Funktionale Programmierung (in Clojure)

Einführung

Burkhardt Renz

Fachbereich MNI
Technische Hochschule Mittelhessen

Wintersemester 2014/15

Übersicht

- Paradigmen von Programmiersprachen
  - Paradigmen von Programmiersprachen
  - Das imperative Paradigma
  - Das funktionale Paradigma
  - Das relationale Paradigma
  - Das objektorientierte Paradigma
- Arten funktionaler Sprachen
- Eigenschaften funktionaler Programmierung
- Literatur et al.

Paradigmen von Programmiersprachen

Paradigma kommt aus dem Griechischen und bedeutet „Lehrbeispiel“, „Vorbild“, „Vorzeigestück“

Verwendet oft im Sinne von „grundlegende Denkweise“, „Leitbild“, „Weltanschauung“

Bezüglich Programmiersprachen:

- wie sieht man den Prozess einer Berechnung?
- wie definiert man eine Berechnung?
- wie definiert und verwendet man Daten?

Wichtigste Paradigmen

- imperative Programmierung
- funktionale Programmierung
- logische, bzw. relationale Programmierung
- objektorientierte Programmierung

Das imperative Paradigma

- Von-Neumann-Architektur:
  Speicher – Steuerwerk – Rechenwerk
- Detaillierte Beschreibung, wie der Speicher sukzessive modifiziert werden soll $\rightarrow$ *imperativ* (lat. imperare = befehlen)
- Programmierung durch Veränderung des Speicherzustands mittels Wertzuweisung
- Maschinenorientierte Datenstrukturen
- Beispiele: FORTRAN, Pascal, C – Summieren in Java

Das funktionale Paradigma

- Programm $\leftrightarrow$ Berechnung einer Funktion
- basierend auf Alonzo Churchs λ-Kalkül
- Funktionen sind „Bürger erster Klasse“ – können selbst Parameter oder Rückgabewert einer Funktion sein
- Konstruktion von Funktionen mittels anderer Funktionen: *Funktionen höheren Typs*
- *Referenzielle Transparenz*: Ein Ausdruck entspricht seinem Wert und hat immer denselben Wert
- Werte (*immutable objects*) statt zustandsbehaftete Objekte
- Beispiele: Lisp, ML, Haskell, Scheme, Erlang, Clojure – Summieren in Clojure
- Auch eingebettet in objektorientierte Sprachen: JavaScript, C#, Java, Scala, Python, ...

Das relationale Paradigma

- Programmausführung $\leftrightarrow$ Ableitungsprozess, Beweis
- Programm spezifiziert die Fragestellung, nicht einen Weg zur Lösung
- Keine lineare Berechnung: System sucht eine Bindung logischer Variablen, die die in der Spezifikation gestellten Bedingungen erfüllen
- Mehrere Lösung (sogar unendlich viele) möglich
- Beispiele: PROLOG, MiniKanren, core.logic in Clojure – siehe Relation für Addition

Das objektorientierte Paradigma

- Programm zur Laufzeit $\leftrightarrow$ Geflecht von Objekten, die durch Nachrichten interagieren
- Objekte kapseln ihren Zustand, der durch Methoden/Nachrichten geändert wird
- Klassen sind Vorlagen für Objekte, die zur Laufzeit instanziiert werden
- Vererbung von Schnittstellen und Verhalten
- Polymorphismus
- Beispiel: Smalltalk, C++, Java, C# – Stack in Java vs. Stack in Clojure

Übersicht

- Paradigmen von Programmiersprachen
- Arten funktionaler Sprachen
  - Auswertungsart
  - Syntax
  - Typisierung
- Eigenschaften funktionaler Programmierung
- Literatur et al.

Auswertungsart

Strikte/applikative Auswertung

Erst werden die Parameter evaluiert und *dann* wird deren Wert an die Funktion übergeben (*eager evaluation*) z.B. Scheme, Clojure

Verzögerte/normale Auswertung

Auswertung der Parameter erst dann, wenn sie tatsächlich benötigt werden (*lazy evaluation*) z.B. Haskell

Syntax

Infix-Notation

```
(2 + 3) * square(3)
```

z.B. Scala

Präfix-Notation

```
(* (+ 2 3) (square 3))
```

z.B. Lisp, Clojure

Typisierung

Dynamische Typisierung

Typ eines Werts wird zur Laufzeit bestimmt
Datenstrukturen können typischerweise Werte verschiedener Typen enthalten
(basiert auf klassischem λ-Kalkül)
z.B. Lisp, Clojure (jedoch: Clojure kompiliert zu Java)

Statische/strikte Typisierung

Typinferenz zur Kompilierzeit
Definition von Funktionen für bestimmte Typen
Datenstrukturen haben typischerweise Werte desselben Typs
(basiert auf λ-Kalkül mit Typen)
z.B. Haskell, F#, ML

Übersicht

- Paradigmen von Programmiersprachen
- Arten funktionaler Sprachen
- Eigenschaften funktionaler Programmierung
- Literatur et al.

Eigenschaften funktionaler Programmierung

1. Mächtige Konstrukte $\rightarrow$ kompakter Code
2. Interaktive Entwicklung in der REPL
3. Modularisierung
4. Gute Wiederverwendbarkeit reiner Funktionen
5. Verifizierbarkeit
6. Gut geeignet für Nebenläufigkeit, inhärent thread-sicher

Übersicht

- Paradigmen von Programmiersprachen
- Arten funktionaler Sprachen
- Eigenschaften funktionaler Programmierung
- Literatur et al.
  - Literatur
  - Internet-Quellen

Literatur


Gerald Jay Sussman, Harold Abelson
Structure and Interpretation of Computer Programs
MIT Press 1996.
`http://mitpress.mit.edu/sicp/full-text/book/book.html`


Stuart Halloway, Aaron Bedra:
Programming Clojure, Second Edition
Pragmatic Programmers, 2012.


Stefan Kamphausen, Tim Oliver Kaiser:
Clojure
dpunkt.verlag, 2010.

Literatur

Chas Emerick, Brian Carper, Christophe Grand
Clojure Programming
O'Reilly 2012.

Dominikus Herzberg:
Funktionale Programmierung mit Clojure
`http://denkspuren.blogspot.de/2013/04/freies-clojure-buch-funktionale.html`

Internet-Quellen

Rich Hickey and the Clojure Community:
Clojure - home
http://clojure.org/

The Clojure Community:
Clojure - Google Groups
https://groups.google.com/forum/#!forum/clojure

The Clojure Community:
Clojure Documentation
http://clojure-doc.org/

Hal Abelson, Gerald Jay Sussman:
Structure and Interpretation of Computer Programs
Video Lectures (1986)
https://groups.csail.mit.edu/mac/classes/6.001/abelson-sussman-lectures/