Praktische Informatik 3: Funktionale Programmierung

Vorlesung 14 / 01. Februar 2022

Rückblick und Ausblick

Thomas Barkowsky

Wintersemester 2021/22

Organisatorisches

- E-Klausur am 10. März 2022
- Anmeldung zur E-Klausur über Stud.IP bis 28. Februar 2022
  - unabhängig von PABO-Anmeldung
- Evaluation der Veranstaltung über Stud.IP
  - bitte teilnehmen!
  - Lob & Tadel
  - konstruktive Kritik
  - Verbesserungsideen

heute in dieser Vorlesung...

- Rückblick
  - Was haben wir gelernt?
  - Funktionale Programmierung und Abstraktion
- Ausblick
  - Haskell in der Industrie
  - die Zukunft der funktionalen Programmierung
- Prüfung
  - Scheinkriterien
  - Beispiele zur E-Klausur

Rückblick

Die wichtigen Themen

- Vgl. Ziele und Inhalte lt. Modulhandbuch

- Ziele:
  - Konzepte und typische Merkmale funktionalen Programmierens verstehen und anwenden können

  - Datenstrukturen und Algorithmen funktional umsetzen und praktisch anwenden lernen

  - Probleme analysieren und Lösungsstrategien entwickeln

Inhalte der Veranstaltung

- Grundlagen:
  - Funktionsdefinition, Rekursion, *Pattern matching*, Auswertung, Reduktion, Normalform, Funktionen höherer Ordnung, *Currying*, Typkorrektheit, Typinferenz
- Datentypen:
  - Standarddatentypen und -funktionen, Algebraische Datentypen, Typkonstruktoren, Typklassen, Polymorphie, *List comprehension*
- Algorithmen und Datenstrukturen:
  - Bäume, Graphen, *Queues*, zyklische Strukturen, unendliche Strukturen

Inhalte der Veranstaltung (2)

- Strukturierung und Spezifikation:
  - Module, Interfaces, abstrakte Datentypen, Signaturen und Axiome
- Theoretische Aspekte:
  - Referentielle Transparenz, Lambda-Ausdrücke, η-Kontraktion, Faltung
- Fortgeschrittene Programmierung:
  - Funktionale I/O, Monaden, Parsing, (zufallsbasiertes) Testen, domänenspezifische Sprachen, Datenbanken

Nichtlinearität der Lehrveranstaltung

- Kursverlauf nicht linear, z.B.
  - Funktionsdefinition und punktfreie Notation
  - Signaturen und Typklassen
  - Listen und ihre Konstruktion
  - I/O und Monaden
  - *List comprehension* und Listen als Monaden
  - Algebraische Datentypen und Typklassen
  - Algebraische Datentypen und *Records*
  - *Mapping* und Funktoren
  - Faltung und ihre Verallgemeinerung
  - …

Funktionale Programmierung und Abstraktion

- Was sind die drei wichtigsten Dinge beim Programmieren?
  - Abstraktion, Abstraktion und Abstraktion
- In Haskell: Abstraktion von …
  - Funktionen (*Currying*, Funktionen höherer Ordnung, Funktionen als Daten, Kombinatoren, …)
  - Typen (Typvariablen, Polymorphie, Typklassen, …)
  - Datenspeicherung (algebraische Datentypen)
  - Berechnungsmustern (Klassen `Foldable`, `Functor`, `Applicative`, `Monad`, …)
  - Repräsentation von Daten (abstrakte Datentypen)
  - Umgebung (Plattform) durch abstrakten Datentyp `IO`
  - …

Ausblick

Die Zukunft funktionaler Sprachen

- Funktionale Programmierung hat eine lange Tradition (LISP: 1958)

- Lange Zeit eher Nischendasein
  - zunächst imperative, später objektorientierte Programmierung
  - funktionale Programmierung vergleichbar mit logikorientierter Programmierung

- Schon immer wichtig in Informatik-Ausbildung
  - Entwicklung informatischer Konzepte
  - Abstraktion…

Die Zukunft funktionaler Sprachen (2)

- Immer mehr Konzepte in traditionell nicht-funktionalen Sprachen
  - Funktionale Syntax in objektorientierter Programmierung
    - Dot-Notation `object.method()`
  - funktionale Komposition von Methoden
    - `x.one().two().three()` – *read left to right*
  - Funktionen höherer Ordnung (z.B. Filterung von Daten)
  - anonyme Funktionen (Lambda-Ausdrücke)

- … bis hin zu expliziter Spracherweiterung Richtung funktionale Sprache
  - Beispiele: Scala, Clojure, O'Caml, F#, …

Haskell in der Industrie

- siehe https://wiki.haskell.org/Haskell_in_industry
  - listet 100+ Firmen, die u.a. Haskell verwenden
  - AT&T, Credit Suisse, Deutsche Bank, Ericsson, Facebook, Google, Intel, Microsoft, NVIDIA, The New York Times, …

- Vielfältige Anwendungen…
  - Network Security, Data Transformation, Domänenspezifische Sprachen, Management virtueller Servercluster, Multicore Parallel Programming, Anti-Spam, Schaltkreisentwicklung, Verschlüsselung, …

Scheinkriterien und E-Klausur

Scheinkriterien

- Elektronische Klausur (“Programmierübung”)
- Mindestens 50% der erreichbaren Punkte aus den Übungsblättern und 50% in der E-Klausur
- Note gemittelt aus Übungsblättern und E-Klausur
- Notenspiegel (in % aller Punkte)

Pkt. %	Note	Pkt. %	Note	Pkt. %	Note	Pkt. %	Note
		89,5-85	1,7	74,5-70	2,7	59,5-55	3,7
≥ 95	1,0	84,5-80	2,0	69,5-65	3,0	54,5-50	4,0
94,5-90	1,3	79,5-75	2,3	64,5-60	3,3	49,5-0	n.b.

E-Klausur

- Ähnlich wie Probeklausur: MC Fragen und Programmieraufgaben
- Insgesamt 25 Punkte (5+20), 90 Minuten Bearbeitungszeit
- 4-5 Aufgaben im Programmierteil, je 3 bis 5 Punkte, z.T. mit Teilaufgaben

- Instruktionen und Beispiele für Programmieraufgaben…

Programmieraufgaben insgesamt 20 Punkte

Lösen Sie die folgenden Programmieraufgaben. Im Ordner `Aufgaben` finden Sie neben der Datei `Tests.hs` fünf Haskell-Dateien, welche jeweils den Rumpf mit Typsignatur enthalten. Ergänzen Sie diese mit einer Implementation, und beachten Sie bitte:

- Verändern Sie die vorgegeben Typsignaturen nicht.
- Benennen Sie die Dateien nicht um, und legen Sie keine weiteren Dateien an.
- Achten Sie darauf, dass sich Ihre Abgabe übersetzen läßt; nicht übersetzbare Lösungen werden mit jeweils 0 Punkten bewertet.

Weitere nützliche Hinweise

- Sie können Visual Studio Code zum Bearbeiten der Programmieraufgabe nutzen; es ist mit Erweiterungen für Haskell vorkonfiguriert, und wird durch das Icon auf dem Desktop gestartet.
- Zum Übersetzen benutzen Sie den `ghci`. Auf dem Desktop finden Sie ein Icon, welches den `ghci` startet.
- Die Datei `Tests.hs` enthält Beispiele aus der Aufgabenstellung als Unit-Tests. So können Sie schnell prüfen, ob Ihre Lösung plausibel ist. Natürlich sind diese Tests nur notwendig, aber nicht hinreichend für eine vollständige Lösung der Aufgaben.

  Zum Ausführen der Tests laden Sie bitte `Tests.hs` in den `ghci`, und rufen die Funktion `run` auf.
- Sie finden auf dem Desktop Icons zur Dokumentation des `ghc` (insbesondere der Standard-Bücherei).

Beispiel Programmieraufgaben

- Definieren Sie eine Funktion

```
ostern :: String -> Int
```

  die zählt, wie oft in einer Zeichenkette die Buchstabenfolge "ei" enthalten ist.

- Beispiel:

```
> ostern "ei, ei, oh, eiaiei"
4
```

Beispiel Programmieraufgaben

- Definieren Sie eine Funktion concatSnd, welche eine Liste aus Paaren von Elementen und Listen auf ein Liste von Paaren von Elementen abbildet.
- Beispiele:

```
> concatSnd [(True, "ab"), (False, "foo")]
[(True,'a'),(True,'b'),(False,'f'),(False,'o'),(False,'o')]
```

```
> concatSnd [(0, [5, 6]), (1, [7, 8, 9])]
[(0,5),(0,6),(1,7),(1,8),(1,9)]
```

Beispiel Programmieraufgaben

- Definieren Sie eine Funktion

```
subseqs :: [a]-> [[a]]
```

  welche die nichtleeren Teillisten einer gegebenen Liste berechnet.

- Beispiel:

```
> subseqs "pi3"
["p","pi","pi3","i","i3","3"]
```

Beispiel Verständnisfrage

- Betrachten Sie folgende Werte:

```
Otto
Karl Otto "Heinz"
Karl (Karl Otto [1,7]) "17"
```

- Für welche der folgenden Typdeklarationen sind diese Werte wohlgetypt?

```
data T a = Otto | Karl (T a) [a]
```

```
data T a b = Otto | Karl a b
```

```
data T a = Otto | Karl (T a) String
```

```
data T a b = Otto a | Karl b [a]
```

Beispiel Verständnisfrage

- Betrachten Sie folgende Werte:

```
Otto
Karl Otto "Heinz"
Karl (Karl Otto [1,7]) "17"
```

- Für welche der folgenden Typdeklarationen sind diese Werte wohlgetypt?

```
data T a = Otto | Karl (T a) [a]
```

```
data T a b = Otto | Karl a b
```

```
data T a = Otto | Karl (T a) String
```

```
data T a b = Otto a | Karl b [a]
```

Beispiel Verständnisfrage

- Gegeben sei folgende Funktionsdefinition:

```
f :: a -> [a] -> [a]
f a (b:bs) = b : f a bs
f a []     = [a]
```

- Welche der folgenden Definitionen sind äquivalent?

```
f1 x xs = foldr (:) xs [x]
```

```
f2 = (flip (++) . (:[]))
```

```
f3 x xs = foldl (flip (:)) xs x
```

```
f4 a = (++ [a])
```

Beispiel Verständnisfrage

- Gegeben sei folgende Funktionsdefinition:

```
f :: a -> [a] -> [a]
f a (b:bs) = b : f a bs
f a []     = [a]
```

- Welche der folgenden Definitionen sind äquivalent?

```
f1 x xs = foldr (:) xs [x]
```

```
f2 = (flip (++) . (:[]))
```

```
f3 x xs = foldl (flip (:)) xs x
```

```
f4 a = (++ [a])
```

PI3 Tutoren gesucht…

- Für WS 2022/23 (und darüber hinaus): meldet Euch bei Christoph Lüth (oder bei mir)

- **Letzte Fragen???**


Good
Bye!