Vorlesung vom 25.10.2004: Einführung

Personal

- Vorlesung: Christoph Lüth `<cxl>`, MZH 8050, Tel. 7585
- Tutoren: Shi Hui `<shi>`
  Diedrich Wolter `<dwolter>`
  Christian Maeder `<maeder>`
  Klaus Lüttich `<luettich>`
  Kai-Florian Richter `<richter>`
  Dennis Walter `<dw>`
  Wolfgang Machert `<wmachert>`
  Hennes Märtins `<maertins>`
- Fragestunde: Berthold Hoffmann `<hof>`
- Website: `www.informatik.uni-bremen.de/~cxl/lehre/pi3.ws04`.
- Newsgroup: `fb3.lv.pi3`.

Termine

- Vorlesung: Mo 10 – 12, kleiner Hörsaal („Keksdose“)
- Tutorien:

Di	8 – 10	MZH 1380	Wolfgang Machert
	17 – 19	MZH 1380	Diedrich Wolter
Mi	8 – 10	MZH 6240	Hennes Märtins
		MZH 7250	Shi Hui
	13 – 15	MZH 7230	Christian Maeder
		MZH 6240	Klaus Lüttich
		MZH 1380	Dennis Walter
Mi	17 – 19	MZH 1380	Kai-Florian Richter

- Fragestunde (FAQ): Mi 9 – 11 Berthold Hoffmann (MZH 8130)

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
  - z.B: Nebenläufigkeit; Grafik und Animation
- Lektüre:
  Simon Thompson: Haskell — The Craft of Functional Programming
  (Addison-Wesley, 1999)

Scheinkriterien — Vorschlag:

- Erfolgreiche Bearbeitung eines Übungsblattes: $\geq 40\%$
- Alle Übungsblätter erfolgreich zu bearbeiten.
- Es gibt ein Bonusübungsblatt, um Ausfälle zu kompensieren.
- Individualität der Leistung wird sichergestellt durch:
  - Vorstellung einer Lösung im Tutorium
  - Beteiligung im Tutorium
  - Ggf. Prüfungsgespräch (auch auf Wunsch)

Spielregeln

- PI1- und PI2- Schein sind Voraussetzung.
- Quellen angeben bei
  - Gruppenübergreifender Zusammenarbeit;
  - Internetrecherche, Literatur, etc.
- Erster Täuschungsversuch:
  - Null Punkte
  - Fachgespräch.
- Zweiter Täuschungsversuch: Kein Schein.
- Deadline verpaßt?
  - Vorher ankündigen, sonst null Punkte.

Einführung in die Funktionale Progammierung

Warum funktionale Programmierung (FP) 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?

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

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

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` $\rightsquigarrow$ `2*(6+ 4)` $\rightsquigarrow$ `20`

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

Auswertungsstrategie

- Von außen nach innen, links nach rechts.

  `inc (addDouble (inc 3) 4)`

  $\rightsquigarrow$`(addDouble (inc 3) 4)+ 1`

  $\rightsquigarrow$`2*(inc 3+ 4)+ 1`

  $\rightsquigarrow$`2*(3+ 1+ 4)+ 1`

  $\rightsquigarrow$`2*8+1` $\rightsquigarrow$`17`

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

Nichtnumerische Werte

Rechnen mit Zeichenketten:

```
repeater s = s ++ s
```

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