Vorlesung vom 21.10.2002: Einführung

Personal

- Vorlesung: Christoph Lüth <cxl>
  MZH 8120, Tel. 7585
- Stud. Tutoren: Felix Beckwermert <foetus>
  Michael Skibbe <mskibbe>
  Dennis Walter <dw>
  Rafael Trautmann <pirate>
  Thomas Meyer <mclee>
- Website: www.informatik.uni-bremen.de/~cxl/pi3.
- Newsgroup: fb3.lv.pi3.

Termine

- Vorlesung:

  Mo 10-12, kleiner Hörsaal („Keksdose“)

- Tutorien:

Di	10-12	MZH 1380	Dennis Walter
Mi	15-17	MZH 6240	Rafael Trautmann
Mi	15-17	MZH 7250	Thomas Meyer
Mi	17-19	MZH 6240	Felix Beckwermert
Mi	17-19	MZH 1380	Michael Skibbe

Übungsbetrieb

- Ausgabe der Übungsblätter über die Website Montag nachmittag
- Besprechung der Übungsblätter in den Tutorien;
- Bearbeitungszeit zwei Wochen ab Tutorium,

  Abgabe im Tutorium;
- Voraussichtlich sechs Übungsblätter.

Inhalt der Veranstaltung

- Deklarative und funktionale Programmierung
  - Betonung auf Konzepten und Methodik
- Bis Weihnachten: Grundlagen
  - Funktionen, Typen, Funktionen höherer Ordnung, Polymorphie
- Nach Weihnachten: Ausweitung und Anwendung
  - Prolog und Logik; Nebenläufigkeit/GUIs; Grafik und Animation
- Lektüre:
  Simon Thompson: *Haskell — The Craft of Functional Programming* (Addison-Wesley, 1999)

Scheinrelevanz

„Der in der DPO'93 aufgeführte prüfungsrelevante PI3-Schein kann nicht nur über das SWP sondern alternativ auch über PI3 abgedeckt werden. Die in der DPO zusätzlich aufgeführte Forderung der erfolgreichen Teilnahme am SWP bleibt davon unberührt."

Mit anderen Worten:

- Entweder prüfungsrelevante Studienleistung in PI3 sowie erfolgreiche Teilnahme an SWP
- oder Prüfungsrelevante Studienleistung in SWP

Scheinkriterien — Vorschlag:

- Ein Übungsblatt ist bearbeitet, wenn mindestens 20% der Punktzahl erreicht wurden.
- Alle Übungsblätter bearbeitet und mindestens 60% der Punktzahl erreicht.
- Individualität der Leistung wird sichergestellt durch:
  - Vorstellung einer Lösung im Tutorium
  - Beteiligung im Tutorium
  - Ggf. Prüfungsgespräch (auch auf Wunsch)

Einführung in FP

Warum funktionale Programmierung lernen?

- Abstraktion
  - Denken in Algorithmen, nicht in Programmiersprachen
- FP konzentriert sich auf wesentlichen Elemente moderner Programmierung:
  - Datenabstraktion
  - Modularisierung und Dekomposition
  - Typisierung und Spezifikation
- Blick über den Tellerrand — Blick in die Zukunft
  - Studium $\neq$ Programmierkurs— was kommt in 10 Jahren?

Referentielle Transparenz

- Programme als Funktionen

$$P : Eingabe \rightarrow Ausgabe$$

- Keine Variablen — keine Zustände
- Alle Abhängigkeiten explizit:
- Rückgabewert hängt ausschließlich von Werten der Argumente ab, nicht vom Aufrufkontext:

Referentielle Transparenz

Geschichtliches

- Grundlagen 1920/30
  - Kombinatorlogik und λ-Kalkül (Schönfinkel, Curry, Church)
- Erste Sprachen 1960
  - LISP (McCarthy), ISWIM (Landin)
- Weitere Sprachen 1970– 80
  - FP (Backus); ML (Milner, Gordon), später SML und CAML; Hope (Burstall); Miranda (Turner)
- 1990: Haskell als Standardsprache

Funktionen als Programme

Programmieren durch Rechnen mit Symbolen:

$$\begin{aligned} 5 * (7 - 3) + 4 * 3 &= 5 * 4 + 12 \\ &= 20 + 12 \\ &= 32 \end{aligned}$$

Benutzt Gleichheiten ($7 - 3 = 4$ etc.), die durch Definition von $+$, $*$, $-$, . . . gelten.

Programmieren mit Funktionen

- Programme werden durch Gleichungen definiert:

```
inc x = x+ 1
addDouble x y = 2*(x+ y)
```

- Auswertung durch Reduktion von Ausdrücken:

```
addDouble 6 4 ⇝2*(6+ 4) ⇝20
```

- Nichtreduzierbare Ausdrücke sind Werte
  - Zahlen, Zeichenketten, Wahrheitswerte, . . .

Auswertungsstrategie

- Von außen nach innen, links nach rechts.

  ```
  inc (addDouble (inc 3) 4)
  ```

  ⇝`(addDouble (inc 3) 4)+ 1`

  ⇝`2*(inc 3+ 4)+ 1`

  ⇝`2*(3+ 1+ 4)+ 1`

  ⇝`2*8+1` ⇝`17`

- Entspricht call-by-need (verzögerte Auswertung)
  - Argumentwerte werden erst ausgewertet, wenn sie benötigt werden.

Nichtnumerische Werte

- Rechnen mit Zeichenketten:

```
repeat s = s ++ s

repeat (repeat "hallo ")
⇝repeat "hallo"++ repeat "hello"
⇝("hallo "++ "hallo ")++("hallo "++ "hallo ")
⇝"hallo hallo hallo hallo"
```

Typisierung

Typen unterscheiden Arten von Ausdrücken

- Basistypen
- strukturierte Typen (Listen, Tupel, etc)

Wozu Typen?

- Typüberprüfung während Übersetzung erspart Laufzeitfehler
- Programmsicherheit

Übersicht: Typen in Haskell

Ganze Zahlen	`Int`	`0 94 -45`
Fließkomma	`Double`	`3.0 3.141592`
Zeichen	`Char`	`'a' 'x' '\034' '\n'`
Zeichenketten	`String`	`"yuck" "hi\nho\"\n"`
Wahrheitswerte	`Bool`	`True False`
Listen	`[a]`	`[6, 9, 20]`
		`["oh", "dear"]`
Tupel	`(a, b)`	`(1, 'a') ('a', 4)`
Funktionen	`a-> b`	

Definition von Funktionen

- Zwei wesentliche Konstrukte:
  - Fallunterscheidung
  - Rekursion
- Beispiel:

```
fac :: Int-> Int
fac n = if n == 0 then 1
        else n * (fac (n-1))
```

- Auswertung kann divergieren!

Haskell in Aktion: `hugs`

- `hugs` ist ein Haskell-Interpreter
  - Klein, schnelle Übersetzung, gemächliche Ausführung.
- Funktionsweise:
  - `hugs` liest Definitionen (Programme, Typen, . . .) aus Datei (Skript)
  - Kommandozeilenmodus: Reduktion von Ausdrücken
  - Keine Definitionen in der Kommandozeile
  - Hugs in Aktion.

Zusammenfassung

- Haskell ist eine funktionale Programmiersprache
- Programme sind Funktionen, definiert durch Gleichungen
  - Referentielle Transparenz — keine Zustände oder Variablen
- Ausführung durch Reduktion von Ausdrücken
- Typisierung:
  - Basistypen: Zahlen, Zeichen(ketten), Wahrheitswerte
  - Strukturierte Typen: Listen, Tupel
  - Jede Funktion `f` hat eine Signatur `f :: a-> b`