

Programmier-Paradigmen

Tutorium – Gruppe 4 & 8

Henning Dieterichs

Organisatorisches

- Henning Dieterichs, henning-propa@outlook.de
- Schreibt mich für Anregungen, Fragen und Feedback jederzeit an
- Folien: <https://github.com/hediet/propa2016>
- Es gibt ein Forum: https://i44web1.ipd.kit.edu/paradigmen_forum/
 - Ihr erreicht nicht nur einen Tutor, sondern alle ⇒ Schnellere Antworten

Übungsblätter sind freiwillig, aber...

- Abgabe immer Freitags 14:00 Uhr
- Besprechung/Korrektur am nächsten Montag
- Analoge Abgaben
 - Einwurf in die Briefkästen im UG von Geb. 50.24
 - Kopfzeile: [PP2016] Abgabe Blatt <X> von <Name> (Gruppe 4/8)
- Digitale Abgaben
 - Einwurf in die Mailbox henning-propa@outlook.de
 - Betreff: [PP2016] Abgabe Blatt <X> von <Name> (Gruppe 4/8)
- Gruppenabgaben lieber als keine Abgaben
 - Gruppenmitglieder als CC erwähnen

... Klausur

- 120 Minuten
- 120 Punkte in 10 Aufgaben
- „Kofferklausur“ – Alles aus **Papier** darf mitgenommen werden
 - Vorlesungsfolien, Lösungen zu Altklausuren
 - Nachschlagewerke
 - Wenn es jemand schafft: Computer auf Pappmachè-Basis 😊
- Zeit reicht definitiv nicht, alles nachzuschlagen!

Kurze Vorstellung / Wünsche

- Name (wer will)
- Studienfach
- Lieblingsprogrammiersprachen
- Stil des Tutoriums:
 - Interaktiv
 - Vorlesungsstil
- Inhalt:
 - Besprechung der Aufgaben
 - Wiederholung
 - Vertiefung
 - Vorgreifen
 - Sonstiges?

Fragen zu Blatt 0?

```
max3_1 x y z = if tmp > x then tmp else x  
               where tmp = if y > z then y else z
```

```
max3_1' x y z =  
    let tmp = if y > z then y else z  
    in if tmp > x then tmp else x
```

```
max3_2 x y z  
    | x >= y && x >= z = x  
    | y >= z && y >= x = y  
    | otherwise = z
```

```
max3_3 x y z = max (max x y) z
```

Kleiner Vorgriff: Typen

- $f1 :: Int$ heißt: $f1$ hat den Typ Int
- $f2 :: Bool \rightarrow Int$ heißt: $f2$ ist eine Funktion von $Bool$ nach Int .
- $f3 :: Bool \rightarrow Int \rightarrow Int == Bool \rightarrow (Int \rightarrow Int)$
 $f3$ ist eine Funktion, die ein $Bool$ entgegennimmt und eine Funktion zurückgibt, die ein Int entgegennimmt und ein Int zurückgibt.
- $f3\ True ::$

Kleiner Vorgriff: Typen

- $f1 :: Int$ heißt: $f1$ hat den Typ Int
- $f2 :: Bool \rightarrow Int$ heißt: $f2$ ist eine Funktion von $Bool$ nach Int .
- $f3 :: Bool \rightarrow Int \rightarrow Int == Bool \rightarrow (Int \rightarrow Int)$
 $f3$ ist eine Funktion, die ein $Bool$ entgegennimmt und eine Funktion zurückgibt, die ein Int entgegennimmt und ein Int zurückgibt.
- $f3\ True :: Int \rightarrow Int$
- $f3\ True\ 1 = (f3\ True)\ 1 :: Int$

Wiederholung:
Bekannte Funktionen /
Operatoren

Funktionen vs. Operatoren

- Operator, z.B. $+$
 - Aufruf: $a + b$ oder $(+) a b$ oder $(a+) b$ oder $(+b) a$
 - Ausnahme: (a , b) oder $(,) a b$
- Funktion, z.B. f
 - Aufruf: $f a b$ oder $a `f` b$ oder $(a `f`) b$ oder $(`f` b) a$

Funktionen und Operatoren: Allgemein

$(+)$, $(-)$, $(*)$, div , $(^)$	$:: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$
$(\&\&)$, $()$	$:: \text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}$
$(<)$, $(<=)$, $(==)$, $(>=)$, $(>)$	$:: \text{Int} \rightarrow \text{Int} \rightarrow \text{Bool}$
$(\$)$	$:: (a \rightarrow b) \rightarrow a \rightarrow b$
$(.)$	$:: (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow c)$
id	$:: a \rightarrow a$
flip	$:: (a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$
const	$:: a \rightarrow b \rightarrow a$

Tuple:	$(,)$	$:: a \rightarrow b \rightarrow (a, b)$
Triple:	$(,,)$	$:: a \rightarrow b \rightarrow c \rightarrow (a, b, c)$

...

Funktionen und Operatoren: Listen

Concat: (++) :: [a] -> [a] -> [a]

Cons: (:) :: a -> [a] -> [a]

Element At: (!!) :: [a] -> Int -> a

head, tail, last :: [a] -> a

null :: [a] -> Bool

take, drop :: Int -> [a] -> [a]

length :: [a] -> Int

Funktionen und Operatoren: Kombinatoren

`map` $:: (s \rightarrow t) \rightarrow [a] \rightarrow [t]$

`filter` $:: (s \rightarrow t) \rightarrow [a] \rightarrow [t]$

`sum, product` $:: [Int] \rightarrow Int$

`foldr` $:: (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b$

z.B. `foldr f ' | ' [1,2,3] == f(1, f(2, f(3, ' | ')))`

`foldl` $:: (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b$

z.B. `foldl f ' | ' [1,2,3] == f(f(f(' | ', 1), 2), 3)`

`zip` $:: [a] \rightarrow [b] \rightarrow [(a, b)]$

`zipWith` $:: (a \rightarrow b \rightarrow c) \rightarrow [a] \rightarrow [b] \rightarrow [c]$

Listen

`[1..4] = [1,2,3,4] = 1:2:3:4:[]`

`[1,3..9] = [1,3..10] = [1,3,5,7,9]`

`[1..] = [1,2,3,4,5,...]`

`[1,3..] = [1,3,5,7,...]`

`[1,1..] = [1,1,1,1,...]`

`[x | x <- [1..1000], x `mod` 5 == 1]`

`[(x,y) | x <- [1..10], y <- [1..2]]`

Achtung, so nicht: `[y | y <- [1..], y < 10]`

Akkumulatoren

- Aufgabe: Funktion, die die Länge eines Arrays ermittelt
 - Irgendwie
 - Mit Akkumulator
 - Mit foldr oder foldl
- Zum Nachschlagen
 - Foldr vs Foldl: https://wiki.haskell.org/Foldr_Foldl_Foldl'

Aufgabe zum Nachdenken

- Definiere unendliche Liste „primes“
- `isqrt :: Integer -> Integer`
 - `isqrt = floor . sqrt . fromIntegral`
- Für jede Primzahl $p > 3$ gibt es eine Primzahl p' mit $(\text{isqrt } p) < p' < p$.
 - Beweis: Ziemlich schwierig!

Haskell



<http://xkcd.com/1312/>