Praktische Informatik 3: Funktionale Programmierung

Vorlesung 5 (15.11.2022): Übungen

Christoph Lüth

Wintersemester 2022/23

Was zum Selberdenken.

Die **konstante** Funktion ist

```
const :: α→ β→ α
const c _ = c
```

Übung 5.1: Was macht diese Funktion?

```
mystery xs = sum (map (const 1) xs)
```

Was zum Selberdenken.

Die **konstante** Funktion ist

```
const :: α→ β→ α
const c _ = c
```

Übung 5.1: Was macht diese Funktion?

```
mystery xs = sum (map (const 1) xs)
```

Lösung: Betrachten wir eine Beispiel-Auswertung:

```
sum (map (const 1) []) ⇝ sum [] ⇝ 0
sum (map (const 1) [True, False, True]) ⇝ sum [1,1,1] ⇝ 3
sum (map (const 1) "foobaz") ⇝ sum ([1,1,1,1,1,1]) ⇝ 6
```

Was zum Selberdenken.

Die **konstante** Funktion ist

```
const :: α→ β→ α
const c _ = c
```

Übung 5.1: Was macht diese Funktion?

```
mystery xs = sum (map (const 1) xs)
```

Lösung: Betrachten wir eine Beispiel-Auswertung:

```
sum (map (const 1) []) ⇝ sum [] ⇝ 0
sum (map (const 1) [True, False, True]) ⇝ sum [1,1,1] ⇝ 3
sum (map (const 1) "foobaz") ⇝ sum ([1,1,1,1,1,1]) ⇝ 6
```

Die mysteriöse Funktion berechnet die **Länge** der Liste `xs`!

Jetzt seit ihr dran...

Für das Palindrom hatten wir eine Funktion `clean` definiert:

```
clean :: String→ String
clean [] = []
clean (s:xs) | isAlphaNum s = toLower s : clean xs
             | otherwise    = clean xs
```

Übung 5.2: `Clean` refactored

Wie sieht `clean` mit `map` und `filter` (und **ohne Rekursion**) aus?

Jetzt seit ihr dran...

Für das Palindrom hatten wir eine Funktion `clean` definiert:

```
clean :: String→ String
clean [] = []
clean (s:xs) | isAlphaNum s = toLower s : clean xs
             | otherwise    = clean xs
```

Übung 5.2: `Clean` refactored

Wie sieht `clean` mit `map` und `filter` (und **ohne Rekursion**) aus?

Lösung:

```
clean' :: String→ String
clean' xs = map toLower (filter isAlphaNum xs)
```

Jetzt seit ihr dran!

Übung 5.3: `elem` selbstgemacht

Wie könnte die vordefinierte Funktion

$$\texttt{elem} \;::\; \texttt{(Eq}\;\alpha\texttt{)} \Rightarrow \alpha \rightarrow [\alpha] \rightarrow \texttt{Bool}$$

definiert sein?

Jetzt seit ihr dran!

Übung 5.3: `elem` selbstgemacht

Wie könnte die vordefinierte Funktion

```
elem :: (Eq α) ⇒ α → [α] → Bool
```

definiert sein?

Lösung: Eine Möglichkeit:

```
elem x xs = not (null (filter (λy→ x == y) xs))
```

Hierbei ist `λx. y ==x` eine **anonyme** (unbenannte) Funktion (ein Ausdruck mit Funktionstyp).