Einleitung: Funktionales Programmieren

in Haskell

Christian Höner zu Siederdissen
christian.hoener.zu.siederdissen@uni-jena.de

Theoretische Bioinformatik, Bioinformatik Uni Jena

19. Okt. 2023

Bürokratie Zuerst

- Einführungsveranstaltung, Themenwünsche sind gerne gesehen
- VL 90 Minuten, kombiniert mit kleinen Übungen
- “Hausaufgaben” als Brücke zur nächsten VL
- Prüfung: mündlich: Themen der VL, funktionale Algorithmen und Datenstrukturen
- Von Vorteil ist: Notebook / Rechner um Code direkt ausprobieren zu können
- bitte bei Fragen direkt melden

Raumwechsel!

ab *Donnerstag, 2023-10-26* findet die Vorlesung hier statt:
Jentower, Leutragraben 1, 07743 Jena
Raum: 08N04

- den Jentower betreten
- die Fahrstühle finden, sie befinden sich neben der Rezeption im Erdgeschoss
- vor den Fahrstühlen die "8" drücken
- ein Fahrstuhl

  $$A \ldots F$$

  wird angezeigt, diesen betreten und im 8. Stock aussteigen
- dort gibt es zwei Glastüren, eine davon werde ich euch auf lassen
- den Raum 08N04 finden (nach der Glastür gleich links am Gang)

Tools, Literatur, etc

- ein installierter GHC: `www.haskell.org/ghc` (Version ziemlich egal)
- es gibt auch Online-Tools (nicht getestet von mir)
  `play.haskell.org`, (hat keine "echte" Speichern/Laden Funktion)
- Graham Hutton, Programming in Haskell
- `http://learnyouahaskell.com/` (free!)

Vorlesungsablauf

- Ziel: Bearbeitung mehrerer “interessanter” Probleme
- Wir verfeinern jeweils erste Lösungen um funktionale Konzepte zu verstehen
- und funktionale Techniken kennen zu lernen

"High-level" weniger Code schreiben

- Funktionen erster Klasse
- Typinferenz
- Pattern Matching
- Exceptions
- Continuations

Abstraktion

- Typpolymorphie
- Typklassen
- Monaden

Laufzeitumgebung nützliche Features

- Speicherverwaltung, Garbage collection
- Concurrency
- Multithreading
- Software Transactional Memory

Haskell ist (laut Wikipedia [1]) eine Allzweckprogrammiersprache, mit statischem Typsystem, pur, funktional, mit Typinferenz und lazy.

Allzweck: Spiele, Webserver, Textverarbeitung, Versionsmanagement, Numerik, “scientific programming”, etc.pp.

[1] https://en.wikipedia.org/wiki/Haskell

- Jeder Ausdruck, jede Funktion, jeder Datentyp, (alles!) hat bereits zur Kompilierzeit einen Typ.
- Fehler die im Typsystem erkannt werden, können also zur Laufzeit nicht mehr auftreten.
- Erlaubt es, mehr Bugs frühzeitig zu erkennen, reduziert Fehleranfälligkeit, erhöht Sicherheit, bessere Wiederverwendbarkeit von Code
- Bessere Codeoptimierung durch den Compiler: schnellere Programme

Typinferenz: Typen von Funktionen werden vom Compiler ausgerechnet und können auch (interaktiv!) abgefragt werden.

Pur

- Keine Seiteneffekte, `helloWorld` kann keine Dateien löschen.
- Es gibt keine "Mutation", kein `x=x+1`. Mutation ist ein Seiteneffekt
- Bedeutung: gegeben gleiches x liefert $f(x)$ immer das gleiche Ergebnis
- Funktionskomposition: $f(x) = x + 1$, $g(x) = x^2$ kombiniert: $h(x) = f(g(x))$
- Algebra, Equational reasoning, Optimierungen: man kann alle Vorkommen von $f(x)$ durch $x + 1$ ersetzen: $h(x) = f(g(x)) = g(x) + 1 = x^2 + 1$
- Algorithmen sind einfacher zu verstehen
- Einfacher zu parallelisieren

- damit natürlich auch keine Ein- und Ausgabe? (Details, Details, ...)

Lazy

- Jeder Ausdruck wird zum *letztmöglichen* Zeitpunkt berechnet. (Call-by-need)
- Wegen "purity": wir können Berechnungen "cachen" / memoisieren.
- Eigene Kontrollstrukturen (`if-then-else`, `for`
- unendliche Datenstrukturen: macht viele Probleme einfacher beschreibbar

Es geht los!

Noch Fragen?

Wir werden Programme konstruieren durch

- Aussuchen geeigneter Datentypen für alle Daten
- Kombination von Funktionen um die gewünschte Ausgabe zu erreichen
- "ein bischen" Beweisen das wir korrekte Programme konstruieren

Meine Idee dieses Jahr ist:

- Funktionale Programmierung an interessanten Beispielen zeigen.
- Die "Standard-Library" lernt ihr zusammen mit den Hausaufgaben.
- Einige der Probleme erlauben auch kreatives Kräftemessen: wer hat den schnelleren Algorithmus? Wer hat den kürzesten Code?

Die kleinste freie Zahl

- Gegeben: Liste mit Teilmenge der Zahlen aus $\mathbb{N} = \{0, 1, 2, \dots\}$
- Beispiel $[8, 23, 9, 0, 12]$ oder auch $[5, 3, 2, 1, 0]$
- Problem: Finde die kleinste, freie Zahl:

```
1 minFrei [8,23,9,0,12] = 1
2 minFrei [5,3,2,1,0] = 4
```

- "finde den nächsten freien Platz"
- Eingabe ist eine unsortierte Liste ohne Duplikate.

- Konstruiere einen Algorithmus fuer das Problem
- Bestimme seine Effizienz

Eine elegante Lösung

$$[5, 3, 2, 1, 0]$$

Wie würdet ihr das Problem lösen? (Allgemein, nicht in Haskell) Wie würde man das Problem mathematisch beschreiben?

Eine elegante Lösung

$$[5, 3, 2, 1, 0]$$

$$\begin{aligned} \mathsf{minFrei} &: \mathbb{P}(\mathbb{N}) \to \mathbb{N} \\ \mathsf{minFrei}(X) &= \mathsf{min}(\mathbb{N} \setminus X) \end{aligned}$$

(Die Definitionsmenge ist die Potenzmenge von $\mathbb{N}$)

Eine elegante Lösung

$$[5, 3, 2, 1, 0]$$

$$\begin{aligned} \text{minFrei} &: \mathbb{P}(\mathbb{N}) \to \mathbb{N} \\ \text{minFrei}(X) &= \min(\mathbb{N} \setminus X) \end{aligned}$$

(Die Definitionsmenge ist die Potenzmenge von $\mathbb{N}$)

```
1 import Data.List ( (\\) )
2
3 minFrei :: [Int] -> Int
4 minFrei xs = head ([0..] \\ xs)
```

GHC Interpreter starten

```
1      ghci
2      Prompt: 1 + 2  Enter
3      Prompt: 3
4
```

play.haskell.org

```
1      main :: IO ()
2      main = do
3        print "Hallo, Welt!"
4        print (1+2)
5
```

play.haskell.org braucht eine main Funktion. Viele andere Online Haskell Umgebungen auch.

Kleinste freie Zahl

```
1 import Data.List ( (\\) )
2
3 minFrei :: [Int] -> Int
4 minFrei xs = head ([0..] \\ xs)
```

- `import Data.List ( (\\) )` importiert aus dem Listenmodul die "−" Funktion für Listen
- `minFrei :: [Int] -> Int` hat den Typ: Liste von Int's nach Int
- `[5,3,2,1,0]` ist eine 5-elementige Liste
- `[0..]` ist eine unendliche Liste `[0,1,2,3,4...]`
- `as \\ bs` ist das Listenäquivalent zu $A \setminus B$
- `head :: [Int] -> Int` gibt den Kopf einer Liste zurück

Aufgaben zu nächster Woche

Lesestoff `http://learnyouahaskell.com/chapters`, "Ready, set, go!"

Suchmaschine `hoogle.haskell.org`

Fragen Schreibt Fragen zur grundlegenden Nutzung von Haskell auf und stellt diese Anfang nächster Woche.

Installation Gibt es Probleme bei der Installation von GHC? Schickt eine Email. (Wir können aber auch die Rechner im Jentower 08N04 nutzen)

Aufgabe Implementiert folgende Ausdrücke:

```haskell
gerade :: [Int]
gerade = error "alle geraden Zahlen"

laenge :: [Int] -> Int
laenge xs = error "Anzahl Elemente in xs"
```

- Funktionen haben 1 oder mehr Argumente und einen Rückgabetyp. `plus :: Int -> Int -> Int`
- Infixfunktionen: `2 + 3 == 5` oder `(+) 2 3 == 5`
- Funktionen können partiell angewandt werden: `(+2) 3 == 5`, oder auch `map (+1) [1,2,3] == [2,3,4]`
- `map :: (a->b) -> [a] -> [b]` ist eine Funktion höherer Ordnung. `map (+1) :: [Int] -> [Int]` addiert 1 zu allen Zahlen

Listen

Datentypen bestehen aus einem Typkonstruktor und ein[2] oder mehr Datenkonstruktoren.
Der Listentyp [a] ist definiert als:

```
1    data [] a = [] | a : [a]
2    -- Ein Selbstbau waere:
3    data List a = Nil | Cons a (List a)
4
```

- Listen sind Datentypen mit Typkonstruktor `data [] a`
- und Datenkonstruktoren `[]` und (:) a [a].

[2] Fragt Hoogle nicht nach `Void`

Pattern Matching

Datenstrukturen werden durch “Pattern Matching” zerlegt.
Der Kopf einer nicht-leeren Liste:

```
1 head :: [a] -> a
2 head [] = error "undefined behaviour"
3 head (x:xs) = x
4
```

Listendifferenz:

```
1 (\\) :: Eq a => [a] -> [a] -> [a]
2 xs \\ [] = xs
3 xs \\ (y:ys) = [ x | x <- xs, x /= y ] \\ ys
4
```

`Eq` a Ist ein “Constraint” und sagt das die *a* vergleichbar $(==)$, $(/=)$ sein müssen.

List-Builder Notation

Was ist das: `[ x | x <- xs, x /= y ]`

- `[ x | x <- xs, x /= y ]`
- nimmt alle x aus der Liste xs
- Die Funktion `x /= y` testet Ungleichheit von x und y.
- $(==)$ und $(/=)$ haben den Typ `a -> a -> Bool`, hier `Int -> Int -> Bool`
- Und `data Bool = False | True` [3]

[3] Ja, in Haskell ist `Bool` nicht "eingebaut"