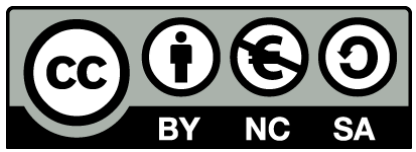
Vorlesung
Programmierung und Modellierung mit Haskell

Probeklausur 1 – Lösungen und Bewertung

François Bry

12.6.2017



Ablauf der Probeklausur

1. Bearbeitungsdauer: 1 Stunde
2. 4 Aufgaben
3. Jede Aufgabe wird 15 Minuten lang auf der Leinwand angezeigt.
4. Auf Backstage kann zu jeder Zeit jede Aufgabe angesehen werden.
5. Ausschließlich die Programmersprache Haskell soll verwendet werden.

Ablauf der Korrektur

Unmittelbar nach Ablauf der Probeklausur:

1. Die Lösung jeder Aufgabe wird gegeben.
2. Die Bewertung der Lösung wird erläutert.
3. Jeder Studierende
   - bewertet selbst seine Lösung,
   - kann seine Bewertung über Backstage mitteilen.

Aufgabe 1 – Teil 1

Wählen sie die korrekten Antworte aus oder geben Sie die korrekten Antworte an, falls keine der angegebenen Antworte korrekt ist:

1. Sei die folgende Definition: `f x = (x + 1 :: Int)`. Der Typ von `f` ist:
   - 1.1 `Int -> Int`
   - 1.2 `Int`
   - 1.3 `Integer`
   - 1.4 nichts davon, sondern ....
   - 1.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.
2. Der Typ von `(\x -> "b"++ x)` ist:
   - 2.1 `String -> String` (oder `[Char] -> [Char]`)
   - 2.2 `Char -> String` (oder `Char -> [Char]`)
   - 2.3 `Char -> Char`
   - 2.4 nichts davon, sondern ....
   - 2.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.

Aufgabe 1 – Teil 2

Wählen sie die korrekten Antworte aus oder geben Sie die korrekten Antworte an, falls keine der angegebenen Antworte korrekt ist:

3. Sei die folgende Definition: `g x = x ++ x`. Der Typ von `g` ist:
   - 3.1 `String -> String` (oder `[Char] -> [Char]`)
   - 3.2 `[a] -> [a]`
   - 3.3 `Integer -> Integer`
   - 3.4 nichts davon, sondern ....
   - 3.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.
4. Sei die folgende Definition:

```
h []       = []
h (x:xs) = x ++ x ++ (h xs)
```

   Der Typ von `h ["b", "c"]` ist:
   - 4.1 `String (oder [Char])`
   - 4.2 `[String]` (oder `[[Char]]`)
   - 4.3 `[[String]]` (oder `[[[Char]]]`)
   - 4.4 nichts davon, sondern ....
   - 4.5 Dieser Ausdruck ist kein Haskell-Ausdruck und hat folglich keinen Typ.

Lösung der Aufgabe 1

1. Sei die folgende Definition: `f x = (x + 1 :: Int)`. Der Typ von `f` ist:
   - 1.1 `Int -> Int`
2. Der Typ von `(\x -> "b"++ x)` ist:
   - 2.1 `String -> String` (oder `[Char] -> [Char]`)
3. Sei die folgende Definition: `g x = x ++ x`. Der Typ von `g` ist:
   - 3.2 `[a] -> [a]`
4. Sei die folgende Definition:

```
h []       = []
h (x : xs) = x ++ x ++ (h xs)
```

   Der Typ von `h ["b", "c"]` ist:
   - 4.2 `String (oder [Char])`

Bewertung der Aufgabe 1

1 Punkt für jede korrekte Antwort.

Bewertung der Aufgabe 1

Umfrage Selbstbewertung der Aufgabe 1

Umfrage Selbstbewertung der Aufgabe 1

Punktzahl:

- 0 oder 1 Punkt
- 2 Punkte
- 3 Punkte
- 4 Punkte

Aufgabe 2

Die Summe der ersten natürlichen Zahlen kann wie folgt definiert werden:

$$\sum_{i=0}^{i=0} i = 0$$

$$\sum_{i=0}^{i=n} i = \left(\sum_{i=0}^{i=n-1} i \right) + n \quad \texttt{für} \;\; n \geq 1$$

1. Geben Sie eine rekursive Funktion `summe` an, die dieser Definition unmittelbar entspricht. Die Funktion `summe` soll nicht terminieren, wenn sie auf negative ganze Zahlen angewandt wird.
2. Geben sie eine rekursive Funktion `summe'` an, die angewandt auf nicht-negative ganze Zahlen sich wie `summe` verhält und angewandt auf negative ganze Zahlen `0` liefert.
3. Geben Sie eine weitere rekursive Funktion `summe''` an, die endrekursiv ist, und sich wie `summe'` verhält.

Lösung der Aufgabe 2

1.
```
summe 0 = 0
summe n = n + (summe (n-1))
```

2.
```
summe' n|n <= 0     = 0
summe' n|otherwise = n + (summe' (n-1))
```

3.
```
summe'' n|n <= 0     = 0
summe'' n|otherwise = let h 0 ak = ak
                          h n ak = h(n-1)(n+ak)
                      in h n 0
```

Bewertung der Aufgabe 2

- 1 Punkte für jedes der folgenden Beweismerkmale, wenn sie korrekt und vollständig sind:
  - Funktion `summe`.
  - Funktion `summe'`
- Unter der Bedingung, dass die Funktion `summe''` endrekursiv ist 1 Punkte für jedes der folgenden Beweismerkmale
  - korrekter Einsatz eines Akkumulators
  - korrekte Verwendung einer Hilfsfunktion (die nicht lokal definiert sein muss).

Bewertung der Aufgabe 2

Umfrage Selbstbewertung der Aufgabe 2

Umfrage Selbstbewertung der Aufgabe 2

Punktzahl:

- 0 oder 1 Punkt
- 2 Punkte
- 3 Punkte
- 4 Punkte

Aufgabe 3

Seien die folgenden Definitionen:

```
f n = if n == 0 then 1 else n * (f (n-1))
doppelt x = x + x
null x = 0

hd :: [Int] -> Int
hd (x:xs) = x
```

Es ist *nicht* nötig, bei der Lösung der folgenden Aufgaben die Umgebung anzugeben. Pro Zeile soll nur einen Auswertungsschritt angegeben werden.

1. Geben Sie die Auswertung von `f 1` in applikativer Reihenfolge.
2. Geben Sie die Auswertung von `f 1` in normaler Reihenfolge.
3. Geben Sie die verzögerte Auswertung von `null (doppelt 1)`.
4. Geben Sie die verzögerte Auswertung von `hd [4..]`.

Lösung der Aufgabe 3 – Teil 1

```
f n = if n == 0 then 1 else n * (f (n-1))
```

1.
```
f 1
if 1 == 0 then 1 else 1 * (f (1-1))
if False then 1 else 1 * (f (1-1))
1 * (f (1-1))
1 * (f 0)
1 * if 0 == 0 then 1 else 0 * (f (0-1))
1 * True then 1 else 0 * (f (0-1))
1 * 1
1
```

2.
```
f 1
if 1 == 0 then 1 else 1 * (f (1-1))
if False then 1 else 1 * (f (1-1))
1 * (f (1-1))
1 * if (1-1) == 0 then 1 else (1-1) * (f (1-1)-1)
1 * if 0 == 0 then 1 else  (1-1) * (f (1-1)-1)
1 * if True then 1 else (1-1) * (f (1-1)-1)
1 * 1
1
```

Lösung der Aufgabe 3 – Teil 2

```
doppelt x = x + x
null x = 0

hd :: [Int] -> Int
hd (x:xs) = x
```

3. `null (doppelt 1)`
   `0`

4. `hd [4..]`
   `hd (4:[5..])`
   `4`

Bewertung der Aufgabe 3

Eine Antwort ist nur dann vollständig, wenn kein Auswertungsschritt fehlt.

1 Punkt für jede korrekte und vollständige Antwort.

Bewertung der Aufgabe 3

Umfrage Selbstbewertung der Aufgabe 3

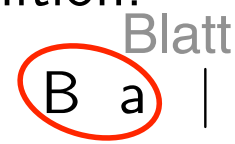
Umfrage Selbstbewertung der Aufgabe 3

Punktzahl:

- 0 oder 1 Punkt
- 2 Punkte
- 3 Punkte
- 4 Punkte

Aufgabe 4

Sei die folgende Definition:




```
data BB a = L | B a | K (BB a) a (BB a)
```

wobei `BB` für Binärbaum steht, `L` für leer und `K` für Knoten.

1. Geben Sie (in Haskell) einen ausgeglichenen Baum vom Typ `Num a => BB a` für die Werte `0`, `1`, `2`, `3`, `4`, `5`, und `6` an.
2. Geben Sie eine rekursive Suchfunktion suche vom Typ `Eq a => a -> BB a -> Bool` für Binärbäume vom Typ `BB` an.

Lösung der Aufgabe 4

```
data BB a = L | B a | K (BB a) a (BB a)
```

1. `(K (K (B 0) 1 (B 2)) 3 (K (B 4) 5 (B 6)))`

2. ```
   suche :: Eq a => a -> BB a -> Bool
   suche _ L = False
   suche x (B w)          = x == w
   suche x (K bl w br)=(suche x b)||x == w||(suche x br)
   ```
   ```

Bewertung der Aufgabe 4

1 Punkt für

- einen in Haskell nach dem Datentyp `BB` korrekt kodierten und ausgesgliechenen Baum, der genau die Werte `0`, `1`, `2`, `3`, `4`, `5`, und `6` enthält.
- jeden der drei Fällen der Definition von `suche`, der korrekt und vollständig ist.

Bewertung der Aufgabe 4

Umfrage Selbstbewertung der Aufgabe 4

Umfrage Selbstbewertung der Aufgabe 4

Punktzahl:

- 0 oder 1 Punkt
- 2 Punkte
- 3 Punkte
- 4 Punkte

Bewertung der 1. Probeklausur

Umfrage Selbstbewertung der 1. Probeklausur

Umfrage Selbstbewertung der 1. Probeklausur

Gesamte Punktzahl:

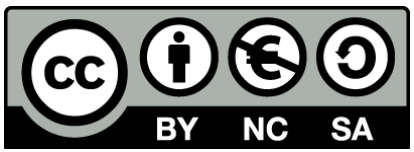
1. $[0, 4]$ Punkte
2. $[5, 8]$ Punkte
3. $[9, 12]$ Punkte
4. $[12, 16]$ Punkte

Vorlesung
Programmierung und Modellierung mit Haskell

Probeklausur 2 – Lösungen und Bewertung

François Bry

24.7.2017



Ablauf der Probeklausur

1. Bearbeitungsdauer: 1 Stunde
2. 4 Aufgaben
3. Jede Aufgabe wird 15 Minuten lang auf der Leinwand angezeigt.
4. Auf Backstage kann zu jeder Zeit jede Aufgabe angesehen werden.
5. Ausschließlich die Programmersprache Haskell soll verwendet werden.

Ablauf der Korrektur

Unmittelbar nach Ablauf der Probeklausur:

1. Die Lösung jeder Aufgabe wird gegeben.
2. Die Bewertung der Lösung wird erläutert.
3. Jeder Studierende
   - bewertet selbst seine Lösung,
   - kann seine Bewertung über Backstage mitteilen.

Aufgabe 1

Seien die folgenden Definitionen:

```
data BB a = L | K a (BB a) (BB a)
b = K 1 (K 2 (K 3 L L) (K 4 L L)) (K 5 (K 6 L L) (K 7 L L))

tief :: a -> (b -> a -> a -> a) -> BB b -> a
tief fL fK  L                                = fL
tief fL fK (K w linkerBaum rechterBaum) =
    fK w (tief fL fK linkerBaum) (tief fL fK rechterBaum)
```

wobei `BB` für Binärbaum steht, `L` für leer und `K` für Knoten. Ergänzen Sie:

```
anzahlKnoten :: BB a -> Int
anzahlKnoten baum =
    tief 0 (\w links rechts -> ............ ) baum

baumTiefe :: BB a -> Int
baumTiefe baum =
    tief 0 (\w links rechts -> ............ ) baum

istIn :: Eq a => a -> BB a -> ...........
istIn wert baum =
    tief False (\w links rechts -> ........... ) baum
```

Lösung der Aufgabe 1

```
anzahlKnoten :: BB a -> Int
anzahlKnoten baum =
    tief 0 (\w links rechts -> 1 + links + rechts) baum

baumTiefe :: BB a -> Int
baumTiefe baum =
    tief 0 (\w links rechts -> 1 + max links rechts) baum

istIn :: Eq a => a -> BB a -> Bool
istIn wert baum =
    tief False (\w l r -> wert == w || l || r) baum
```

Bewertung der Aufgabe 1

Bewertung: 1 Punkt für jede korrekte und vollständige Antwort.

Bewertung der Aufgabe 1

Umfrage Selbstbewertung der Aufgabe 1

Aufgabe 2 – Teil 1

Zwei Arten von Dokumente werden erfasst:

- Ein Artikel wird erfasst mit:
  - einem oder mehreren Autoren,
  - einem Titel.
- Ein Buch wird erfasst mit
  - null, einem oder mehreren Autoren,
  - einem Titel.

Es wird angenommen, dass Autoren und Titeln beliebige Zeichenketten sind.

1. Definieren Sie einen rekursiven Typ `Autoren` mit Konstruktoren
   - `EA` (für genau Einen Autor),
   - und `MA` (für Mehrere Autoren)

   für den Autor oder die Autoren eines Artikels.
2. Geben Sie einen Wert des von Ihnen definierten Typs `Autoren` für die folgenden Autoren:

   `"a1"` `"a2"` `"a3"`

Aufgabe 2 – Teil 2

3. Definieren Sie
   - einen Typ `Artikel` mit Konstruktor `A` für Artikeln,
   - einen Typ `Buch` mit Konstruktor `B` für Bücher,
   - eine Funktion `aTitel :: Artikel -> String`, die den Titel eines Artikels zurückgibt,
   - eine Funktion `bTitel :: Buch -> String`, die den Titel eines Buches zurückgibt.
4. Sei die folgende Typklasse `Dok` (für Dokument) gegeben:

```
class Dok where
  dokTitel :: Dok -> String
```

   Ergänzen Sie die folgenden Definitionen von `Artikel` und `Buch` als Instanzen der Typklasse `Dok`:

```
instance Dok Artikel where
    ..........

instance Dik Buch where
    ..........
```

Lösung der Aufgabe 2

1.
```
data Autoren = EA String | MA String Autoren
```

2.
```
a = MA "a1" (MA "a2" (EA "a3"))
```

3.
```
data Buch = B [String] String
data Artikel = A Autoren String

bTitel :: Buch -> String
bTitel B aut titel = titel

aTitel :: Artikel -> String
aTitel A aut titel = titel
```

4.
```
instance Dok Artikel where
    dokTitel = aTitel

instance Dok Buch where
    dokTitel = bTitel
```

Bewertung der Aufgabe 2

1 Punkt für jede korrekt und vollständige Antwort.

Bewertung der Aufgabe 2

Umfrage Selbstbewertung der Aufgabe 2

Aufgabe 3

1. Geben Sie die Definition eines neuen Typs `Zahl a`, wobei `a` eine Typvariable ist, mit einem einzigen Konstruktor `Z` der Stelligkeit 1.
2. Ergänzen Sie die folgende Funktionsdefinition:

```
plus :: ..... a => Zahl a ............
plus ..... ..... = Z (x1 + x2)
```

3. Ergänzen Sie die folgende Monoid-Definition so, dass `Zahl a` (für a ein Typ `Integeral`) ein Monoid für die Addition ist:

```
instance ........... => Monoid (Zahl a) where
  mempty = ...........
  mappend z1 z2 = ...........
```

Lösung der Aufgabe 3

1. ```
   data Zahl a = Z a
   ```

   oder

   ```
   newtype Zahl = Z a
   ```

2. ```
   plus :: Num a => Zahl a -> Zahl a -> Zahl a
   plus (Z x1) (Z x2) = Z (x1 + x2)
   ```

3. Integral a => Monoid
   oder Num a => Monoid

   ```
   instance ~~Monoid Num a =>~~ Zahl a where
     mempty = Z 0
     mappend z1 z2 = plus z1 z2
   ```

Bewertung der Aufgabe 3

1 Punkt für jede korrekte und vollständige Antwort.

Bewertung der Aufgabe 3

Umfrage Selbstbewertung der Aufgabe 3

Aufgabe 4

Sei die folgende Definition gegeben:

```
anwenden :: (Int -> Int -> Int) -> Maybe Int
                        -> Maybe Int -> Maybe Int

anwenden op sz1 sz2 = do
   z1 <- sz1
   z2 <- sz2
   return (op z1 z2)
```

1. Unter Verwendung von `anwenden` und `(+)`, definieren Sie eine zweistellige Funktion `plus` zur Addition von Werten des Typs `Maybe Int`.
2. Definieren Sie, ohne `anwenden` einzusetzen, eine zweistellige Funktion `division` mit Typ
   `Maybe Int -> Maybe Int -> Maybe Int`,
   so dass eine Division mit dem Null-Element von `Maybe Int` den passenden Wert vom Typ `Maybe Int` zurückgibt.

Lösung der Aufgabe 4

1. `plus = anwenden (+)`

2.
```
division :: Maybe Int -> Maybe Int -> Maybe Int
division sz1 sz2 = do
    z1 <- sz1
    z2 <- sz2
    if z2 == 0
        then Nothing
        else Just (div z1 z2)
```

Bewertung der Aufgabe 4

- **1. Frage:**
  1 Punkt für eine korrekte Antwort.
- **2. Frage:**
  1 Punkt für die Gesamtlogik falls passend.
  1 Punkt für das Ermitteln der Zahlen vom Typ `Int` aus den Werten vom Typ `Just Int`.
  1 Punkt falls beide zurückgegebene Werte (`Nothing` und Just (div z1 z2)) korrekt sind.

Bewertung der Aufgabe 4

Umfrage Selbstbewertung der Aufgabe 4

Bewertung der 2. Probeklausur

Umfrage Selbstbewertung der 2. Probeklausur