

Funktionale Programmierung am Beispiel Haskell (Teil 2)

Jan Schöppach

Hochschule Coburg - Fakultät Elektrotechnik und Informatik

23. Juni 2009



1 Problemstellung

2 Bedingungen

- Bewachte Gleichungen

3 Funktionen

- Die erste Funktion
- Funktionen mit mehreren Argumenten
 - Currying
 - curry/uncurry
- Rekursive Funktionen
- Polymorphismus
- Überladen von Funktionen
 - Typklassen
- Funktionen höherer Ordnung

4 Module

- Moduldefinierung
- Import

5 Ein-/Ausgabe

- getChar & putChar

6 Fazit

7 Quellen



- Einführung in die funktionale Programmierung in Haskell



- Einführung in die funktionale Programmierung in Haskell
- Hauptaugenmerk auf Funktionen



- Einführung in die funktionale Programmierung in Haskell
- Hauptaugenmerk auf Funktionen
- Keine Vertiefung funktionaler Programmierung da dies in Teil 1 dieser Arbeit schon erläutert



Vorkenntnisse

- Grundlegende Programmiersprachenkenntnisse
(am besten Java)



Vorkenntnisse

- Grundlegende Programmiersprachenkenntnisse (am besten Java)
- Seminararbeit: „Funktionale Programmierung am Beispiel Haskell (Teil 1)“ von Dominik Wachter



1 Problemstellung

2 Bedingungen

- Bewachte Gleichungen

3 Funktionen

- Die erste Funktion
- Funktionen mit mehreren Argumenten
 - Currying
 - curry/uncurry
- Rekursive Funktionen
- Polymorphismus
- Überladen von Funktionen
 - Typklassen
- Funktionen höherer Ordnung



If? Then - Else

Beispiel

```
1 if n == 1
2 then 1
3 else 0
```

- BASIC-ähnliche Syntax



If? Then - Else

Beispiel

```
1 if n == 1
2 then 1
3 else 0
```

- BASIC-ähnliche Syntax
- else-Block darf nicht weggelassen werden!



Bewachte Gleichungen

- Für Bedingungen: Bewachte Gleichungen



Bewachte Gleichungen

- Für Bedingungen: Bewachte Gleichungen
- Ähnlich case-Anweisung in Java



Bewachte Gleichungen

- Für Bedingungen: Bewachte Gleichungen
- Ähnlich case-Anweisung in Java

Beispiel

```
1 | n == 1 = 1
2 | n == 10 = 2
3 | n == 100 = 3
4 | otherwise = 0
```



1 Problemstellung

2 Bedingungen

- Bewachte Gleichungen

3 Funktionen

- Die erste Funktion
- Funktionen mit mehreren Argumenten
 - Currying
 - curry/uncurry
- Rekursive Funktionen
- Polymorphismus
- Überladen von Funktionen
 - Typklassen
- Funktionen höherer Ordnung



■ Funktionen als Basis funktionaler Programmierung



- Funktionen als Basis funktionaler Programmierung
- Für gleiche Argumente immer gleiches Ergebnis



Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```



Die erste Funktion

Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```

- Erste Zeile: Funktionssignatur



Die erste Funktion

Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```

- Erste Zeile: Funktionssignatur
- Input: Integer



Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```

- Erste Zeile: Funktionssignatur
- Input: Integer
- Output: Integer



Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```

- Erste Zeile: Funktionssignatur
- Input: Integer
- Output: Integer
- Kann auch weggelassen werden

Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```

- Erste Zeile: Funktionssignatur
- Input: Integer
- Output: Integer
- Kann auch weggelassen werden
- Zweite Zeile: Funktionsdefinition



Zum Vergleich: Die selbe Funktion in Java

Beispiel

```
1 toByte :: Int → Int
2 toByte mb = mb * 1024 * 1024
```

Beispiel

```
1 public int toByte (int mb)
2 {
3     return mb * 1024 * 1024;
4 }
```



■ Angabe der Argumente als Tupel

Beispiel

```
1 difference :: (Int, Int) → Int
2 difference (n, x) = x - n
```



Funktionen mit mehreren Argumenten

■ Angabe der Argumente als Tupel

Beispiel

```
1 difference :: (Int, Int) → Int
2 difference (n, x) = x - n
```

■ Aufruf der Funktion wie folgt:

Beispiel

```
difference (10, 20)
```



Funktionen mit mehreren Argumenten

- Mehrargumentige Funktionen auch als curried Funktionen möglich



Funktionen mit mehreren Argumenten

- Mehrargumentige Funktionen auch als curried Funktionen möglich
- Übergabe der Argumente nicht als Tupel sondern in Reihe



Funktionen mit mehreren Argumenten

- Mehrargumentige Funktionen auch als curried Funktionen möglich
- Übergabe der Argumente nicht als Tupel sondern in Reihe
- Curried-Funktion gibt wieder eine Funktion zurück



Funktionen mit mehreren Argumenten

- Mehrargumentige Funktionen auch als curried Funktionen möglich
- Übergabe der Argumente nicht als Tupel sondern in Reihe
- Curried-Funktion gibt wieder eine Funktion zurück

Beispiel

```
1 difference2 :: Int → Int → Int
2 difference2 n x = x - n
```



- Aufruf der Funktion analog der standard Funktionen



Funktionen mit mehreren Argumenten

- Aufruf der Funktion analog der standard Funktionen

Beispiel

```
difference2 10 20
```

- oder

Beispiel

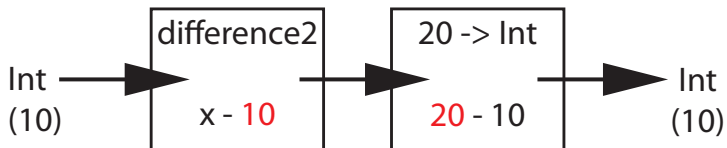
```
10 'difference2' 20
```



Funktionen mit mehreren Argumenten

Beispiel

```
(difference2 10) 20
```





Funktionen mit mehreren Argumenten

- Benutzung einer uncurried Funktion als curried und umgekehrt mit „**curry**“ und „**uncurry**“



Funktionen mit mehreren Argumenten

- Benutzung einer uncurried Funktion als curried und umgekehrt mit „**curry**“ und „**uncurry**“

Beispiel

```
curry difference 10 20  
uncurry difference2 (10,20)
```



- Da es keine Nebenwirkungen und Schleifen gibt, Rekursion wichtiger Weg um diese zu Realisieren



Rekursive Funktionen

- Da es keine Nebenwirkungen und Schleifen gibt, Rekursion wichtiger Weg um diese zu Realisieren
- Funktion ruft sich selbst auf



Die Fibonacci-Folge

Zitat: [Beck04, S.4]

„Die Folge 1, 1, 2, 3, 5, 8, 13, 21, 34, ... heißt die Fibonacci-Folge. Dabei erhält man das nächste Glied der Folge, indem man die zwei davor stehenden Zahlen addiert.“



Die Fibonacci-Folge

Zitat: [Beck04, S.4]

„Die Folge 1, 1, 2, 3, 5, 8, 13, 21, 34, ... heißt die Fibonacci-Folge. Dabei erhält man das nächste Glied der Folge, indem man die zwei davor stehenden Zahlen addiert.“

Mathematische Form

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$



Die Fibonacci-Folge

Zitat: [Beck04, S.4]

„Die Folge 1, 1, 2, 3, 5, 8, 13, 21, 34, ... heißt die Fibonacci-Folge. Dabei erhält man das nächste Glied der Folge, indem man die zwei davor stehenden Zahlen addiert.“

Mathematische Form

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

- Lässt sich leicht in eine Haskell Funktion konvertieren



Rekursive Funktionen

Mathematische Form

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

Beispiel

```
1 fibonacci :: Int -> Int
2 fibonacci 0 = 0
3 fibonacci 1 = 1
4 fibonacci n = fibonacci(n-1) + fibonacci(n-2)
```



Beispiel

```
1 fibonacci :: Int → Int
2 fibonacci 0 = 0
3 fibonacci 1 = 1
4 fibonacci n = fibonacci(n-1) + fibonacci(n-2)
```

Beispiel

```
1 fibonacci :: Int → Int
2 fibonacci n = if n ≤ 1 then n
3             else fibonacci(n-1)
4             + fibonacci(n-2)
```



- Funktionen für verschiedene Typen gültig



Polymorphismus

- Funktionen für verschiedene Typen gültig
- Trifft auf die meisten standard Funktionen zu, zum Beispiel:



Polymorphismus

- Funktionen für verschiedene Typen gültig
- Trifft auf die meisten standard Funktionen zu, zum Beispiel:

Beispiel: [Hut07, S.23]

```
fst    :: (a , b) → a
head   :: [a] → a
take   :: Int → [a] → [a]
zip    :: [a] → [b] → [(a , b)]
id     :: a → a
```



Polymorphismus

Beispiel: [Hut07, S.23]

```
> length [1, 3, 5, 7]
```

```
4
```

```
> length ["Yes", "No"]
```

```
2
```

```
> length [isDigit, isLower, isUpper]
```

```
3
```

Beispiel

```
length :: [a] → Int
```



Beispiel

```
length :: [a] → Int
```



Beispiel

```
length :: [a] → Int
```

- Variablenname beliebig wählbar

Beispiel

```
length :: [a] → Int
```

- Variablenname beliebig wählbar
- aber: Kleinbuchstabe an erster Stelle



Überladen von Funktionen

- Funktion die mit verschiedenen Typen klar kommt



Überladen von Funktionen

- Funktion die mit verschiedenen Typen klar kommt
- Zum Beispiel „+“ Operator:



Überladen von Funktionen

- Funktion die mit verschiedenen Typen klar kommt
- Zum Beispiel „+“ Operator:

Beispiel

$$(+) :: \text{Num } a \Rightarrow a \rightarrow a \rightarrow a$$



Überladen von Funktionen

- Funktion die mit verschiedenen Typen klar kommt
- Zum Beispiel „+“ Operator:

Beispiel

```
(+) :: Num a => a -> a -> a
```

- Polymorpher Typ



Überladen von Funktionen

- Funktion die mit verschiedenen Typen klar kommt
- Zum Beispiel „+“ Operator:

Beispiel

```
(+) :: Num a => a -> a -> a
```

- Polymorpher Typ
- aber: mit Einschränkung auf numerische Typen



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:

Tabelle: [Hut07, S.23-28]

Klasse	Erklärung	Beispiel
Num	Numerische Zahlen	1, 2, 1.0, 1.5
Eq	Vergleichbare Typen	Bool, String, Int, Float
Read	Einlesbare Typen	Alles von String
Show	Zeigbare Typen	Alles zu String
Fractional	Brüche	$\frac{7.0}{2.0}$, $\frac{2}{3}$



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:

Tabelle: [Hut07, S.23-28]

Klasse	Erklärung	Beispiel
Num	Numerische Zahlen	1, 2, 1.0, 1.5
Eq	Vergleichbare Typen	Bool, String, Int, Float
Read	Einlesbare Typen	Alles von String
Show	Zeigbare Typen	Alles zu String
Fractional	Brüche	$\frac{7.0}{2.0}$, $\frac{2}{3}$



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:

Tabelle: [Hut07, S.23-28]

Klasse	Erklärung	Beispiel
Num	Numerische Zahlen	1, 2, 1.0, 1.5
Eq	Vergleichbare Typen	Bool, String, Int, Float
Read	Einlesbare Typen	Alles von String
Show	Zeigbare Typen	Alles zu String
Fractional	Brüche	$\frac{7.0}{2.0}$, $\frac{2}{3}$



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:

Tabelle: [Hut07, S.23-28]

Klasse	Erklärung	Beispiel
Num	Numerische Zahlen	1, 2, 1.0, 1.5
Eq	Vergleichbare Typen	Bool, String, Int, Float
Read	Einlesbare Typen	Alles von String
Show	Zeigbare Typen	Alles zu String
Fractional	Brüche	$\frac{7.0}{2.0}$, $\frac{2}{3}$



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:

Tabelle: [Hut07, S.23-28]

Klasse	Erklärung	Beispiel
Num	Numerische Zahlen	1, 2, 1.0, 1.5
Eq	Vergleichbare Typen	Bool, String, Int, Float
Read	Einlesbare Typen	Alles von String
Show	Zeigbare Typen	Alles zu String
Fractional	Brüche	$\frac{7.0}{2.0}$, $\frac{2}{3}$



Überladen von Funktionen

- Neben „Num“ noch weitere Typklassen möglich
- Hier eine Auswahl:

Tabelle: [Hut07, S.23-28]

Klasse	Erklärung	Beispiel
Num	Numerische Zahlen	1, 2, 1.0, 1.5
Eq	Vergleichbare Typen	Bool, String, Int, Float
Read	Einlesbare Typen	Alles von String
Show	Zeigbare Typen	Alles zu String
Fractional	Brüche	$\frac{7.0}{2.0}$, $\frac{2}{3}$



Überladen von Funktionen

- Mit Überladung difference2-Funktion allgemeiner definiert:



Überladen von Funktionen

- Mit Überladung difference2-Funktion allgemeiner definiert:

Beispiel

```
1 difference2 :: Num a => a -> a -> a
2 difference2 n x = x - n
```



- Funktionen, die Funktionen zurückgeben



Funktionen höherer Ordnung

- Funktionen, die Funktionen zurückgeben
- Funktionen, die Funktionen als Argument annehmen



Funktionen höherer Ordnung

- Funktionen, die Funktionen zurückgeben
- Funktionen, die Funktionen als Argument annehmen

Beispiel: [Hut07, S.61]

```
1 twice :: (a -> a) -> a -> a
2 twice f x = f (f x)
```



Funktionen höherer Ordnung

- Funktionen, die Funktionen zurückgeben
- Funktionen, die Funktionen als Argument annehmen

Beispiel: [Hut07, S.61]

```
1 twice :: (a -> a) -> a -> a
2 twice f x = f (f x)
```

Beispiel

```
> twice (*2) 3
12
```

4 Module

- Moduldefinierung
- Import

5 Ein-/Ausgabe

- getChar & putChar

6 Fazit

7 Quellen



Modul

- Sammlung von Klassen, Funktionen und Typen

Modul

- Sammlung von Klassen, Funktionen und Typen
- Festgelegtes Interface

Modul

- Sammlung von Klassen, Funktionen und Typen
- Festgelegtes Interface
- Festlegbar auf welche Elemente Zugriff gewährt wird



Beispiel

```
module Modulname where
```



Beispiel

```
module Modulname where
```

- Modulkopf am Anfang der Datei



Beispiel

```
module Modulname where
```

- Modulkopf am Anfang der Datei
- Darauffolgend Klassen, Funktionen und Typen des Moduls



Beispiel

```
module Modulname where
```

- Modulkopf am Anfang der Datei
- Darauffolgend Klassen, Funktionen und Typen des Moduls
- Dateiname der *.hs-Datei gleich dem Modulnamen



Beispiel

```
1 module Modulname(  
2     funktion1 ,  
3     funktion2  
4 ) where
```



Beispiel

```
1 module Modulname(  
2     funktion1 ,  
3     funktion2  
4 ) where
```

- Bestimmte Elemente in Klammern angeben



Beispiel

```
1 module Modulname(  
2     funktion1 ,  
3     funktion2  
4 ) where
```

- Bestimmte Elemente in Klammern angegeben
- Auf alle Anderen von ausserhalb kein Zugriff

■ Importierung mit

Beispiel

```
import Modulname
```

■ Importierung mit

Beispiel

```
import Modulname
```

■ Teilweise Importierung mit

Beispiel

```
import Modulname (funktion1)
```



- Teilweise Importierung auch mit

Beispiel

```
import Modulname hiding (funktion1)
```



- Teilweise Importierung auch mit

Beispiel

```
import Modulname hiding (funktion1)
```

- Importiert alle Elemente ausser „funktion1“



4 Module

- Moduldefinierung
- Import

5 Ein-/Ausgabe

- getChar & putChar

6 Fazit

7 Quellen



- getChar zum Einlesen



getChar & putChar

- getChar zum Einlesen
- putChar zum Ausgeben



Beispiel

```
1 input :: IO ()
2 input = do c ← getChar
3         putChar '\n'
4         putChar c
```



Beispiel

```
1 input :: IO ()
2 input = do c ← getChar
3           putChar '\n'
4           putChar c
```

- „do“ um Befehle der Reihe nach Auszuführen



Beispiel

```
1 input :: IO ()
2 input = do c ← getChar
3           putChar '\n'
4           putChar c
```

- „do“ um Befehle der Reihe nach auszuführen
- ← um Befehle an Variablen zu binden



Beispiel

```
> input
a — Das ist die Eingabe
a — Das die Ausgabe
```

Fazit



Hutton, Graham: Programming in Haskell. 1. Aufl., Cambridge University Press, New York, 2007
(ISBN-13: 978-0-511-29615-4)



Hudak, Paul; Peterson, John; Fasel Joseph: A Gentle Introduction to Haskell 98, 1999
www.haskell.org/tutorial/haskell-98-tutorial.pdf



Wachter, Dominik: Seminararbeit: Funktionale Programmierung am Beispiel Haskell (Teil 1), 2009



Becker, Johannes: Fibonacci und der goldene Schnitt, 2004
www.uni-giessen.de/~g013/goldfibo/goldfibo.pdf



Vielen Dank für die
Aufmerksamkeit!