Tutorium_1

November 23, 2020

0.1 Organisatorisches

Tutorium 2 und 4 - **Feedback/Probleme/...:** - propa@teuber.dev - Chat - **Fragen:** ILIAS Forum - **Diese Folien/Notebooks:** https://teuber.dev/propa/2020

0.1.1 Übungsblätter

- **Kein** Übungsschein
- **Trotzdem:** Bitte unbedingt Übungsblätter machen!
  - Alle Abgaben über: https://praktomat.cs.kit.edu/pp_2020_WS/
- Besprechung der Lösung im darauffolgenden Tutorium

1 Klausur (Informatik)

- 120 Minuten
- Wann? Wir werden sehen...
- Beliebige Papier-Materialien erlaubt
  - **Vorlesungsfolien**
  - Musterlösungen
  - Bücher
  - Dokus
  - ...

2 Ein paar Worte zu diesem Tutorium...

- Zweites Propa-Tutorium
- Erstes Online-Tutorium
- **Bitte gebt Feedback!**
  - Was gefällt euch?
  - Was soll ich weiter machen?
  - Was soll ich weglassen?

2.1 Tutorien leben von Mitarbeit!

- Ja ich weiSS, das ist nervig - besonders online...
- Ihr habt mehr von der Zeit, wenn ihr in die Diskussion einsteigt
- Ich lade euch ein eure Kameras anzumachen
  Es ist verdammt seltsam in einen Computer hineinzureden

- Bitte **beteiligt euch** wenn ich Fragen stelle! Am besten per Mikrofon, alternativ auch per Chat
- Ich werde auch mit Online Umfragen experimentieren. Bitte macht mit!

3 Haskell ist anders...

- (Fast) alles Funktionen
  - Abbildung von Eingabe im **Definitionsbereich**...
  - ...auf Ausgabe im **Wertebereich**
  - Keine **Seiteneffekte**
  - Weniger "Wie?", mehr "Was?"

- Keine Variablen
- Typisiert (Typinferenz)

```
In [1]: f x = cos x / 2
        f pi

-0.5
```

4 Funktionsdefinition

```
myfunction x y z = <Ausdruck>

g x = sin x
f x y = (x / sin x) * g y

In [2]: g x = sin x
        f x y = (x / sin x) * g y

Line 1: Eta reduce
Found:
g x = sin x
Why not:
g = sin
```

5 Funktionsaufruf

- `myfunction x y`
- Linksassoziativ

```
In [3]: f 2 2 -- Rufe f mit Parametern 2 und 2 auf
        g (f 2 2) -- Rufe g mit Ergebnis von (f 2 2) als Parameter auf
```

```
2.0


0.9092974268256817
```

6 λ-Ausdrücke

Statt

```
f x = <Ausdruck>
```

kann man auch schreiben:

```
f = \x -> <Ausdruck>
```

7 Funktionen höherer Ordnung

Funktionen können auch Funktionen als Ein- oder Ausgabe haben

```
In [4]: callWithPlus5 f = \x -> f (x+5)
        invertNum x = -x

        invertNum (-3)
        (callWithPlus5 invertNum) (-3)
```

```
Line 1: Redundant lambda
Found:
callWithPlus5 f = \ x -> f (x + 5)
Why not:
callWithPlus5 f x = f (x + 5)Line 5: Redundant bracket
Found:
(callWithPlus5 invertNum) (-3)
Why not:
callWithPlus5 invertNum (-3)


3


-2
```

Eingabe der Funktion `callWithPlus5`: Funktion die `Integer` entgegennimmt
Ausgabe der Funktion `callWithPlus5`: Funktion die `Integer` entgegennimmt

8 Eingebaute Typen

- “‘haskell Int

- ```haskell
Bool
```

- “‘haskell Float

- ```haskell
Char
```

- “‘haskell [Int]

- ```haskell
(Int,Float)
```

# 9 Eingebaute Funktionen

## 9.1 Für `Int` und `Float`

- `(+)`
- `(-)`
- `(*)`
- `(^)` (zweites Argument muss Int sein)
- `(<)`
- `(<=)`
- `(>)`
- `(>=)`

## 9.2 Ansonsten

- Für `Int`: `div` und `mod`
- Für `Float`: `(/)`
- Für `Bool`: `(&&)` und `(||)`

# 10 Typnotation

- Jede Funktion in Haskell hat einen Typ
- Nicht explizit, sondern durch Typinferenz
- Nützlich bei Compiler Ausgaben
- Rechtsassoziativ

```
In [5]: h x y = x + y + 2
        :t h

h :: forall a. Num a => a -> a -> a
```
```

Hier relevant: `a -> a -> a` bzw.:`a -> (a -> a)`
Parameter 1: Typ `a`
Parameter 2: Typ `a`
Ausgabe: Typ `a`
Zusätzlich bedeutet `Num a`, dass a die *Typklasse* `Num` hat.
Details dazu später

11 Was bedeutet also?

```
In [6]: :t (||)
```

```
(||) :: Bool -> Bool -> Bool
```

Abbildung von zwei `Bool` Eingaben auf eine `Bool` Ausgabe
Umfrage

12 Wie schreiben wir...

...eine Funktion die als Eingabe nimmt: 1. Ein `Integer` 2. Ein `Bool` 3. Eine Liste von `Char` und deren Ausgabe ein `Float` ist

13 Wie schreiben wir...

...eine Funktion die als Eingabe eine Funktion vom Typ `Integer -> Bool` nimmt und eine Funktion vom Typ `Bool->Integer` ausgibt?

14 Scoping

Nervig:

```
In [7]: add5 x = x + 5
        add10 x = add5 (add5 x)
        add10 10
```

```
20
```

`add5` wird nur als Subroutine von `add10` verwendet.
Besser:

```
In [8]: add10' x = add5' (add5' x)
            where add5' z = z + 5
        add10' 10
```

```
20
```

15 Fallunterscheidung

15.1 1. If Then Else

```
In [9]: absoluteVal0 x = if x>=0 then x else -x
        absoluteVal0 (-10)
```

```
10
```

15.2 2. Pattern-Matching

```
In [10]: condInvert True a = -a
         condInvert False a = a
         condInvert True 10
         condInvert False 10
```

```
-10
```

```
10
```

15.3 3. Gates

```
In [11]: absoluteVal1 x
             | x >= 0 = x
             | otherwise = -x
         absoluteVal1 (-10)
```

```
10
```

15.4 ...and errors

```
In [12]: absoluteVal1 x
             | x > 0 = x
             | x == 0 = error "For some (unknown) reason this is supposed to be impossible for
             | otherwise = -x
         absoluteVal1 0
```

```
    For some (unknown) reason this is supposed to be impossible for 0
CallStack (from HasCallStack):
  error, called at <interactive>:3:16 in interactive:Ghci386
```

16 Rekursion

Klassisches Beispiel: Fakultät

```
In [13]: fak 0 = 1
         fak n = n * fak (n-1)

         fak 6
```

```
720
```

16.1 Endrekursion

- **Linear Rekursiv**
  In jedem Definitionszweig nur ein rekursiver Aufruf
- **Endrekursion**
  Wenn in jedem Zweig der rekursive Aufruf nicht in andere Aufrufe eingebettet

Warum Endrekursion?

```
In [1]: -- Aufgabe 1: Endrekursive Version von `fak` (2 min)
        fak' n = fakAcc n 1
            where
                fakAcc 1 a = a
                fakAcc n a = fakAcc (n-1) (a*n)

        fak' 5
```

```
120
```

```
In [1]: -- Aufgabe 2: Endrekursion für Fibbonaci Zahlen (5 min)
        fib 0 = 0
        fib 1 = 1
        fib n = fib (n-1) + fib (n-2)
        fib 30
        fib' n = fibAcc n 1 0
            where fibAcc m acc1 acc0
                    | m == 0 = acc0
                    | m == 1 = acc1
                    | otherwise = fibAcc (m-1) (acc1+acc0) acc1
        fib' 30
```

```
832040
```

```
832040
```

17 GHCI Demo

18 Listen

Eine Liste ist entweder... - ... leer `[]` - ... ein Konstrukt `(x:xs)` aus einem Listenkopf `x` und einer Restliste `xs`

```
In [16]: -- Gib alle Zahlen kleiner gleich n als Liste zurück
         mylist 0 = []
         mylist n = n : mylist (n-1)
         mylist 5
```

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

```
In [5]: -- Aufgabe 3: Schreibe eine Funktion, die alle geraden Zahlen < n zurückgibt (3min)
        -- Aufgabe 3: Schreibe eine Funktion, die alle geraden Zahlen < n zurückgibt
        evennumbers n
            | n == 0 = [0]
            | (n `mod` 2) == 0 = n : evennumbers (n-2)
            | otherwise = evennumbers (n-1)

        evennumbers'' n = evennumbers' n
            where evennumbers' n
                    | n==0 = [0]
                    | even n = n : evennumbers' (n-2)
                    | otherwise = evennumbers' (n-1)
        evennumbers 1
        evennumbers 21

        evennumbers'' 1
        evennumbers'' 21
```

```
Line 8: Eta reduce
Found:
evennumbers'' n = evennumbers' n
Why not:
evennumbers'' = evennumbers'
```

```
[0]
```

```
[20,18,16,14,12,10,8,6,4,2,0]
```

```
[0]
```

```
[20,18,16,14,12,10,8,6,4,2,0]
```

19 Eingebaute Listenfunktionen

- `(++)` Konkatenation
- `(!!)` Indexzugriff
- `head`, `last` Erstes bzw. letztes Element
- `null` ist Liste leer?
- `take` / `drop` Die ersten n Elemente nehmen/auslassen
- `length` Länge der Liste
- `reverse` Dreht die Liste um
- `elem x l` Testet, ob x in der Liste enthalten ist

20 Listen: Fun Facts

20.1 Fallunterscheidung

Beispiel Listenlänge:

```
length [] = 0
length (x:xs) = 1 + length xs
```

20.2 Strings

. . . sind eigentlich nur Listen von Chars

```
In [18]: head "Text"
         tail "Text"
```

```
'T'
```

```
"ext"
```

21 Module

- Zur Abgabe der Übungsblätter
- Dateiname: `<Name>.hs`
- Dateinhalt:

```
module <Name> where
...
```

- Name erhaltet ihr in Aufgabenstellung

22 Haskell in der Klausur

- Alles in der Prelude
- Alles auf den Folien
- Wenn aus Übungsblatt oder alter Klausur: Dann schreiben woher es kommt (sehr unwahrscheinlich, dass notwendig!)

23 Übungsblatt 0

Probleme? Fragen?

```
In [19]: max3if x y z = if x >= y
             then if x >= z then x else z
             else if y >= z then y else z

         max3guard x y z
             | x >= y && x >= z = x
             | y >= x && y >= z = y
             | otherwise = z

         max3max x y z = max x (max y z)

         max3if 3 4 5
         max3guard 3 4 5
         max3max 3 4 5
```

```
Line 1: Use guards
Found:
max3if x y z
  = if x >= y then if x >= z then x else z else
       if y >= z then y else z
Why not:
max3if x y z
  | x >= y = if x >= z then x else z
  | y >= z = y
  | otherwise = z


5


5


5
```

24 Weitere Übungen...

```
In [6]: -- Aufgabe 4 (10 min) baut die Listenfunktionen nach
        concat' [] l2 = l2
        concat' (x:xs) l2 = x:concat' xs l2
        indexAccess (x:xs) 0 = x
        indexAccess (x:xs) n = indexAccess xs (n-1)
        head' (x:xs) = x
```

```
        last' (x:[]) = x
        last' (x:xs) = last' xs
        null' [] = True
        null' xs = False
        take' 0 l = []
        take' n (x:xs) = x:take' (n-1) xs
        drop' n [] = []
        drop' 0 l = l
        drop' n (x:xs) = drop' (n-1) xs
        reverse' (x:[]) = [x]
        reverse' (x:xs) = reverse' xs ++ [x]
        elem' x [] = False
        elem' x (y:xs) = if x==y then True else elem' x xs

        mytestlist = [0,1,2,3,4,5,6,7,8,9]
        concat' [-2,-1] mytestlist
        indexAccess mytestlist 4
        head' mytestlist
        last' mytestlist
        null' mytestlist
        null' []
        take' 3 mytestlist
        drop' 3 mytestlist
        reverse' mytestlist
        elem' 3 mytestlist
        elem' 10 mytestlist
```

```
Line 2: Use foldr
Found:
concat' [] l2 = l2
concat' (x : xs) l2 = x : concat' xs l2
Why not:
concat' xs l2 = foldr (:) l2 xsLine 7: Use list literal pattern
Found:
(x : [])
Why not:
[x]Line 16: Use list literal pattern
Found:
(x : [])
Why not:
[x]Line 19: Redundant if
Found:
if x == y then True else elem' x xs
Why not:
(x == y) || elem' x xs


[-2,-1,0,1,2,3,4,5,6,7,8,9]
```

```
4
```

```
0
```

```
9
```

```
False
```

```
True
```

```
[0,1,2]
```

```
[3,4,5,6,7,8,9]
```

```
[9,8,7,6,5,4,3,2,1,0]
```

```
True
```

```
False
```

25 Feedback

In [ ]: