Eigenschaft der Funktion $flip$ definiert.
Schaffen Sie das auch *punktfrei*?
4. Definieren Sie mit map eine Funktion, die alle Zahlen einer Liste quadriert, bzw. deren Vorzeichen umkehrt.
5. Definieren Sie mit $filter$ eine Funktion, die aus einer Liste von Zahlen alle positiven Zahlen herausfiltert, bzw. aus einer Zeichenkette alle Layoutzeichen ' ', '\n' (Zeilenwechsel) und '\t' (Tabulator) herausfiltert.⁴
6. Definieren Sie mit map und $filter$ eine Funktion, die von allen Zahlen einer Liste, die ungleich null sind, den Kehrwert bildet.
7. Erkennen Sie einen Zusammenhang zwischen map und $filter$ auf der einen Seite und Listenkomprehensionen auf der anderen Seite?
Beschreiben Sie die Funktionen toL (Folie 9) und $letters$ (Folie 10) und $sieve$ (Folie 11) mit Listenkomprehensionen.
Wie könnte man folgendes Schema für die Listenkomprehension
 $[F\ x \mid x \leftarrow L, P\ x]$
definieren? (Dabei sind $L :: [a]$, $P :: a \rightarrow Bool$ und $F :: a \rightarrow b$ Ausdrücke.
8. Unter welcher Bedingung gilt
 $(map\ f) \circ (map\ g) = id$
Punktfrei definiert, wenn es geht.
9. Definieren Sie das *Produkt*, die *Konjunktion* und die *Disjunktion* der Elemente einer Liste mit einfacher Rekursion.
Definieren Sie dies auch mit $foldr$.
Für welche der Funktionen dürften Sie nach dem Satz auf Folie 21 statt dessen auch $foldl$ verwenden?
Für welche der Funktionen wäre dies günstiger?

⁴ *Tipp*: die Funktion $elem\ x\ xs$ von Folie 22 überprüft, ob ein Wert x in einer Liste xs von Werten enthalten ist.

10. Definieren Sie die Funktion *zipWith*, die auf Folie 25 erwähnt ist.
11. Definieren Sie eine Funktion $width :: Tree\ a \rightarrow Int$, die die *Breite* eines Baums t liefert, also die Anzahl der in t enthaltenen Markierungen. Finden sie mindestens zwei Eigenschaften von *width* heraus.
12. Definieren Sie eine Funktion $mapT :: (a \rightarrow b) \rightarrow Tree\ a \rightarrow Tree\ b$, die eine Funktion auf alle Markierungen eines Baums abbildet. Geben Sie mindestens zwei Eigenschaften von *mapT* an.

6 Fragen zur Vorlesung am 1. 12. 2010

1. In Haskell hat jede Variable und Funktion einen statischen Typ. Wie ist das in Java?
 - Hat jede Variable x einen statischen Typ t ?
 - Enthält die Variable x zu jedem Zeitpunkt ihrer Existenz immer einen Wert genau dieses Typs t ?
2. Schreiben Sie auf, wie folgende Typen als Werte des Haskell-Datentyps *Typ* dargestellt werden:
 - *Bool*
 - $(,,) \text{ Int Bool } (([]) \text{ Int})$ (für Naschkatzen: $(\text{Int}, \text{Bool}, [\text{Int}])$)
 - $(\rightarrow) (([]) \text{ a}) (([]) \text{ b})$ (für Naschkatzen: $[\text{a}] \rightarrow [\text{b}]$)
 - $(\text{a} \rightarrow \text{b} \rightarrow \text{b}) \rightarrow \text{b} \rightarrow [\text{a}] \rightarrow \text{b}$
3. Gilt in Haskell für den Ausdruck “**let** $x = e_1$ **in** e_2 ” die Typregel (4) oder (5)?
Finden Sie das mit dem `ghci` heraus!
4. Versuchen Sie folgende Haskell-Typen zu unifizieren und geben Sie ggf. die “allgemeinsten” Substitutionen an, die die beiden Typen gleich machen:
 - (Int, a) und $(\text{b}, [\text{b}])$
 - $\text{a} \rightarrow \text{Int}$ und $(\text{b} \rightarrow \text{b} \rightarrow \text{b}) \rightarrow \text{b}$
 - $\text{a} \rightarrow \text{b}$ und $\text{Int} \rightarrow \text{Int} \rightarrow (\text{Int}, \text{Bool})$
 - (Int, a) und $(\text{b}, [\text{b}], \text{b})$
 - $(\text{Int}, [\text{Int}])$ und $([\text{a}], \text{a})$

7 Fragen zur Vorlesung am 8. 12. 2010

1. Den Typ *Poly a* von Übungsblatt 3 sollte man besser als *abstrakten Datentyp* definieren, in Haskell also in einem Modul, das den Namen *Polynoms* tragen könnte.
 - (a) Wie müsste die Schnittstelle des Moduls aussehen?
 - (b) Reichen die auf dem Blatt angegebenen Funktionen aus, um Polynome als abstrakten Datentyp zu benutzen?
Wenn nicht: Welche Funktionen müssten noch zusätzlich exportiert werden?
 - (c) Die Invariante der Datentypimplementierung wird auf dem Übungsblatt ja benannt.
Weisen Sie nach, dass Ihre Funktionen die Invariante bewahren!
 - (d) Welche Eigenschaften der Funktionen sind für die *Benutzer* relevant?
Überlegen Sie einige Eigenschaften der Funktionen, z.B. als Gleichungen, die Zusammenhänge zwischen den Funktionen beschreiben!
2. Wie könnte der Typ *Poly a* von Übungsblatt 3 in Java als *abstrakten Datentyp* definiert werden?
Vergleichen Sie die Definitionen in Haskell und Java!
3. Definieren Sie ein Modul *SetAsOrderedList*, das Mengen mit der Schnittstelle auf Folien 16-17 als geordnete Listen implementiert!
4. Definieren Sie ein Modul *MapAsOrderedPairList*, das Abbildungen mit der Schnittstelle auf Folien 20-21 als nach der ersten Komponente geordnete Paarlsten implementiert!
 - (a) Formulieren Sie die Invariante für die Implementierung des Typs!
 - (b) Schätzen Sie den Aufwand der Operationen ab!
 - (c) Definieren Sie Eigenschaften der exportierten Funktionen!

8 Fragen zur Vorlesung am 5. 1. 2011

In der Vorlesung ging es um die Spezifikation eines abstrakten Datentyps, die aus den Signaturen seiner Funktionen und Axiomen über deren Verhalten besteht.

Betrachten Sie die abstrakten Datentypen (ADTen), die Sie in den Fragen zur letzten Vorlesung konstruieren sollten:

- Die Polynome *Poly a*
- Die Mengen *Set a*

Tun Sie für diese abstrakten Datentypen das Folgende:

1. Stellen Sie die Signaturen der abstrakten Datentypen zusammen.
2. Finden Sie Eigenschaften der Operationen.
3. Definieren Sie diese Eigenschaften als testbare Eigenschaften (wie auf den Folien 13–14).
4. Beweisen Sie die Eigenschaften.
5. Definieren Sie *properties* für *quickCheck* (wie auf den Folien 22–25).
6. Testen Sie die Eigenschaften. mit *quickCheck*.