Funktionales Programmieren
(Praktische Informatik 3)

Berthold Hoffmann
Studiengang Informatik
Universität Bremen

Winter 03/04

Einführung
(Vorlesung vom 20.10.2003)

Organisation

Personal

- Vorlesung: Berthold Hoffmann <hof>
  MZH 8130, Tel. 2450
- Tutoren: Felix Beckwermert <foetus>
  Klaus Lüttich <luettich>
  Christian Maeder <maeder>
  Franck Ngueuleu <nngf>
  Pascal Schmidt <pharao90>
  Dennis Walter <dw>
- WWW: www.tzi.de/agbkb/lehre/ws03-04/pi3.
- Newsgroup: fb3.lv.pi3.

Termine

Vorlesung: Mo 10-12, kleiner Hörsaal (HS 1010)

Tutorien:	Di	08-10	MZH 1380	Berthold Hoffmann
	Di	17-19	GW2 B2880	Franck Ngueuleu
	Mi	08-10	MZH 6240	Dennis Walter
	Mi	08-10	MZH 7250	Felix Beckwermert
	Mi	13-15	MZH 6240	Klaus Lüttich
	Mi	13-15	MZH 7230	Christian Maeder
	Mi	13-15	MZH 7250	Pascal Schmidt
	Mi	17-19	MZH 1380	Felix Beckwermert

Übungsbetrieb

- Ausgabe der Aufgabenblätter über die Website Montag nachmittag
- Besprechung der Aufgabenblätter in den Tutorien
- Bearbeitungszeit zwei Wochen ab Tutorium
- Abgabe
  - Text: im Tutorium
  - Code: per Email
- Voraussichtlich sechs Aufgabenblätter.

Inhalt der Veranstaltung

- Deklaratives und funktionales Programmieren
  - Schwerpunkt: Konzepten und Methodik
- 2003: Grundlagen
  - Funktionen, Typen, Funktionen höherer Ordnung, Polymorphie
- 2004: Ausweitung und Logik oder Anwendung
  - Prolog und Logik oder Nebenläufigkeit/GUI/Grafik/Animation
- Lektüre:
  Simon Thompson: *Haskell — The Craft of Functional Programming* (2/e, Addison-Wesley, 1999)

Scheinrelevanz

Vorläufige PO (2002):

- Pflicht zum Diplom, Wahlpflicht zum Bachelor

Alte DPO (1993):

- Entweder prüfungsrelevante Studienleistung in PI3
  sowie erfolgreiche Teilnahme am SWP
- oder (nur) prüfungsrelevante Studienleistung in SWP

Scheinkriterien — Vorschlag:

- Ein Aufgabenblatt ist bearbeitet, wenn mindestens 30% der Punktzahl erreicht wurde.
- Einen Schein bekommt, wer alle Aufgabenblätter bearbeitet und mindestens 60% der Punktzahl erreicht.
- Individualität der Leistung wird sichergestellt durch:
  - Beteiligung im Tutorium (z.B. Vorstellung einer Lösung)
  - Ggf. durch ein Fachgespräch

Einführung in Funktionales Programmieren

Funktionales Programmieren — WOZU?

- Alle Programmierstile kennen lernen
  - PI1&2: imperativ und objektorientiert
  - PI3: funktional (und vielleicht auch logisch)
- Zukunftssicher studieren!
  - Gibt es ein Programmieren nach Ih und OhOh?

Funktionales Programmieren — WOZU?

- Alle Programmierstile kennen lernen
  - PI1&2: imperativ und objektorientiert
  - PI3: funktional (und vielleicht auch logisch)
- Zukunftssicher studieren!
  - Gibt es ein Programmieren nach Ih und OhOh?
  - Und was ist in 20 Jahren?

Funktionales Programmieren — WOZU?

- Alle Programmierstile kennen lernen
  - PI1&2: imperativ und objektorientiert
  - PI3: funktional (und vielleicht auch logisch)
- Zukunftssicher studieren!
  - Gibt es ein Programmieren nach Ih und OhOh?
  - Und was ist in 20 Jahren?
  - FUN! FUN! FUN!

Typisch funktional

- Innovative Elemente des funktionalen Programmierens:
  - rekursive Datenstrukturen
  - polymorphe Typinferenz (Herleitung von Typen)
  - Funktionen sind Werte erster Klasse
  - Modularisierung
  - Datenabstraktion
  - Verifikation und Spezifikation
- Motto: What, not How!

Der Kern des funktionalen Programmierens

- Programme sind Funktionen

$$P : Eingabe \rightarrow Ausgabe$$

- Abstraktion vom Speicher mit Variablen und Zustände
- Alle Abhängigkeiten sind explizit:
  - Ergebniswert hängt ausschließlich von Werten der Argumente ab, nicht vom Kontext des Aufrufs

Referenzielle Transparenz

Geschichtliches

- Grundlagen 1920–30
  - Kombinatorlogik und λ-Kalkül (Schönfinkel, Curry, Church)
- Erste Sprachen 1960
  - LISP(McCarthy, 1960), später: Scheme(Sussman, 1975), Common LISP(Steele, 1984) ISWIM (Landin, 1966)
- Weitere Sprachen 1970–80
  - FP(Backus, 1978); ML(Milner, Gordon, 1979), später Standard ML, CAML; Hope(Burstall, 1980); Miranda(Turner, 1985); Erlang(Ericsson, 1987); Concurrent Clean(Plasmeijer, 1988)
- 1990: Haskell

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 (vorgegebene) 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
```

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)
```

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
```

- Nicht reduzierbare Ausdrücke sind Werte
  - Zahlen, Zeichenketten, Wahrheitswerte, . . .

Auswertungsstrategie

- Von außen nach innen, links nach rechts.

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

$\rightsquigarrow$

Auswertungsstrategie

- Von außen nach innen, links nach rechts.

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

```
⇝(addDouble (inc 3) 4)+ 1
```

⇝

Auswertungsstrategie

- Von außen nach innen, links nach rechts.

```
inc (addDouble (inc 3) 4)
⇝(addDouble (inc 3) 4)+ 1
⇝2*(inc 3+ 4)+ 1
⇝
```

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
⇝
```

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:

```
  rep s = s ++ s

rep (rep "Hallo! ")
```

$\rightsquigarrow$

Nichtnumerische Werte

- Rechnen mit Zeichenketten:

```
  rep s = s ++ s

rep (rep "Hallo! ")

⇝rep "Hallo! "++ rep "Hallo! "

⇝
```

Nichtnumerische Werte

- Rechnen mit Zeichenketten:

```
  rep s = s ++ s

rep (rep "Hallo! ")
⇝rep "Hallo! "++ rep "Hallo! "
⇝("Hallo! "++ "Hallo! ")++("Hallo! "++ "Hallo! ")
⇝
```

Nichtnumerische Werte

- Rechnen mit Zeichenketten:

```
  rep s = s ++ s

rep (rep "Hallo! ")
⇝rep "Hallo! "++ rep "Hallo! "
⇝("Hallo! "++ "Hallo! ")++("Hallo! "++ "Hallo! ")
⇝"Hallo! Hallo! Hallo! Hallo! "
```

Typisierung

Typen unterscheiden Arten von Werten und Ausdrücken

- Basisdatentypen
- Strukturierte Datentypen (Liste, Tupel, etc.)

Wozu Typen?

- Typüberprüfung während der Übersetzung erspart Fehler zur Laufzeit
- Das erhöht die Programmsicherheit

Übersicht: Typen in Haskell

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

Definition von Funktionen

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

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

- Die Auswertung kann divergieren!

Definition von Funktionen

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

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

- Die Auswertung kann divergieren! Weshalb zum Beispiel?

Haskell in Aktion: `hugs` und `ghci`

- `hugs` ind `ghci` sind Haskell-Interpreter
  - Klein, schnelles Einlesen, gemächliche Ausführung.
- Funktionsweise:
  - `hugs` und `ghci` lesen Definitionen aus einer `.hs`-Datei (module)
  - Kommandozeilenmodus: Reduktion von Ausdrücken
  - Keine Definitionen in der Kommandozeile
  - `hugs` und `ghci` in Aktion.

Zusammenfassung

- Haskell ist eine funktionale Programmiersprache.
- Programme sind Funktionen, definiert durch Gleichungen.
  - Referenzielle Transparenz — keine Zustände oder Variablen
- Ausführung durch Reduktion von Ausdrücken
- Typisierung:
  - Basisdatentypen: Zahlen, Zeichen, Zeichenketten, Wahrheitswerte
  - Strukturierte Datentypen: Listen, Tupel
  - Jede Funktion `f` hat eine Signatur `f :: a-> b`