Compling Tutorium I

Anton Hampe Nils Nuernbergk

October 14, 2019

Hilfe!

Wen fragen?
Wenn ihr Hilfe braucht könnt ihr uns (bei Fragen zu Haskell, Editoren, eurem Rechner allegemein)
oder Professor Kobele (bei Fragen zu Informatik, Computerlinguistik, Benotung) gerne Kontaktieren.

Anton	mail@antonhampe.de
Nils	n.nuernbergk@studserv.uni-leipzig.de
Professor Kobele	gkobele@uni-leipzig.de

Wann sind die Tutorien?
Das bestimmt ihr mit, über die Doodle Poll! (Link)

Starting Out

Wozu dieses Tutorium?

In diesem Kurs lernt ihr die Grundlagen von Haskell, einer funktionalen Programmiersprache. Gegen Ende des Semesters werden wir dazu übergehen gemeinsam an euren eigenen Programmen zu arbeiten. Drei dieser Projekte werdet ihr bei Professor Kobele abgeben, so kommen eure Noten zustande.

Was brauchen wir für den Kurs?

Compiler & Interpreter	Haskell Platform
Editor (Empfehlung)	Windows, Linux: Visual Studio Code
	Mac: TextMate
Lehrbuch	Learn You a Haskell (Link)

Haskell Platform

Was bekommen wir mit dem Download? (Windows Link)

Compiler	GHC
Interpreter	GHCi
Build Tool	Stack

Was können wir damit anstellen?

Compiler	Code kompelieren
Interpreter	Code interaktiv austesten
Build Tool	Große Projekte organisieren, Packages installieren

Wie bekomme ich Haskell Platform für Linux?

Wir geben in einem Admin Account folgenden Befehl in der Kommandozeile ein:

sudo apt-get install haskell-platform

Die Kommandozeile (Windows)

Wo finden wir die Kommandozeile?

Windows	Win+R, "cmd", Enter
Linux	Ctrl+Alt+T

Was können wir in der Kommandozeile anstellen?

- Programme starten
- Dateien öffnen, umbenennen, speichern, bewegen
- Programme herunterladen, aktualisieren
- ...und vieles mehr!

Nützliche Befehle (Windows):

ghci	GHCi in der Kommandozeile starten
:! cd	Das aktuelle directory erfahren
:cd *dir*	*dir* zum aktuellen directory machen
:load *file*	*file* laden

Visual Studio Code

Was bekommen wir mit Visual Studio Code (VSCode)?
(Link)

- Einen Text-Editor
- Debugging
- Test-Suite
- Syntax-Highlighting
- Hover-for-Type
- Autocomplete
- ...und noch vieles mehr!

Haskelly und Haskell Syntax Hightlighting für VSCode

Vorerst macht VSCode nicht viel. Wir können Textdateien öffnen, bearbeiten und speichern. Diese Dateien können alles mögliche sein: Eine Beamer Präsentation, eine Haskell Datei, ein Latex Dokument und vieles mehr.

Damit wir die vielen nützlichen Optionen für Haskell nutzen können brauchen wir ein paar Erweiterungen:

VSCode	Haskelly
VSCode	Haskell Syntax Highlighting
Stack	intero
Stack	stack-run
Stack	QuickCheck

Haskelly und Haskell Syntax Hightlighting für VSCode

Um Erweiterungen in VSCode hinzuzufügen klicken wir links auf das Extensions-Icon, suchen nach der Erweiterung die wir wollen und installieren sie.
Haskelly braucht allerdings noch ein paar dependencies, die wir über Stack installieren müssen.
Dafür öffnen wir die Kommandozeile und geben folgende Befehle ein:

- stack upgrade
- stack install stack-run
- stack install intero
- stack install QuickCheck

(In Linux fügen wir vor den Befehlen noch "sudo" ein.)

Haskell?

Was ist Haskell?

Haskell ist eine *funktionale*, *nicht strikte*, *stark typisierte*, *high-level* Programmiersprache.

Was bedeutet das?

funktional	Haskell basiert auf dem Lambda-Kalkül
nicht strikt	Code wird nur soweit evaluiert, wie er gebraucht wird
stark stypisiert	Haskell Code ist sehr sicher; Alle Funktionen und Daten haben einen unveränderlichen Typ
high-level	Haskell Code abstrahiert sehr weit von der Maschinensprache

Haskell: Eine funktionale Sprache

Funktionales Programmieren
Den Kern von Haskell bilden Funtkionen.
So etwas wie Variablen gibt es in Haskell nicht.

Warum ist das gut?
Wir können einen Codeblock immer durch einen äquivalenten Codeblock ersetzen.
Wenn wir einmal eine Funktion $x = 5$ definieren, können wir immer 5 durch x ersetzen; der Wert von x wird sich nie verändern.
Man sagt: Code in Haskell hat keine **Nebeneffekte**.

Haskell: Eine nicht strikte Sprache

Nicht striktes Programmieren
Haskell ist egal wie lange ein Codeblock oder eine Liste brauchen würde um komplett abgearbeitet zu werden.
Haskell erledigt Aufgaben nur so weit wie sie gerade gebraucht werden.

Warum ist das gut?
Wir können mit unendlichen Listen arbeiten und trotzdem eine Aufgabe in endlicher Zeit erledigen.
Wenn wir eine unendliche Liste $xs = [1,2..]$ definieren, können wir Haskell sagen, dass wir nur die ersten drei Elemente der Liste brauchen und Haskell "schaut" nur auf die ersten drei Elemente der Liste.

Haskell: Eine stark typisierte Sprache

Stark typisiertes Programmieren
Haskell versucht ständig herauszufinden welchen Typ Funktionen und Daten haben. Wenn einmal ein Typ festgelegt ist, kann er nicht mehr verändert werden.

Warum ist das gut?
Wenn wir einmal eine Funktion $f\,x = x + 1$ definieren, dann weiß Haskell, dass die Funktion ein Argument vom Typ *Int* braucht. Wenn wir versuchen im Code die Funktion f mit dem *String* *"hallo"* zu füttern, dann beschwert sich Haskell schon beim kompelieren, weil *"hallo"* nicht den Typ *Int* hat.
So können wir schon vor dem kompelieren Fehler beheben, die später noch für große Probleme sorgen würden.

Haskell: Eine high-level Sprache

Kompelieren und Maschinensprache
Jedes Programm wird von unserem Rechner in Maschinensprache (Einsen und Nullen) ausgeführt. Wir können Programme direkt in Maschinensprache schreiben, aber das ist sehr umständlich.
Programmiersprachen sind Abstraktionen von der Maschinensprache, so wird der Code leichter verständlich für uns Menschen. Ein Compiler kompeliert den Code den wir in einer Programmiersprache schreiben. Das heißt er übersetzt unseren Code in Maschinensprache.

High-level und low-level Sprachen
Low-level Sprachen wie C sind sehr nah an der Maschinensprache, dadurch sind sie oft schneller. High-level Sprachen abstrahieren sehr viel, dadurch sind sie sehr bündig und elegant.
Haskell ist eine high-level Sprache, die trotzdem sehr schnell ist.

Unsere erste Funktion

Wie sehen Funktionen in Haskell aus?

Name	Argumente	=	Output
f		=	1
g	x	=	$x+1$
h	$x\ y$	=	$x+y$

Wo können wir eine Funktion definieren?

- In GHCi
- In einer .hs Datei

Unsere erste Funktion

GHCi starten und beenden

Wir öffnen entweder die GHCi Anwendung über das Icon, oder geben in der Kommandozeile den Befehl *ghci* ein.
Um GHCi in der Kommandozeile zu beenden benutzen wir *:quit*.
Für eine Übersicht aller Befehle geben wir *:?* ein.

Unsere Funktion in GHCi definieren

Wir definieren Funktionen nach dem Muster auf der letzten Folie.
Der Befehl *addone x = x+1* definiert eine Funktion, die einen *Int* als Argument nimmt, eins addiert und das Ergebnis zurückgibt.

Unsere Funktion in GHCi aufrufen

Diese Funktion können wir jetzt aufrufen, indem wir den Namen gefolgt von einem Argument eingeben.
Zum Beispiel: *addone 4*

Unsere erste .hs Datei

Eine neue .hs Datei erstellen
Wenn wir GHCi schließen sind alle Funktionen die wir definiert haben wieder weg. Wenn wir Code speichern wollen, machen wir das in einer .hs Datei.
Wir starten unseren Editor, erstellen eine neue Datei und geben die Definition unserer Funktion ein. Danach speichern wir diese Datei mit einer .hs Endung.
Zum Beispiel: tutorium.hs

Eine .hs Datei laden (VSCode)
In VSCode können wir unten links auf den "Load GHCi" Knopf klicken um die Datei in GHCi zu laden.

Unsere erste .hs Datei

Eine .hs Datei laden (GHCi)
In GHCi können wir eine Datei nur öffnen, wenn wir uns im richtigen directory (Ordner) befinden.
Um das aktuelle directory anzuzeigen geben wir *:! cd* ein.
Um in das directory unserer .hs zu navigieren, benutzen wir *:cd dir*, wobei wir *dir* durch den Dateipfad unserer .hs ersetzen.
Zum Beispiel: *:cd C:\Users\Anton \Desktop\Compling Tutorium*

Wenn wir im richtigen directory sind, können wir die .hs mit *:load file* laden, wobei wir *file* durch den Namen unserer Datei ersetzen.
Zum Beispiel: *:load tutorium*

Jetzt können wir die Funktionen, die wir in der .hs gespeichert haben wie gewohnt aufrufen und mit Argumenten füttern.