

# Vorlesung Programmierung und Modellierung mit Haskell

## Probeklausur 1 – Aufgaben

François Bry

12.6.2017



CC BY-NC-SA 3.0 DE

<https://creativecommons.org/licenses/by-nc-sa/3.0/de/>

# 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 Programmiersprache Haskell soll verwendet werden.

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 Antworten aus oder geben Sie die korrekten Antworten an, falls keine der angegebenen Antworten 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 Antworten aus oder geben Sie die korrekten Antworten an, falls keine der angegebenen Antworten 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.

## Aufgabe 2

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

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

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

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.

## Aufgabe 3

Seien die folgenden Definitionen:

$f\ n = \text{if } n == 0 \text{ then } 1 \text{ else } n * (f\ (n-1))$

$\text{doppelt } x = x + x$

$\text{null } x = 0$

$\text{hd} :: [\text{Int}] \rightarrow \text{Int}$

$\text{hd } (x:xs) = x$

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  $\text{null } (\text{doppelt } 1)$ .
4. Geben Sie die verzögerte Auswertung von  $\text{hd } [4..]$ .

## 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 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.



# Vorlesung Programmierung und Modellierung mit Haskell

## Probeklausur 2 – Aufgaben

François Bry

24.7.2017



CC BY-NC-SA 3.0 DE

<https://creativecommons.org/licenses/by-nc-sa/3.0/de/>

# 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 Programmiersprache 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
```

# Aufgabe 2 – Teil 1

Zwei Arten von Dokumenten 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 Dok Buch where
```

```
.....
```

## 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 `Integral`) ein Monoid für die Addition ist:

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

## 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.