Einführung in die Funktionale Programmierung mit Haskell

für Studierende mit Nebenfach Informatik

Steffen Jost

LFE Theoretische Informatik, Institut für Informatik,
Ludwig-Maximilians Universität, München

17. April 2013

Veranstalter

Dr Steffen Jost

2010–	Wissenschaftlicher Mitarbeiter am Lehrstuhl für Theoretische Informatik, Prof. Hofmann
2005–2010	Research Fellow, St Andrews, Schottland
2002–2005	Doktorand am Lehrstuhl für Theoretische Informatik
1995–2002	Diplomstudium Mathematik, TU Darmstadt

Forschungsschwerpunkte:

- Automatisierte Programmanalyse
- Typsysteme
- Formale Methoden
- Funktionale Programmierung

Sprechstunde SoSe2013: Do 14:00h in E109

Hinweise

- Die Vorlesung ist ausgerichtet auf Studierende mit Nebenfach Informatik.
- Aufgrund von Themenüberschneidungen mit der Vorlesung
  “Programmierung & Modellierung”
  (SoSe2013: Mo10-13 bei Prof. Bry, PMS)
  können Studierende nur eines dieser beiden Module einbringen.
- Für kurzfristige Ankündungen bitte die Nachrichten auf der Vorlesungshomepage beachten:
  `http://www.tcs.ifi.lmu.de/lehre/ss-2013/efpn`
  Dazu gibt es auch einen RSS-Feed
- Prüfung erfolgt durch Klausur am Ende der Vorlesung.

Planung

Sollumfang: 3+2 SWS **Raumbuchung:** 4+2 SWS

$\Rightarrow$ 20 Vorlesungen + 12 Übungen (jeweils 2 Stunden)

Frage: Können wir Vorlesung & Übung gelegentlich tauschen?

Vorläufige Planung:

Mi Vorlesung	Do Vorlesung	Fr Übung
17. Apr 13	18. Apr 13	19. Apr 13
24. Apr 13	25. Apr 13	26. Apr 13
1. Mai 13	2. Mai 13	3. Mai 13
8. Mai 13	9. Mai 13	10. Mai 13
15. Mai 13	16. Mai 13	17. Mai 13
22. Mai 13	23. Mai 13	24. Mai 13
29. Mai 13	30. Mai 13	31. Mai 13
5. Jun 13	6. Jun 13	7. Jun 13
12. Jun 13	13. Jun 13	14. Jun 13
19. Jun 13	20. Jun 13	21. Jun 13
26. Jun 13	27. Jun 13	28. Jun 13
3. Jul 13	4. Jul 13	5. Jul 13
10. Jul 13	11. Jul 13	12. Jul 13
17. Jul 13	18. Jul 13	19. Jul 13

42 Termine, davon
3 Feiertage
1 Klausurtermin
6 Entfällt
= 32 Veranstaltungen

Literatur

Die Vorlesung richtet sich nach folgenden Quellen:

	Buch/Online	
Learn You a Haskell for Great Good! by Miran Lipovača	B	O
A Gentle Introduction To Haskell by Paul Hudak, John Peterson, Joseph Fasel		O
Real World Haskell by Bryan O'Sullivan, Don Stewart, John Goerzen	B	O
Programming in Haskell by Graham Hutton	B	
School of Haskell by FPComplete		O
Haskell Wiki		O

so wie diverse Skripte des Lehrstuhls TCS.
Links/ISBN der Quellen, Skript ⇒ Vorlesungshomepage

Weitere wichtige Online-Quelle: www.haskell.org

Haskell-Platform: GHC Kompiler & Interpreter,
Dokumentation der Standardbibliotheken, Hoogle

Inhalte

Einführung in die Funktionale Programmierung mit Haskell

- Funktionsbegriff, Haskell Syntax und erste Schritte
- Typen, Listen und Polymorphie
- Rekursion und Terminierung
- Funktionen Höherer Ordnung
- Benutzerdefinierte Datenstrukturen
- Klassen
- Monaden und I/O

  — Optionale Themen —
- Funktoren
- Parallele Auswertung
- Typsysteme
- Verzögerte Auswertung
- Semantik

Motivation

Heute wollen wir folgende Fragen klären:

- Warum ist Funktionale Programmierung sinnvoll?
- Warum schauen wir uns dazu gerade Haskell an?
- Warum ist das speziell für das Nebenfach Informatik interessant?

Hinweis:
Diese Fragen kann man nicht ganz eindeutig abschließend klären, meine Antworten darauf beruhen auf meiner persönlichen Meinung.

Ich versuche heute einen informellen und intuitiven Ausblick darauf zu geben. Ein tieferes Verständnis, und damit Eure persönliche Antworten, kann erst **am Ende des Semesters** erwartet werden.

Motivation

Heute wollen wir folgende Fragen klären:

- Warum ist Funktionale Programmierung sinnvoll?
- Warum schauen wir uns dazu gerade Haskell an?
- Warum ist das speziell für das Nebenfach Informatik interessant?

Hinweis:
Diese Fragen kann man nicht ganz eindeutig abschließend klären, meine Antworten darauf beruhen auf meiner persönlichen Meinung.

Ich versuche heute einen informellen und intuitiven Ausblick darauf zu geben. Ein tieferes Verständnis, und damit Eure persönliche Antworten, kann erst **am Ende des Semesters** erwartet werden.

Warum Funktionale Programmierung?

Funktionale Programmierung mit Haskell erlaubt einfache und billige Softwareentwicklung:

- Schnell kann schneller Code geschrieben werden
- Code ist kompakt, gut lesbar und damit gut zu warten
- Korrektheitsbeweise von Programmen relativ leicht
- Nebenläufigkeit ist unproblematisch

> *I learned Haskell a couple of years ago, having previously programmed in Python and (many) other languages. Recently, I've been using Python for a project (...), and find my Python programming style is now heavily influenced (for the better, I hope ;-) by my Haskell programming experience.*
>
> *Graham Klyne, Haskell-Wiki*

Imperativ vs. Deklarativ

Imperative Sprachen (Assembler, C, Java, etc.) orientieren sich an den Fähigkeiten der Maschine. Programmierung wird durch Abstraktion vieler Einzelschritte vereinfacht (z.B. Schleifen).

Funktionale Programmierung ist dagegen **deklarativ**:

> *Spezifiziere* **was** *berechnet werden soll,*
> **nicht wie** *es berechnet wird!*

Charakteristische Eigenschaft deklarativer Sprachen ist die **Church-Rosser Eigenschaft**:

> *Die Reihenfolge der Auswertung ist unerheblich für das Ergebnis der Berechnung ist.*

Imperativ vs. Deklarativ

Imperative Sprachen (Assembler, C, Java, etc.) orientieren sich an den Fähigkeiten der Maschine. Programmierung wird durch Abstraktion vieler Einzelschritte vereinfacht (z.B. Schleifen).

Funktionale Programmierung ist dagegen **deklarativ**:

> *Spezifiziere* **was** *berechnet werden soll,*
> **nicht wie** *es berechnet wird!*

Charakteristische Eigenschaft deklarativer Sprachen ist die **Church-Rosser Eigenschaft**:

> *Die Reihenfolge der Auswertung ist unerheblich für das Ergebnis der Berechnung ist.*

Imperativ vs. Deklarativ

Deklarative Sprachen sind fast immer durch die Sprache der Mathematik motiviert:

Prädikaten-Logik	$\Rightarrow$	Prolog
Relationale Algebra	$\Rightarrow$	SQL
λ-Kalkül	$\Rightarrow$	Haskell

Rein deklarative Sprachen in der Praxis jedoch oft problematisch, da der Ressourcenverbrauch eines Programmes (Speicherplatz, Zeit, etc.) kritisch vom **wie** abhängt. (Prolog, SQL ohne join).

Funktionale Programmierung geht einen Mittelweg:

Die Berechnung bleibt **deterministisch**

(wir kümmern uns nur nicht so sehr darum)

Imperativ vs. Deklarativ

Deklarative Sprachen sind fast immer durch die Sprache der Mathematik motiviert:

Prädikaten-Logik	$\Rightarrow$	Prolog
Relationale Algebra	$\Rightarrow$	SQL
λ-Kalkül	$\Rightarrow$	Haskell

Rein deklarative Sprachen in der Praxis jedoch oft problematisch, da der Ressourcenverbrauch eines Programmes (Speicherplatz, Zeit, etc.) kritisch vom **wie** abhängt. (Prolog, SQL ohne join).

Funktionale Programmierung geht einen Mittelweg:

Die Berechnung bleibt **deterministisch**

(wir kümmern uns nur nicht so sehr darum)

Referentielle Transparenz

Eine wichtige Eigenschaft deklarativer Sprachen ist die **Referentielle Transparenz**

- Der Wert einer Variablen ist **unveränderlich**
- Aufrufe mit gleichen Argumenten liefern gleiches Ergebnis
- Keine Seiteneffekte!

Referentielle Transparenz

Konsequenzen:

- Programme lokal verständlich und kompositional.
- Testen/Verifikation reduziert sich auf Gleichheitsschließen.
- Kompiler kann sehr aggressiv optimieren, da die Berechnungsreihenfolge komplett unbedeutend ist.
- Parallele Berechnung einfacher: **Verzahnung** fast ohne Effekt.
- **Persistenz & Sharing:** Keine Probleme durch **Aliasing** erlaubt die Wiederverwendung von Datenstruktur(-teilen). *Nachteil:* Veränderung benötigen zusätzliche Kopien, was andere Herangehensweise an effiziente Datenhaltung erzwingt

"Purely Functional Data Structures", Okasaki

Nachteile Funktionaler Programmierung

FP ist alt, doch kaum verbreitet?! λ-Kalkül 1930s, Lisp 1958

- Völlig andere (mathematische) Denkweise
- Großer Kontrollverlust gegenüber herkömmlichen Sprachen
- Maschinen-nahe Hand-Optimierung schwierig durchführbar
- Oft geringe kommerzielle Unterstützung IDE, Debugger,...
- Gute Merkmale funktionaler Programmierung werden mittlerweile auch in anderen Sprachen angeboten,
  z.B. Funktionen höherer Ordnung, Anonyme Funktionen, Java Garbage Collector inzwischen auf viele kurzlebige Objekte ausgerichtet, etc.

Die Einschränkung, der Maschine keine direkten Befehle geben zu können, wird von vielen Informatikern als unangenehm empfunden. Die daraus gewonnene Sicherheit ist jedoch vorteilhaft für *Mehrpersonen*-Projekte und Wartung!

Nachteile Funktionaler Programmierung

FP ist alt, doch kaum verbreitet?! λ-Kalkül 1930s, Lisp 1958

- Völlig andere (mathematische) Denkweise
- Großer Kontrollverlust gegenüber herkömmlichen Sprachen
- Maschinen-nahe Hand-Optimierung schwierig durchführbar
- Oft geringe kommerzielle Unterstützung IDE, Debugger,...
- Gute Merkmale funktionaler Programmierung werden mittlerweile auch in anderen Sprachen angeboten,
  z.B. Funktionen höherer Ordnung, Anonyme Funktionen, Java Garbage Collector inzwischen auf viele kurzlebige Objekte ausgerichtet, etc.

Die Einschränkung, der Maschine keine direkten Befehle geben zu können, wird von vielen Informatikern als unangenehm empfunden. Die daraus gewonnene Sicherheit ist jedoch vorteilhaft für *Mehrpersonen*-Projekte und Wartung!

Nachteile Funktionaler Programmierung

FP ist alt, doch kaum verbreitet?! λ-Kalkül 1930s, Lisp 1958

- Völlig andere (mathematische) Denkweise
- Großer Kontrollverlust gegenüber herkömmlichen Sprachen
- Maschinen-nahe Hand-Optimierung schwierig durchführbar
- Oft geringe kommerzielle Unterstützung IDE, Debugger,...
- Gute Merkmale funktionaler Programmierung werden mittlerweile auch in anderen Sprachen angeboten,
  z.B. Funktionen höherer Ordnung, Anonyme Funktionen, Java Garbage Collector inzwischen auf viele kurzlebige Objekte ausgerichtet, etc.

Die Einschränkung, der Maschine keine direkten Befehle geben zu können, wird von vielen Informatikern als unangenehm empfunden. Die daraus gewonnene Sicherheit ist jedoch vorteilhaft für *Mehrpersonen*-Projekte und Wartung!

Haskell

- Effektfreie funktionale Sprache mit verzögerter Auswertung
- Benannt nach Haskell Curry (1900-82), Logiker
- Standards: Haskell98 und Haskell2010
- Viele verschiedene Implementierung verfügbar; wichtigste ist die **Haskell Platform** mit "batteries included": Kompiler, Interpreter und zahlreichen Bibliotheken

  GHC : Glasgow/Glorious Haskell Compiler — Hammond, 1989
  Hauptentwickler:
  Simon Peyton-Jones — Microsoft Research Cambridge
  Simon Marlow — Facebook

Warum gerade Haskell?

Für einen großen Teil der Vorlesungsinhalte spielt es keine Rolle, welche funktionale Sprache wir betrachten.

- Haskell ist im Gegensatz zu vielen anderen funktionalen Sprachen (z.B. SML) **rein funktional**, d.h. referentielle Transparenz gilt uneingeschränkt
- Sauber an der Mathematik orientierte Notation
  list comprehension, where-Klauseln, etc.
- Haskell hat ein starkes statisches Typsystem
- Klare Unterstützung für Überladen durch Typklassen
- Sehr gute Unterstützung für Monaden
- Einfaches Modulsystem
- Arithmetik mit beliebiger Präzision verfügbar
- Gute Unterstützung für Parallelität
- Produziert sehr “flotten” Code

Warum gerade Haskell?

Die Folklore zum Thema Geschwindigkeit kann man exemplarisch wie folgt beschreiben:

> *Nothing can beat highly micro-optimised C, but real everyday C code is not like that, and is often several times slower than the micro-optimised version would be.*
>
> *Meanwhile the high level of Haskell means that the compiler has lots of scope for doing micro-optimisation of its own. As a result it is quite common for everyday Haskell code to run faster than everyday C.*
>
> *Paul Johnson, Blog*

Aber Folklore ist natürlich immer mit viel Vorsicht zu geniessen!

Auch die Entwicklungszeit eines Programms, welches “schnell genug” ist, hängt von vielen individuellen Parametern ab.

Warum gerade Haskell?

. . . und Haskell ist gerade hip:

- 2011 ACM SIGPLAN Programming Languages Software Award an Hauptentwickler Simon Peyton-Jones und Simon Marlow
- Unterstützung durch kommerzielle Firmen hat begonnen

  `fpcomplete.com` entwickelt IDE
- Viel gute und aktuelle Dokumentation verfügbar
- Große aktive Online Gemeinschaft
- Aktuelle GHC Versionen mehrmals pro Jahr
- Haskell-Artikel in deutscher IT-Presse

  c't 23/2012, iX Developer Special 1/2013, etc.

Neue kommerzielle Interessen werden Multi-Cores zugeschrieben

"The Downfall of Imperative Programming", Milewski

Probleme mit Haskell

- Verzögerte Auswertung macht es sehr schwierig zu erkennen, wie ein Programm ausgewertet wird. Klassisches Debugging wird dadurch nahezu unmöglich – aber auch nicht notwendig
- Rein funktionale Programmierung erzwingt Umdenken in allen Situation
- Kompromissloses Typsystem liefert viele lange Fehlermeldung, bis es endlich mal kompiliert

"Well-typed programs can't go wrong!", Milner

Das alles läuft der "mal kurz verbiegen und ausprobieren" Mentalität zuwider. Durch Ausnutzen der Modularität kann man jedoch trotzdem Programmteile einzeln testen.

- Andere Informatik Vorlesungen an der LMU benutzen SML

Probleme mit Haskell

- Verzögerte Auswertung macht es sehr schwierig zu erkennen, wie ein Programm ausgewertet wird. Klassisches Debugging wird dadurch nahezu unmöglich – aber auch nicht notwendig
- Rein funktionale Programmierung erzwingt Umdenken in allen Situation
- Kompromissloses Typsystem liefert viele lange Fehlermeldung, bis es endlich mal kompiliert

"Well-typed programs can't go wrong!", Milner

Das alles läuft der "mal kurz verbiegen und ausprobieren" Mentalität zuwider. Durch Ausnutzen der Modularität kann man jedoch trotzdem Programmteile einzeln testen.

- Andere Informatik Vorlesungen an der LMU benutzen SML

Warum gerade für das Nebenfach Informatik?

Fähigkeit ist nicht die Frage:

Meine Lehrerfahrung zeigt mir, dass es Studierende mit Nebenfach Informatik gibt, welche auch in herkömmlicher Informatik exzellent sind.

Interesse kann aber variieren:

Als Mathematiker mit Nebenfach Informatik fand ich andere Themengebiete der Informatik spannend, als viele meiner Kommilitonen mit Hauptfach Informatik.

Diese Vorlesung soll also keinesfalls einfacher oder schwerer sein als solche für Studierende mit Hauptfach Informatik, sondern auf andere Interessenbedürfnisse zugeschnitten sein.

Warum gerade für das Nebenfach Informatik?

- Basis in mathematischer Logik vermutlich hilfreich für Studierende aus Mathematik-nahen Studienrichtungen.
- Problem-fokussierte Softwareentwicklung, technische Details können in den Hintergrund treten, wenn man sich nicht speziell damit befassen möchte.
- Komfortabler und leicht zu benutzender Interpreter auch im Alltag schnell nützlich.
- Softwareoptimierung muss nicht am Anfang der Entwicklung stehen, sondern kann erst bei Bedarf leicht nachträglich durchgeführt werden.
- Umdenken fällt vielleicht leichter?

Bitte gebt mir laufend Feedback, was Ihr von der Veranstaltungen haltet und welche Themen besonders interessieren!

Zusammenfassung

Warum Funktionale Programmierung?

- Kurze Entwicklungszeiten
- Gute Lesbarkeit und Wartbarkeit des Codes
- Nebenläufigkeit unproblematisch

Warum Haskell?

- Reinste Funktionale Programmierung ohne Kompromiss
- Schnelle Software dank aggressiver automatischer Optimierung
- Aktuelle Dokumentation und gute Unterstützung

Warum speziell für das Nebenfach Informatik?

- Fokus auf Problem, nicht auf technische Details
- Etwas Abseits der stereotypischen Denkweise der Informatik