Real World Haskell

Tobias Höppner

SoSe 2013

Contents

1 VL I **2**

1.1 Motivation . . . 2
1.2 Was passiert hier?! - der kleine Webserver . . . 2
1.2.1 der kleine Webserver . . . 2
1.2.2 Einbinden von Modulen . . . 3
1.2.3 Do-Notation . . . 3
1.2.4 $-Operator . . . 3
1.2.5 !!-Operator . . . 3
1.3 der größere Webserver . . . 4
1.4 builds . . . 4
1.4.1 mit ghc . . . 4
1.4.2 mit cabal . . . 4
1.5 .(Punkt)-Operator . . . 4
1.6 Generics in Haskell . . . 4
1.7 Stdlib - System.IO . . . 4
1.8 Stdlib - System.Environment . . . 4
1.9 Kommentare und Haddock . . . 4
1.10 Keywords . . . 5

2 VL II **6**

2.1 Neuer Tag, neues Haskell . . . 6

Chapter 1

VL I

1.1 Motivation

Warum eigentlich Haskell?
Haskell Compiler ist mächtig. Weil die Semantik und Typsystem wilde Sachen erlaubt. Wilde Sachen ermöglichen korrekte Software und sind meist sogar effizienter.

1.2 Was passiert hier?! - der kleine Webserver

1.2.1 der kleine Webserver

```
1  import System.Environment (getArgs)
2  import System.IO (hFlush, hClose)
3  import Control.Monad (forever)
4  import Text.Printf (hPrintf, printf)
5  import Network (listenOn, accept, Socket, PortID (..))
6  import Control.Exception (handle, finally, SomeException)
7  import Control.Concurrent (forkIO)
8
9  main = getArgs >>= return . read . (!! 0)
10                >>= listenOn . PortNumber . fromIntegral
11                >>= forever . serve
12
13 serve socket = handle (\e -> print (e :: SomeException)) $ do
14   (sock, host, _) <- accept socket
15
16   forkIO $ flip finally (hClose sock) $ do
17     text <- readFile "index.html"
18
19     hPrintf sock "HTTP/1.1 200 OK\r\nContent-Length: %d\r\n\r\n%s"
20                  (length text) text >> hFlush sock
21
22   printf "Anfrage von %s beantwortet\n" host
```

Was nicht behandelt wurde:

- Fehlerfälle, Exceptions Haskell unterstütz Exceptions
- Effizienz

1.2.2 Einbinden von Modulen

import am Anfang der Datei

- System.IO
- Control.Monad (forever)
- Text.Printf
- Network
- Control.Exception
- Control.Concurrent

1.2.3 Do-Notation

```
main = do
  putStrLn "hallo user!!"
  putStrLn "xxxx"
  main = p "x" >> p "x"
```

ist das gleiche wie

```
main :: IO()
main = do
  args <- getArgs
  read ((!!0) args)
  let x = read ((!! 1) args)
```

Typen

listenOn: $\_ \leftarrow IO \_$

1.2.4 $-Operator

```
f a b
```

a ist eine Fkt. g x k
b ist eine Fkt. k fv

```
f $ g x k $ k f v
```

1.2.5 !!-Operator

Gibt das angegebene Element aus der Liste zurück.

```
(!!) :: [a] -> Int -> a
let xs = []
ys = [1,2,4]
zs = [1..1378]

zs !! 0
```

1.3 der größere Webserver

1.4 builds

1.4.1 mit ghc

```
ghc x.hs
```

Wird unübersichtlich für mehrere Dateien / Module.

1.4.2 mit cabal

```
cabal configure
cabal build
cabal install
```

Projekte werden als **.cabal** gespeichert, sind eleganter und man kann schneller testen.

1.5 .(Punkt)-Operator

```
(.) :: (b -> c) -> (a -> b) -> (a -> c)
f . g
```

entspricht

```
(\x -> f ( g x))
```

1.6 Generics in Haskell

```
List e
m k v
```

1.7 Stdlib - System.IO

- Textinput / Textoutput
  - Print
  - getLine
  - getChar

1.8 Stdlib - System.Environment

- getArgs

1.9 Kommentare und Haddock

```
-- einfacher Kommentar
{- mehrzeiliger Kommentar -}
{- mehrzeiliger Kommentar
  {- verschachtelter Kommentar -}
```

```
-}
-- | haddoc kommentar
```

1.10 Keywords

Programming Guidelines sind brauchbar

```
main = do
  args <- getArgs
  case args of
    [] -> ...
    ["-x"] -> ...
    ["-x",b] -> ...
```

Chapter 2

VL II

2.1 Neuer Tag, neues Haskell