Einladung zum WIT-Kolloquium

Das Wissenschafterinnenkolleg Internettechnologien WIT in Zusammenarbeit mit der OCG lädt ein zu folgendem Vortrag...

How to give a great research talk? How to write a great research paper?

Simon Peyton Jones
Microsoft Research Cambridge

...einer der "Väter" von Haskell !!!

ZEIT: **Freitag, 15. Oktober 2004, 9:30 - 12:00 Uhr**

ORT: TU Wien, Neues Elektrotechnisches Institutsgebäude, HS EI 10,
1040 Wien, Gußhausstraße 27-29

MEHR INFO: `http://www.wit.at/events/` oder `http://inn.tuwien.ac.at/`

Das WIT erbittet Anmeldung unter `wit.tuwien.ac.at/events` !

Hinweise in eigener Sache...

- *Anmeldesystem ist freigeschaltet!*

  `http://www.complang.tuwien.ac.at/ivan/stud/`

  ...Anmeldungen sind noch bis zum 17.10.2004 möglich

- *Homepage der LVA ist umgezogen!*

  `http://www.complang.tuwien.ac.at/knoop/fp185161_ws0405.html`

  ...ist die neue Homepage-URL

Heutiges Thema

- Motivation
- Erste Schritte in Haskell und mit Hugs

Motivation

Warum...

- funktionale Programmierung?
- Haskell?

Warum funktionale Programmierung?

Programmierparadigmen...

- imperativ
  - prozedural (Pascal, Modula, C,...)
  - objektorientiert (Smalltalk, Oberon, C++, Java,...)
- deklarativ
  - funktional (Lisp, ML, Miranda, Haskell, Gofer,...)
  - logisch (Prolog und Varianten)
- ...

Ein Vergleich - imperativ vs. funktional

Gegeben ein Problem P.

Imperativ: Typischer Lösungsablauf in folgenden Schritten

1. Beschreibe eine(n) Lösung(sweg) L für P
2. Gieße L in die Form einer Menge von Anweisungen für den Rechner und organisiere dabei die Speicherverwaltung

Ein Beispiel

Problem: Bestimme die Komponentenwerte eines Feldes ganzer Zahlen mit Wert kleiner oder gleich 10

Typische Lösung imperativ...

```
...
j := 1;
FOR i:=1 TO maxLength DO
    IF a[i] <= 10 THEN b[j] := a[i]; j := j+1 FI
OD;
...
```

Stichwort: **Softwarekrise!**

Zum Vergleich...

Essenz funktionaler Programmierung: Statt des “wie” das “was” in den Vordergrund stellen!

Typische Lösung funktional...

```
...
a :: [Int]
b :: [Int]
...
b = [ n | n <- a, n <= 10 ]
...
```

Stichwort: **Listenkomprehension** (engl. list comprehension!)

...typisch und spezifisch für funktionale Sprachen!

Zum Vergleich: $\{n \mid n \in a \wedge n \leq 10\}$

...etwas der Eleganz der Mathematik in die Programmierung!

Noch nicht überzeugt?

Betrachte *Quicksort*, ein komplexeres Beispiel...

Aufgabe: Sortiere eine Liste L ganzer Zahlen aufsteigend.

Lösung mittels Quicksort:

- *Teile*: Wähle ein Element l aus L und partitioniere L in zwei (möglicherweise leere) Teillisten L_1 und L_2 so, dass alle Elemente von L_1 (L_2) kleiner oder gleich (größer) dem Element l sind.

- *Herrsche*: Sortiere L_1 und L_2 mit Hilfe rekursiver Aufrufe von Quicksort.

- *Zusammenführen der Teilergebnisse*: Trivial, die Gesamtliste entsteht durch Konkatenation der sortierten Teillisten.

Quicksort...

...eine typische imperative (Pseudocode-) Realisierung

```
quickSort (L,low,high)
  if low < high
    then splitInd = partition(L,low,high)
         quickSort(L,low,splitInd-1)
         quickSort(L,splitInd+1,high) fi

partition (L,low,high)
  l = L[low]
  left = low
  for i=low+1 to high do
    if L[i] <= l then left = left+1
                      swap(L[i],L[left]) fi od
  swap(L[low],L[left])
  return left
```

...und dem initialen Aufruf `quickSort(L,1,length(L))`.

Zum Vergleich...

...eine typische funktionale Realisierung von *QuickSort*, hier in Haskell:

```
quickSort :: [Int] -> [Int]

quickSort []     = []
quickSort (x:xs) =
    quickSort [ y | y<-xs, y<=x ] ++
                        [x] ++ quickSort [ y | y<-xs, y>x ]
```

Vorteile funktionaler Programmierung

- *Einfach(er) zu erlernen*
  ...da weniger Grundkonzepte (insbesondere: keine Zuweisung, keine Schleifen, keine Sprünge)

- *Höhere Produktivität*
  ...da Programme dramatisch kürzer als funktional vergleichbare imperative Programme (Faktor 5 bis 10)

- *Höhere Zuverlässigkeit*
  ...da Korrektheitsüberlegungen/-beweise einfach(er)
  (math. Hintergrund, keine durchscheinende Maschine)

Nachteile funktionaler Programmierung

- *Geringe(re) Effizienz*
  ...aber: enorme Fortschritte, zudem Korrektheit vorrangig gegenüber Geschwindigkeit, weiters einfache(re) Parallelisierbarkeit

- *Gelegentlich unangemessen*, etwa für inhärent zustandsbasierte Anwendungen oder zur GUI-Programmierung
  ...aber: Anwendungseignung ist stets zu überprüfen; kein spezifisches Problem fkt. Programmierung

Warum Haskell?

Ein Blick auf andere funktionale (Programmier-)sprachen...

- λ-Kalkül (Ende der 30er-Jahre, Alonzo Church, Stephen Kleene)
- Lisp (frühe 60er-Jahre, John McCarthy)
- ML, SML (Mitte 70er-Jahre, Michael Gordon, Robin Milner)
- Hope (um 1980, Rod Burstall, David McQueen)
- Miranda (um 1980, David Turner)
- OPAL (Mitte der 80er-Jahre, Peter Pepper et al.)
- Haskell (Ende der 80er-Jahre, Paul Hudak, Philip Wadler et al.)
- Gofer (Anfang der 90er-Jahre, Mark Jones)
- ...

Warum nicht Haskell?

Haskell ist...

- eine fortgeschrittene moderne funktionale Sprache
  - starke Typisierung
  - lazy evaluation
  - pattern matching
  - Polymorphie
  - Generizität
  - Datenabstraktion (abstrakte Datentypen)
  - Modularisierung (Programmierung im Großen)
  - ...
- eine Sprache für "real world" Probleme
  (s.a. `http://www.cs.bell-labs.com/~Swadler/realworld/`)
  - mächtige Bibliotheken
  - Schnittstellen z.B. zu C
  - ...

Nicht zuletzt: Wenn auch komplex, ist Haskell eine "gute" Lehrsprache, auch dank Hugs!

Fassen wir noch einmal zusammen...

Steckbrief: **Funktionale Programmierung**

Grundlage:	Lambda-Kalkül
Abstraktion:	Funktionen (höherer Ordnung)
Eigenschaft:	referentielle Transparenz
Historische Bedeutung:	Basis vieler Programmiersprachen
Anwendungsbereiche:	Theoretische Informatik Artificial Intelligence experimentelle Software Programmierunterricht
Programmiersprachen:	Lisp, ML, Miranda, Haskell,...

sowie...

Steckbrief: **Haskell**

benannt nach:	Haskell B. Curry (1900-1982) `http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Curry.html`
Paradigma:	rein funktionale Programmierung
Eigenschaften:	lazy evaluation, pattern matching
Typsicherheit:	stark typisiert, Typinferenz modernes polymorphes Typsystem
Syntax:	komprimiert, intuitiv
Informationen:	`http://haskell.org` `http://haskell.org/tutorial/`
Interpretierer:	Hugs (`http://haskell.org/hugs/`)

Ein erstes Haskell-Programm...

```
{-######################################################
   FirstScript.hs  ...``traditional scripts'' erhalten
                   die Dateiendung .hs
######################################################-}

-- Die konstante Funktion sum17and4
sum17and4 :: Int
sum17and4 = 17+4

-- Die Funktion square zur Quadrierung einer ganzen Zahl
square :: Int -> Int
square n = n*n

-- Die Funktion double zur Verdopplung einer ganzen Zahl
double :: Int -> Int
double n = 2*n

-- Die Funktion doubleSquare, eine Anwendung der vorherigen
doubleSquare :: Int
doubleSquare = double (square (4711 - sum17and4))
```

...in alternative Darstellung.

```
############################################################
   FirstLiterate.lhs  ...``literate scripts'' erhalten
                      die Dateiendung .lhs
############################################################

Die konstante Funktion sum17and4
>     sum17and4 :: Int
>     sum17and4 = 17+4

Die Funktion square zur Quadrierung einer ganzen Zahl
>     square :: Int -> Int
>     square = n*n

Die Funktion double zur Verdopplung einer ganzen Zahl
>     double :: Int -> Int
>     double = 2*n

Die Funktion doubleSquare, eine Anwendung der vorherigen
>     doubleSquare :: Int
>     doubleSquare = double (square (4711 - sum17and4))
```

Hugs: Der Haskell-Interpretierer...

Aufruf von Hugs: **`hugs <fileName>`**

...und im Fall von `FirstScript` für `<fileName>` z.B. weiter mit

```
Main> double (sum17and4)
42
```

Wichtige Kommandos in Hugs:

`:?`	Liefert Liste der Hugs-Kommandos
`:load <fileName>`	Lädt die Haskell-Datei `<fileName>` (erkennbar an Endung `.hs` bzw. `.lhs`)
`:reload`	wiederholt letztes Ladekommando
`:quit`	Beendet den aktuellen Hugs-Lauf
`:info name`	Liefert Information über das mit `name` bezeichnete "Objekt"
`:type exp`	Liefert den Typ des Argumentausdrucks `exp`
`:edit <fileName>.hs`	Öffnet die Datei `<fileName>.hs` enthaltende Datei im voreingestellten Editor
`:find name`	Öffnet die die Deklaration von `name` im voreingestellten Editor
`!<com>`	Ausführen des Unix- oder DOS-Kommandos `<com>`

...mehr dazu: **`http://www.haskell.org/hugs/`**

Fehlermeldungen&Warnungen in Hugs

- Fehlermeldungen
  - Syntaxfehler
    `Main> sum17and4 == 21)` ...liefert
    `ERROR: Syntax error in input (unexpected ‘)’)`
  - Typfehler
    `Main> sum17and4 + False` ...liefert
    `ERROR: Bool is not an instance of class ‘‘Num’’`
  - Programmfehler
    ...später
  - Modulfehler
    ...später
- Warnungen
  - Systemmeldungen
    ...später

Bequem...

Umfangreiche Bibliotheken (`Prelude.hs`,...) mit vielen vordefinierten Funktionen, z.B. zum

- Umkehren von Zeichenreichen (`reverse`)
- Aufsummieren von Listenelementen (`sum`)
- Verschmelzen von Listen (`zip`)
- ...

Soll eine Funktion gleichen Namens definiert werden, können Namenskonflikte durch *Verstecken* (engl. *hiding*) vordefinierter Namen vermieden werden.

Ergänze dazu...

```
import Prelude hiding (reverse,sum,zip)
```

...am Anfang des Haskell-Skripts im Anschluss an die Modul-Anweisung (so vorhanden), um `reverse`, `sum` und `zip` zu verbergen (mehr dazu später im Zusammenhang mit dem Modulkonzept von Haskell).

Zum ersten Aufgabenblatt...

- ...erhältlich ausschließlich im Web unter folgender URL
  `http://www.complang.tuwien.ac.at/knoop/fp185161_ws0405.html`
  ...ab morgen, Mi, den 13.10.2004
- Abgabe: Mi, den 20.10.2004, 12:00 Uhr
- "Nachträgliche" Abgabe: Mi, den 27.10.2004, 12:00 Uhr

Vorschau:
Ausgabe des...

- zweiten Aufgabenblatts: Mi, den 20.10.2004
- dritten Aufgabenblatts: Mi, den 27.10.2004

Vorschau auf die nächsten Vorlesungstermine...

- Di, 19.10.2004, 13-14 Uhr, Treitl-Hörsaal
- **Zusätzlicher Termin**: Di, 19.10.2004, 18-19 Uhr, HS 8 Heinz Parkus, Hauptgebäude, Karlsplatz 13
- Di, 26.10.2004 und Di, 02.11.2004 keine Vorlesung
- Ab Di, 09.11.2004, jeden Dienstag von 13-14 Uhr im Treitl-Hörsaal

Zum Abschluss für heute...

...noch ein weiteres Beispiel in Haskell, die *Fakultätsfunktion*.

Zur Erinnerung:

$$! : I\!N \rightarrow I\!N$$

$$n\,! = \begin{cases} 1 & \text{falls } n = 0 \\ n * (n-1)\,! & \text{sonst} \end{cases}$$

...und *eine* mögliche Realisierung in Haskell:

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