

Fortgeschrittene Techniken der Funktionalen Programmierung
Vorlesung vom 02.02.10:
Rückblick & Ausblick

Christoph Lüth, Dennis Walter

Universität Bremen

Wintersemester 2009/10

Fahrplan

- ▶ Teil I: Monaden und fortgeschrittene Typen
- ▶ Teil II: Fortgeschrittene Datenstrukturen
- ▶ Teil III: Nebenläufigkeit
- ▶ Teil IV: The Future of Programming
 - ▶ Fancy Types
 - ▶ Domain-Specific Languages
 - ▶ The Next Big Thing: F#, Scala
 - ▶ Rückblick, Ausblick

Monaden

- ▶ Monade: Einbettung **imperativer** Konzepte in **funktionaler** Sprache
- ▶ Zustandsmonade $ST\ s\ a = s \rightarrow (a, s)$
 - ▶ Lesemonade $Read\ r\ a = r \rightarrow a$
 - ▶ Schreibmonade $Writer\ w\ a = (a, [w])$
- ▶ Ausnahmemonade $Error\ e\ a = Left\ e \mid Right\ a$
- ▶ Nichtdeterminismus $ND\ a = [a]$
- ▶ IO-Aktionen sind keine Magie
- ▶ Monadentransformer: **Kombination von Monaden**

Fortgeschrittene Datenstrukturen

- ▶ Der Zipper: `in-place-update` für funktionale Sprachen
- ▶ Unendliche Datenstrukturen
- ▶ Graphen: induktiv (erzeugt), aber nicht frei (kein **data**)

Nebenläufigkeit

- ▶ Leichtgewichtige Threads in Haskell
- ▶ Kleines Basiskonzept (threads, MVar)
- ▶ Darüber Abstraktionen: Semaphoren, Monitore, Kanäle, Aktoren, ...
- ▶ Software Transactional Memory
 - ▶ Fundamental anderes Synchronisationskonzept
 - ▶ Auch in anderen Programmiersprachen interessant!

Fancy Types

- ▶ **Hindley-Milner-Polymorphie**: Typcheck ist **exponentiell**
- ▶ Abstract Types have Existential Type
- ▶ Polymorphie höherer Ordnung
 - ▶ Beispiel Zustandsübergangsmonade:
 $\text{runST} :: \text{forall } a. (\text{forall } s. \text{ST } s \ a) \rightarrow a$
- ▶ Typentheorie — der mathematische Hintergrund
- ▶ Lektüre: Cardelli/Wegner, *n Understanding Types, Data Abstraction and Polymorphism* (Klassiker!)

Domain-Specific Languages

- ▶ DSL: Programmiersprache für **spezifische Problemstellung**
- ▶ **Beispiele**: XSLT, Postscript, $\text{\LaTeX}$, VHDL, UML, ...
- ▶ In Haskell **eingebettet**:
 - ▶ **Tief**: Programme als Datentyp
 - ▶ **Flach**: Programme als Haskell-Funktionen
 - ▶ Beispiele: HXT, Lava

The Next Big Thing

- ▶ Scala, F#
- ▶ These: **Funktional** gut für **nebenläufige** Programme
- ▶ Weitere Aspekte:
 - ▶ Polytypische Programmierung
 - ▶ Reflektion
 - ▶ Generic Programming
 - ▶ Template Haskell
 - ▶ Beispiel: XML Serializing

Nicht Behandelt

- ▶ **Implementationsaspekte** — wie übersetzt man funktionale Sprachen?
 - ▶ Meist mehrere Phasen (volle Sprache $\rightarrow$ einfache Sprache $\rightarrow$ ausführbare Sprache)
 - ▶ Abstrakte **Ausführungsmaschinen** (CAM, spineless tagless G-Machine)
- ▶ **Sprachinteroperabilität**
 - ▶ From Heaven to Hell and back: call-out, call-in, das **Foreign Function Interface**
 - ▶ component frameworks, eg. CORBA, Java, .Net
- ▶ **Semantik** — was bedeutet das alles?
 - ▶ Fixpunkte — Theorie der kontinuierlichen Funktionen
 - ▶ Datentypen — Domänentheorie (CPOs, $D = D \rightarrow D$)