

Fortgeschrittene funktionale Programmierung in Haskell

Universität Bielefeld, Sommersemester 2015

Jonas Betzendahl & Stefan Dresselhaus

Überblick für Heute:

Organisatorisches

Tools & Ressourcen

Thinking in Types

Functionally solving problems

Organisatorisches

Organisatorisches: Veranstaltungen

Es gibt Vorlesungen (Freitags, 14-16 Uhr in V2-205)
und Übungen (Montags, 12-14 & 18-20 Uhr in V2-221)

Vorlesungen aufgeteilt in

Teilnahme ist nicht verpflichtend, aber wahrscheinlich von Vorteil.

Organisatorisches (2): Input / Output

Für das Modul gibt es 5 (echte) Leistungspunkte

Als Leistung müsst ihr ein kleines Programmierprojekt abschließen. Details dazu in den Übungen.

Bürokratische Hürden \Rightarrow Leistungspunkte nur für *individuelle* Ergänzung.

Organisatorisches (3): Personenkult

Wir, das sind Jonas Betzendahl und Stefan Dresselhaus.

Mailadressen: {jbetzend,sdressel}@techfak...

Wir sind zwei Masterstudenten an der TechFak, die ihren Wunsch nach mehr und besserer Lehre zu funktionaler Programmierung und Haskell hier in Bielefeld jetzt selbst in die Hand nehmen.

Formal verantwortlich ist Dr. Alexander Sczyrba (asczyrba@techfak...).

Er ist eure Anlaufstelle für Fragen im Kontext der Fakultät und Beschwerden zu uns.

Organisatorisches (4): Material von uns für euch

Alle Übungsblätter, Foliensätze, Beispiele, Vorlagen und sonstige Unterlagen findet ihr entweder im ekVV oder auf der Website dieser Veranstaltung. Die URL ist:

`www.dieseurlmussnocherersetztwerden.de`

Audio/Video-Mitschnitte findet ihr ebenfalls im Web und zwar auf folgender Seite:

`www.irgendwiesowaswieunirekorder.de`

Login: Login, Passwort: 123456

Tools und Ressourcen

T & R (1): Haskell / GHC

Um auf euren eigenen Rechnern Haskell sinnvoll benutzen zu können benötigt ihr den Glasgow Haskell Compiler (GHC).

Der GHC und viele nützliche Dinge mehr sind alle im Rundum-Glücklich-Paket genannt *Haskell Platform* enthalten. Die gibt es für Linux, Mac, BSD und Windows unter folgender URL:

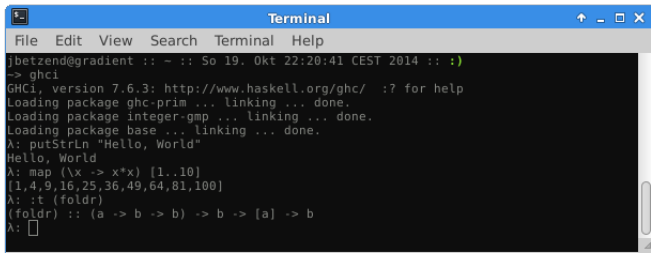
<https://www.haskell.org/platform/>

Wichtig:

Der Interpreter Hugs wird von uns explizit nicht unterstützt.

T & R (2): GHCi

Die mitgelieferte interaktive Umgebung des GHC heißt GHCi.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal content shows the user running the command `ghci` in a shell. The output indicates the version of GHCi (7.6.3) and the linking status of several packages (ghc-prim, integer-gmp, base). The user then enters several Haskell expressions: `putStrLn "Hello, World"`, `map (\x -> x*x) [1..10]`, `t (foldr)`, and `(foldr) :: (a -> b -> b) -> b -> [a] -> b`. The prompt `λ:` is visible at the end of each line.

```
jbetzend@gradient :: ~ :: So 19. Okt 22:20:41 CEST 2014 :: :)
-> ghci
GHCi, version 7.6.3: http://www.haskell.org/ghc/  ? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
λ: putStrLn "Hello, World"
Hello, World
λ: map (\x -> x*x) [1..10]
[1,4,9,16,25,36,49,64,81,100]
λ: :t (foldr)
(foldr) :: (a -> b -> b) -> b -> [a] -> b
λ: 
```

Der GHCi stellt ein REPL (Read - Evaluate - Print - Loop) bereit, die beim Entwickeln *sehr* nützlich sein kann (ähnlich zu Hugs). Mehr dazu in den Übungen.

T & R (3): Hackage

Dieser Tage gibt es eine Vielzahl an nützlichen und mächtigen Bibliotheken für Haskell. Zu Hause sind die meisten davon auf *Hackage*:

`https://hackage.haskell.org/`

Auf Hackage findet ihr übersichtliche Zusammenfassungen der Bibliotheken, detaillierte Auflistungen der exportierten Funktionen und Datentypen und direkte Links zu den jeweiligen Implementationen (!).

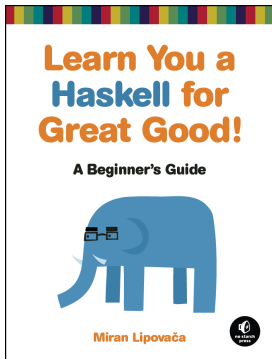
T & R (4): cabal

Ebenfalls in der *Haskell Platform* enthalten ist `cabal`, Haskells eigenes Paket-Management-System. `cabal` erlaubt es euch, einfach via Terminal Bibliotheken von Hackage lokal (!) und im Zweifelsfall auch problemlos in einer Sandbox zu installieren. `cabal` ist außerdem ein Build-System, kann Test-Suites ausführen und und und. Aber auch dazu später mehr.

Das folgende Beispiel installiert `lens`, eine Bibliothek die uns im Lauf der Vorlesung noch begegnen wird:

```
$ cabal update && cabal install lens
```

T & R (5): LYAHFGG



Das Buch „Learn You A Haskell“ ist die besteTM Ressource um die ersten Schritte in Haskell zu lernen. Es ist verständlich und locker geschrieben und berühmt für gute Erklärungen.

Ihr findet es online frei und kostenlos verfügbar hier:
<http://learnyouahaskell.com/>

T & R (6): Hoogle

Hoogle ist eine Haskell-API-Suchmaschine, die es möglich macht, nach Funktionen zu suchen, ihren Typen angibt und zeigt, welche Imports nötig sind.

Ein besonders nützliches Feature ist, dass es auch möglich ist, nach Typsignaturen zu suchen und darauf passende Antworten zu erhalten.

Die URL lautet: `http://www.haskell.org/hoogle/`

Quer durch die Prelude Thinking in Types

Die Prelude ist Haskells Standardbibliothek und hat die wichtigsten Funktionen und Datentypen schon für euch zusammen gestellt.

Die wichtigsten Typen...

Bool	-- <i>True, False</i>
Int, Integer	-- <i>32bit und arbitrary precision</i>
Float, Double	-- <i>Floating-point</i>
Char, String	-- <i>Zeichen & Zeichenketten</i>
	-- <i>(String = [Char])</i>

Die wichtigsten Typen...

Bool -- *True, False*
Int, Integer -- *32bit und arbitrary precision*
Float, Double -- *Floating-point*
Char, String -- *Zeichen & Zeichenketten*
 -- *(String = [Char])*

... und Typkonstruktoren (polymorph)

[a] -- *Listen*
(a,b) -- *Tupel*
Maybe a -- *Option type, "Just a" oder "Nothing"*

QddP: Identity

```
id :: a -> a  
id x = x
```

Typsignaturen, Polymorphismus, Pattern matching

QddP: Konstanten

```
const :: a -> b -> a
```

```
const x _ = x
```

Currying, wildcards

QddP: Funktionsverkettung

$(.) :: (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow$

$(.) f g = \backslash x \rightarrow f (g x)$

Verkettung, Rechts nach Links lesen

QddP: Booleans (1)

```
data Bool = False | True
```

```
not :: Bool -> Bool
```

```
not True = False
```

```
not _    = True
```

QddP: Booleans (2)

`(&&) :: Bool -> Bool -> Bool`

`(&&) True True = True`

`(&&) _ _ = False`

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

`(||) False False = False`

`(||) _ _ = True`

QddP: Maybe (1)

```
data Maybe a = Nothing  
              | Just a
```

Tiefe etwa wie bei Bool

QddP: Either (1)

```
data Either a b = Left  a  
                | Right b
```

Tiefe etwa wie bei Bool

QddP: Tuples

foobar

QddP: Lists

foobar, Tiefe weiter als Maybe, Either, Bool

QddP: basic IO

foobar, Verweis auf später

Solving problems by composition

Beispiel:

Aufgabe: Sie bekommen einen Dateinamen übergeben.
Lesen Sie die Datei Zeile für Zeile aus