Dr. Steffen Jost
Dr. Ulrich Schöpp

Ludwig-Maximilians-Universität München
Institut für Informatik
29. April 2019

Programmierung und Modellierung

Blatt 1

Für die Übungen wird jede Woche montags ein Übungsblatt veröffentlicht. Darauf finden Sie Präsenzaufgaben und Hausaufgaben.

Präsenzaufgaben sind mit A gekennzeichnet und werden in den Übungen der folgenden Woche bearbeitet. Diese Aufgaben sind explizit zur Diskussion innerhalb kleiner Gruppen gedacht. Wir gehen davon aus, dass während der Übung je zwei bis drei Teilnehmer gemeinsam einen Laptop mit installiertem GHC zur Verfügung haben[1] und die Präsenzaufgaben gemeinsam lösen. Rufen Sie den Tutor erst, wenn Ihre Gruppe wirklich stecken bleibt.

Hausaufgaben sind mit H gekennzeichnet und sollen selbständig bearbeitet werden. Die meisten Hausaufgaben sind mit einer Punktzahl gekennzeichnet. Durch Abgabe dieser Aufgaben können Sie einen Bonus erwerben, mit dem Sie Ihre Endnote verbessern können. Wenn Sie die Klausur bestanden haben, dann wird der Hausaufgabenbonus auf Ihre Note angerechnet. Mit dem vollen Hausaufgabenbonus werden Sie sich um etwa zwei Notenstufen verbessern können. Sie können die Hausaufgaben in Gruppen mit bis zu drei Studierenden abgeben. In dieser Gruppe muss die Lösung selbständig erarbeitet sein. Plagiate werden geahndet.

Für die Bearbeitung der Hausaufgaben ist normalerweise eine Woche Zeit. Eventuelle Fragen zu den Hausaufgaben können in den Übungen besprochen werden. Die Frist zur Abgabe Ihrer Lösungen über Uni2Work finden Sie immer am Ende des Übungsblattes.

Bitte beachten Sie bei der Abgabe von Hausaufgaben jeweils den in der Titelzeile der Aufgabe spezifizierten **Dateinamen und das Dateiformat**. Rechnen Sie mit Punktabzug, wenn eine Abgabe zu einer Programmieraufgabe nicht kompiliert, wenn der Dateiname nicht den Anforderungen entspricht, oder wenn das Dateiformat nicht stimmt. Diese Abzüge sollten leicht zu vermeiden sein.

[1] Hinweise zur Installation von GHC finden Sie auf `www.tcs.ifi.lmu.de/lehre/ss-2019/promo/`.

A1-1 ***Zahlen*** Überlegen Sie sich soweit möglich anhand der Beispiele aus der Vorlesung, welche Haskell-Ausdrücke sinnvoll sind und was ihr Wert ist. Überprüfen Sie mit GHCi.

a) `(3 + 2) / 4`

b) `(3 + 2) / 4.1`

c) `div (3 + 2) 4`

d) `div (3 + 2) 4 + 1.0`

e) `fromIntegral (div (3 + 2) 4) + 1.0`

f) `(3 + 2) `div` 4`

g) `div (3 + 2) 4.1`

h) `div (3 + 2) (4.1 - 0.1)`

i) `let x = 10000000000000000000 in x + 1 == x`

j) `let x = 10000000000000000000 in x + 1.0 == x`

k) `let x = 10000000000000000000 in x + 1 :: Int`

A1-2 ***Listen und Strings*** Überlegen Sie sich soweit möglich anhand der Beispiele aus der Vorlesung, welche Haskell-Ausdrücke sinnvoll sind und was ihr Wert ist. Überprüfen Sie mit GHCi.

a) `0 : [1, 2]`

b) `[] : [[1, 2]]`

c) `[] ++ [[1, 2]]`

d) `'a' : "bc"`

e) `'a' : 'b' : 'c'`

f) `'a' : 'b' : 'c' : []`

g) `"a" : "bc"`

h) `"a" ++ "bc"`

i) `"a" ++ [2, 3]`

A1-3 ***Funktionen und GHCi*** Speichern Sie folgenden Code in eine Datei `mystery.hs`:

```
implies :: Bool -> Bool -> Bool
implies x y = if not x then True else y

foo :: Bool -> Bool -> Bool
foo x y = implies (not x) y && implies y (not x)
```

Laden Sie die Datei `mystery.hs` in GHCi. Werten Sie nun die Funktion `foo` mehrfach mit verschiedenen Argumenten aus. Erstellen Sie daraus eine Wahrheitstafel mit allen möglichen Eingaben. *Bonusfrage:* Welchen logischen Operator implementiert `foo`?

A1-4 *Kartesisches Produkt* Berechnen Sie mit Papier und Bleistift die vollständige Menge des jeweiligen kartesischen Produktes. Überprüfen Sie gegebenfalls mit List-Comprehensions in GHCi.

Hinweis: im Gegensatz zu Listen ist die Reihenfolge der Elemente einer Menge unerheblich (und Mengen enthalten auch keine „doppelten“ Elemente).

a) $(\{1,2\} \times \{3,4\}) \times \{5,6\}$

b) $\{1,2\} \times (\{3,4\} \times \{5,6\})$

c) $\{1,2\} \times \{\,\}$

d) Sei A eine Menge mit n Elementen. Wie viele Elemente gibt es in $A \times \{27,69\}$?

A1-5 *List-Comprehensions* Folgende Funktion berechnet die Liste aller positiver Elemente einer gegebenen Liste mithilfe einer List-Comprehension.

```
positive :: [Int] -> [Int]
positive xs = [ x | x <- xs, x > 0 ]
```

Zum Beispiel ist `positive [1, 2, -3] == [1, 2]` wahr.

Schreiben Sie analog folgende Funktionen und testen Sie einige Beispiele mit GHCi.

a) `range :: Int -> Int -> [Int]` soll so definiert sein, dass `range x y` die Liste aller Zahlen zwischen `x` und `y` in aufsteigender Reihenfolge ist.

   Beispiele: `range 1 3 == [1, 2, 3]` und `range 5 1 == [1, 2, 3, 4, 5]`.

b) `split :: Int -> [Int] -> ([Int], [Int])` soll so definiert sein, dass `split x xs` zwei Listen von Zahlen zurückgibt. Die erste soll aus den Zahlen in `xs` bestehen, die kleiner als `x` sind, die andere aus den restlichen Zahlen.

   Beispiel: `split 3 [1, 5, 6, 2, 6] == ([1, 2], [5, 6, 6])`

c) `insert :: Int -> [Int] -> [Int]` soll so definiert sein, dass `insert x xs` eine Liste mit folgender Eigenschaft ist: Zuerst kommen die Zahlen aus `xs`, die kleiner als `x` sind, dann kommt `x` und dann kommen alle restlichen Zahlen aus `xs`.

   Beispiel: `insert 4 [1, 5, 6, 2, 6] == [1, 2, 4, 5, 6, 6]`

H1-1 *Typen* (0 Punkte; Keine Abgabe)
Erst im Kopf, dann mit GHCi überprüfen: Welchen Typ haben folgenden Ausdrücke?

a) `['a', 'b', 'c']`

b) `('a', 'b', 'c')`

c) `[(False, [1]), (True, [(2.0)])]`

d) `([True, True], ('z', 'o', 'o'))`

e) `(\x -> ('a' : x, False, ([x])))`

f) `[(\x y-> (x * 2, y - 1)) m n | m <- [1..5], even m, n <- [6..10]]`

Hinweis: Kontrollieren Sie Ihre Antworten anschließend selbst mit GHCi. Auch wenn wir Typinferenz in der Vorlesung noch nicht behandelt haben, sollten Sie die meisten dieser einfachen Aufgaben bereits ohne GHCi lösen können.

H1-2 ***Wahrheitstafeln*** (3 Punkte; Abgabe: `H1-2.txt` oder `H1-2.pdf`)
Gegeben sei folgende Funktion.

```
f :: Bool -> Bool -> Bool
f x y = x < y
```

Diese Funktion verwendet einen Vergleichstest auf Booleschen Werten, den wir bisher noch nicht kennengelernt haben.

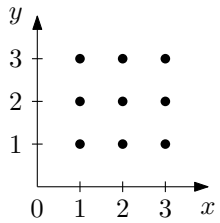
a) Berechnen Sie die Wahrheitstafel von `f` mit GHCi und geben Sie diese an.

b) Welche der folgenden Funktionen hat die gleiche Wahrheitstafel wie `f`?

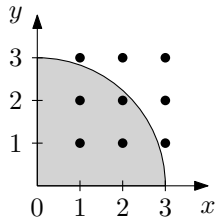
```
g1 :: Bool -> Bool -> Bool
g1 x y = not x && y

g2:: Bool -> Bool -> Bool
g2 x y = not (x || not y)

g3:: Bool -> Bool -> Bool
g3 x y = not x || not y
```

H1-3 ***Mathematische Funktionen*** (3 Punkte; Datei `H1-3.hs` als Lösung abgeben)
In dieser Aufgabe wollen wir eine Methode zur Berechnung von $\pi/4$ implementieren.

Sei k eine natürliche Zahl. Definiere R_k als das kartesische Produkt $\{1,\ldots,k\}\times\{1,\ldots,k\}$. Die Menge R_k kann man sich als Menge von Punkten (x,y) in der Ebene vorstellen. Im Fall $k=3$ enthält R_k zum Beispiel neun Punkte, die in der folgenden Abbildung dargestellt sind.



Wir zeichnen jetzt zusätzlich einen Kreis mit Radius k um den Nullpunkt ein und zählen, wie viele der Punkte aus R_k im diesem Kreis liegen. Für $k=3$ liegen vier Punkte im Kreis:



Mathematisch kann die Menge der Punkte innerhalb des Kreises so definiert werden:

$$I_k = \left\{(x,y) \mid (x,y) \in R_k,\ \sqrt{x^2+y^2} < k\right\}$$

Der Anteil der Punkte innerhalb des Kreises (genau: die Anzahl der Elemente der Menge I_k geteilt durch die Anzahl der Elemente der Menge R_k) ist nun eine Näherung für $\pi/4$. Das

liegt daran, dass der Flächeninhalt eines Kreises mit Radius 1 gerade π ist und man mit dem Verfahren den Flächeninhalt eines Viertels davon annähert. Für kleine k ist diese Näherung sehr ungenau, aber durch Erhöhen von k kann man die Näherung beliebig genau machen.

Schreiben Sie eine Datei `H1-3.hs`, welche zwei Funktionen implementiert:

```
punkteImKreis :: Double -> [(Double, Double)]
anteilImKreis :: Double -> Double
```

Beide Funktionen sollen k als Argument haben. Die Funktion `punkteImKreis` soll die Punkte in der Menge I_k in einer Liste zurückgeben. Die Reihenfolge ist nicht wichtig. Die Funktion `anteilImKreis` soll berechnen, welcher Anteil der Punkte in der Menge R_k im Kreis liegt.

Beispiel: `punkteImKreis 3 == [(1.0,1.0),(1.0,2.0),(2.0,1.0),(2.0,2.0)]` und `anteilImKreis 3 == 4/9 == 0.4444444444444444`.

Sie können Ihre Antwort selbst überprüfen, indem Sie `anteilImKreis k` für immer größer werdende `k` berechnen und mit `pi/4` vergleichen.

Hinweis: Aus der Standardbibliothek dürfen Sie zusätzlich zu den mathematischen Operationen `+`, `-`, `*`, `/`, `^`, `fromIntegral` die Funktionen `length` (Länge einer Liste) und `sqrt` (Quadratwurzel) verwenden. Schlagen Sie diese ggf. in der Dokumentation nach.

Abgabe: Lösungen zu den Hausaufgaben können bis Montag, den 6. Mai über uni2work abgegeben werden. Bitte geben Sie ein `zip`-Archiv mit den Dateien `H1-2.txt` bzw. `H1-2.pdf` und `H1-3.hs` ab.