Praktische Informatik 3: Funktionale Programmierung
Vorlesung 14 vom 31.01.15: Rückblick & Ausblick

Christoph Lüth

Universität Bremen

Wintersemester 2016/17

Fahrplan

- Teil I: Funktionale Programmierung im Kleinen
- Teil II: Funktionale Programmierung im Großen
- Teil III: Funktionale Programmierung im richtigen Leben
  - Aktionen und Zustände
  - Monaden als Berechnungsmuster
  - Domänenspezifische Sprachen (DSLs)
  - Scala — Eine praktische Einführung
  - Rückblich & Ausblick

Organisatorisches

- Bitte an der Online-Evaluation teilnehmen (stud.ip)

Inhalt

- Fachgespräche und Prüfungen
- Rückblick und Ausblick

Fachgespräche und Prüfungen

Fachgespräche

- Fachgespräche bestehen aus der schriftlichen, nichtelektronischen Bearbeitung einer kurzen Programmieraufgabe
  - Abstufung nach Vornote: A (1– 1.7), B (2.0 – 3.0), C (3.3 – 4.0)
- Beispielfragen auf der Webseite (unter “Übungsaufgaben”)

Beispielfrage (C)

Definieren Sie eine Funktion format, die eine Zahl in einer Zeichenkette gegebener Länge rechtsbündig ausgibt.

Bsp. format 4 xy $\rightsquigarrow$ "␣␣xy"

Beispielfrage (B)

Definieren Sie eine Funktion mean, die den arithmetischen Durchschnitt einer Liste von ganzen Zahlen berechnet.

Bsp. mean [2,1,5,4,3] $\rightsquigarrow$ 3.0

Beispielfrage (A)

Zwei Int-Listen sollen ähnlich heißen, wenn Sie die gleichen Zahlen unabhängig von ihrer Reihenfolge und Häufigkeit enthalten. Schreiben Sie eine Testfunktion similar dafür.

Bsp. similar [3,2,2,1,3] [1,2,3] $\rightsquigarrow$ True

Mündliche Prüfung

- Dauer: in der Regel 30 Minuten
- Einzelprüfung, ggf. mit Beisitzer
- Inhalt: Programmieren mit Haskell und Vorlesungsstoff
- Ablauf: “Fachgespräch plus”
  - Einstieg mit leichter Programmieraufgabe wie im Fachgespräch
  - Daran anschließend Fragen über den Stoff

Verständnisfragen

Auf allen Übungsblättern finden sich Verständnisfragen zur Vorlesung. Diese sind nicht Bestandteil der Abgabe, können aber im Fachgesprach thematisiert werden. Wenn Sie das Gefühl haben, diese Fragen nicht sicher beantworten zu können, wenden Sie sich gerne an Ihren Tutor, an Berthold Hoffmann in seiner Fragestunde, oder an den Dozenten.

Verständnisfragen

1. Was bedeutet Striktheit, und welche in Haskell definierbaren Funktionen haben diese Eigenschaft?

Verständnisfragen

1. Was bedeutet Striktheit, und welche in Haskell definierbaren Funktionen haben diese Eigenschaft?

2. Was ist ein algebraischer Datentyp, und was ist ein Konstruktor?

Verständnisfragen

1. Was bedeutet Striktheit, und welche in Haskell definierbaren Funktionen haben diese Eigenschaft?

2. Was ist ein algebraischer Datentyp, und was ist ein Konstruktor?

3. Was sind die drei Eigenschaften, welche die Konstruktoren eines algebraischen Datentyps auszeichnen, was ermöglichen sie und warum?

Verständnisfragen: Übungsblatt 2

1. Welche zusätzliche Mächtigkeit wird durch Rekursion bei algebraischen Datentypen in der Modellierung erreicht? Was läßt sich mit rekursiven Datentypen modellieren, was sich nicht durch nicht-rekursive Datentypen erreichen läßt?

Verständnisfragen: Übungsblatt 2

1. Welche zusätzliche Mächtigkeit wird durch Rekursion bei algebraischen Datentypen in der Modellierung erreicht? Was läßt sich mit rekursiven Datentypen modellieren, was sich nicht durch nicht-rekursive Datentypen erreichen läßt?

2. Was ist der Unterschied zwischen Bäumen und Graphen, in Haskell modelliert?

Verständnisfragen: Übungsblatt 2

1. Welche zusätzliche Mächtigkeit wird durch Rekursion bei algebraischen Datentypen in der Modellierung erreicht? Was läßt sich mit rekursiven Datentypen modellieren, was sich nicht durch nicht-rekursive Datentypen erreichen läßt?

2. Was ist der Unterschied zwischen Bäumen und Graphen, in Haskell modelliert?

3. Was sind die wesentlichen Gemeinsamkeiten, und was sind die wesentlichen Unterschiede zwischen algebraischen Datentypen in Haskell, und Objekten in Java?

Verständnisfragen: Übungsblatt 3

1. Was ist Polymorphie?

Verständnisfragen: Übungsblatt 3

1. Was ist Polymorphie?

2. Welche zwei Arten der Polymorphie haben wir kennengelernt, und wie unterschieden sie sich?

Verständnisfragen: Übungsblatt 3

1. Was ist Polymorphie?

2. Welche zwei Arten der Polymorphie haben wir kennengelernt, und wie unterschieden sie sich?

3. Was ist der Unterschied zwischen Polymorphie in Haskell, und Polymorphie in Java?

Verständnisfragen: Übungsblatt 4

1. Was kennzeichnet strukturell rekursive Funktionen, wie wir sie in der Vorlesung kennengelernt haben, und wie sind sie durch die Funktion `foldr` darstellbar?

Verständnisfragen: Übungsblatt 4

1. Was kennzeichnet strukturell rekursive Funktionen, wie wir sie in der Vorlesung kennengelernt haben, und wie sind sie durch die Funktion `foldr` darstellbar?

2. Welche anderen geläufigen Funktionen höherer Ordnung kennen wir?

Verständnisfragen: Übungsblatt 4

1. Was kennzeichnet strukturell rekursive Funktionen, wie wir sie in der Vorlesung kennengelernt haben, und wie sind sie durch die Funktion `foldr` darstellbar?

2. Welche anderen geläufigen Funktionen höherer Ordnung kennen wir?

3. Was ist η-Kontraktion, und warum ist es zulässig?

Verständnisfragen: Übungsblatt 4

1. Was kennzeichnet strukturell rekursive Funktionen, wie wir sie in der Vorlesung kennengelernt haben, und wie sind sie durch die Funktion `foldr` darstellbar?

2. Welche anderen geläufigen Funktionen höherer Ordnung kennen wir?

3. Was ist η-Kontraktion, und warum ist es zulässig?

4. Wann verwendet man `foldr`, wann `foldl`, und unter welchen Bedingungen ist das Ergebnis das gleiche?

Verständnisfragen: Übungsblatt 5

1. `foldr` ist die „kanonische einfach rekursive Funktion“ (Vorlesung). Was bedeutet das, und warum ist das so? Für welche Datentypen gilt das?

Verständnisfragen: Übungsblatt 5

1. `foldr` ist die „kanonische einfach rekursive Funktion“ (Vorlesung). Was bedeutet das, und warum ist das so? Für welche Datentypen gilt das?

2. Wann kann `foldr f a xs` auch für ein zyklisches Argument `xs` (bspw. eine zyklische Liste) terminieren?

Verständnisfragen: Übungsblatt 5

1. `foldr` ist die „kanonische einfach rekursive Funktion“ (Vorlesung). Was bedeutet das, und warum ist das so? Für welche Datentypen gilt das?

2. Wann kann `foldr f a xs` auch für ein zyklisches Argument `xs` (bspw. eine zyklische Liste) terminieren?

3. Warum sind endrekursive Funktionen im allgemeinen schneller als nicht-endrekursive Funktionen? Unter welchen Voraussetzungen kann ich eine Funktion in endrekursive Form überführen?

Verständnisfragen: Übungsblatt 6

1. Was ist ein abstrakter Datentyp (ADT)?

Verständnisfragen: Übungsblatt 6

1. Was ist ein abstrakter Datentyp (ADT)?

2. Was sind Unterschiede und Gemeinsamkeiten zwischen ADTs und Objekten, wie wir sie aus Sprachen wie Java kennen?

Verständnisfragen: Übungsblatt 6

1. Was ist ein abstrakter Datentyp (ADT)?

2. Was sind Unterschiede und Gemeinsamkeiten zwischen ADTs und Objekten, wie wir sie aus Sprachen wie Java kennen?

3. Wozu dienen Module in Haskell?

Verständnisfragen: Übungsblatt 7

1. Wie können wir die Typen und Operationen der Signatur eines abstrakten Datentypen grob klassifizieren, und welche Auswirkungen hat diese Klassifikation auf die zu formulierenden Eigenschaften?

Verständnisfragen: Übungsblatt 7

1. Wie können wir die Typen und Operationen der Signatur eines abstrakten Datentypen grob klassifizieren, und welche Auswirkungen hat diese Klassifikation auf die zu formulierenden Eigenschaften?

2. Warum „finden Tests Fehler“, aber „zeigen Beweise Korrektheit“, wie in der Vorlesung behauptet? Stimmt das immer?

Verständnisfragen: Übungsblatt 7

1. Wie können wir die Typen und Operationen der Signatur eines abstrakten Datentypen grob klassifizieren, und welche Auswirkungen hat diese Klassifikation auf die zu formulierenden Eigenschaften?

2. Warum „finden Tests Fehler“, aber „zeigen Beweise Korrektheit“, wie in der Vorlesung behauptet? Stimmt das immer?

3. Müssen Axiome immer ausführbar sein? Welche Axiome wären nicht ausführbar?

Verständnisfragen: Übungsblatt 8

1. Der Datentyp Stream α ist definiert als

   data Stream α = Cons α (Stream α)

   Gibt es für diesen Datentyp ein Induktionsprinzip? Ist es sinnvoll?

Verständnisfragen: Übungsblatt 8

1. Der Datentyp Stream α ist definiert als

   data Stream α = Cons α (Stream α)

   Gibt es für diesen Datentyp ein Induktionsprinzip? Ist es sinnvoll?

2. Welche nichtausführbaren Prädikate haben wir in der Vorlesung kennengelernt?

Verständnisfragen: Übungsblatt 8

1. Der Datentyp Stream α ist definiert als

   data Stream α = Cons α (Stream α)

   Gibt es für diesen Datentyp ein Induktionsprinzip? Ist es sinnvoll?

2. Welche nichtausführbaren Prädikate haben wir in der Vorlesung kennengelernt?

3. Wie kann man in einem Induktionsbeweis die Induktionsvoraussetzung stärken, und wann ist das nötig?

Verständnisfragen: Übungsblatt 9

1. Warum ist die Erzeugung von Zufallszahlen eine Aktion?

Verständnisfragen: Übungsblatt 9

1. Warum ist die Erzeugung von Zufallszahlen eine Aktion?

2. Warum ist auch das Schreiben in eine Datei eine Aktion?

Verständnisfragen: Übungsblatt 9

1. Warum ist die Erzeugung von Zufallszahlen eine Aktion?

2. Warum ist auch das Schreiben in eine Datei eine Aktion?

3. Was ist (bedingt durch den Mangel an referentieller Transparenz) die entscheidende Eigenschaft, die Aktionen von reinen Funktionen unterscheidet?

Rückblick und Ausblick

Warum funktionale Programmierung lernen?

- Funktionale Programmierung macht aus Programmierern Informatiker
- Blick über den Tellerrand — was kommt in 10 Jahren?
- Herausforderungen der Zukunft
- Enthält die wesentlichen Elemente moderner Programmierung

Zusammenfassung Haskell

Stärken:

- Abstraktion durch
  - Polymorphie und Typsystem
  - algebraische Datentypen
  - Funktionen höherer Ordnung
- Flexible Syntax
- Haskell als Meta-Sprache
- Ausgereifter Compiler
- Viele Büchereien

Schwächen:

- Komplexität
- Büchereien
  - Nicht immer gut gepflegt
- Viel im Fluß
  - Kein stabiler und brauchbarer Standard
- Divergierende Ziele:
  - Forschungsplattform und nutzbares Werkzeug

Andere Funktionale Sprachen

- Standard ML (SML):
  - Streng typisiert, strikte Auswertung
  - Standardisiert, formal definierte Semantik
  - Drei aktiv unterstütze Compiler
  - Verwendet in Theorembeweisern (Isabelle, HOL)
  - `http://www.standardml.org/`
- Caml, O'Caml:
  - Streng typisiert, strikte Auswertung
  - Hocheffizienter Compiler, byte code & nativ
  - Nur ein Compiler (O'Caml)
  - `http://caml.inria.fr/`

Andere Funktionale Sprachen

- LISP und Scheme
  - Ungetypt/schwach getypt
  - Seiteneffekte
  - Viele effiziente Compiler, aber viele Dialekte
  - Auch industriell verwendet
- Hybridsprachen:
  - Scala (Functional-OO, JVM)
  - F# (Functional-OO, .Net)
  - Clojure (Lisp, JVM)

Was spricht gegen funktionale Programmierung?

- Mangelnde Unterstützung:
  - Libraries, Dokumentation, Entwicklungsumgebungen
  - Wird besser (Scala)...
- Programmierung nur kleiner Teil der SW-Entwicklung
- Nicht verbreitet — funktionale Programmierer zu teuer
- Konservatives Management
  - "Nobody ever got fired for buying IBM"

Was spricht gegen funktionale Programmierung?

- Mangelnde Unterstützung:
  - Libraries, Dokumentation, Entwicklungsumgebungen
  - Wird besser (Scala)...
- Programmierung nur kleiner Teil der SW-Entwicklung
- Nicht verbreitet — funktionale Programmierer zu teuer
- Konservatives Management
  - "Nobody ever got fired for buying SAP"

Haskell in der Industrie

- Simon Marlow bei Facebook: Sigma — Fighting spam with Haskell
- Finanzindustrie: Barclays Capital, Credit Suisse, Deutsche Bank
- Bluespec: Schaltkreisentwicklung, DSL auf Haskell-Basis
- Galois, Inc: Cryptography (Cryptol DSL)
- Siehe auch: Haskell in Industry

Funktionale Programmierung in der Industrie

- Scala:
  - Twitter, Foursquare, Guardian, ...
- Erlang
  - schwach typisiert, nebenläufig, strikt
  - Fa. Ericsson (Telekom-Anwendungen), WhatsApp
- `FL`
  - ML-artige Sprache
  - Chip-Verifikation der Fa. Intel
- Python (und andere Skriptsprachen):
  - Listen, Funktionen höherer Ordnung (map, fold), anonyme Funktionen, Listenkomprehension

Perspektiven funktionaler Programmierung

- Forschung:
  - Ausdrucksstärkere Typsysteme
  - für effiziente Implementierungen
  - und eingebaute Korrektheit (Typ als Spezifikation)
  - Parallelität?
- Anwendungen:
  - Eingebettete domänenspezifische Sprachen
  - Zustandsfreie Berechnungen (MapReduce, Hadoop, Spark)
  - Big Data and Cloud Computing

If you liked this course, you might also like . . .

- Die Veranstaltung Reaktive Programmierung (Sommersemester 2017)
  - Scala, nebenläufige Programmierung, fortgeschrittene Techniken der funktionalen Programmierung
- Wir suchen studentische Hilfskräfte am DFKI, FB CPS
  - Scala als Entwicklungssprache
- Wir suchen Tutoren für PI3
  - Im WS 2017/18 — meldet Euch bei Berthold Hoffmann (oder bei mir)!

An Important Public Service Announcement

Tschüß!