Funktionale Programmierung mit Haskell

050071 Praktikum zur Fachdidaktik
Alexander Ölzant, 9301547

Einleitung

- benannt nach Haskell Curry (1900 - 1982)
- Kommitee: Standardisierung 1987, heute: $Haskell\ 98$
- Interpreter (hugs, Haskell User's Gofer System)
- Compiler (Bytecode) + Engine (GHC, $Glasgow\ Haskell\ Compiler$)

Einleitung Sprachbeschreibung Haskell

- sauber konzipierte deklarative funktionale Sprache
- für alle Systeme verfügbar, Open-Source-Interpreter
- durch Tail Recursion ist Rekursion elegant und effizient
- lazy evaluation: nur Funktionen werden evaluiert, deren Resultate tatsächlich verwendet werden (aber: Eager Haskell mit speculative evaluation)

Haskell

- Kompakter Code
- bringt mathematische Konzepte näher ("Mind Expanding")
- keine Nebeneffekte (side effects
  $\longrightarrow$ Auswertungsreihenfolge einerlei (Theorem von Church/Rosser))
- Zirkuläre Definitionen sind möglich
- *"hübscher Code" (Deklarationen, keine Iterationen)*

Erste Schritte

Funktionale Sprache $\longrightarrow$ Evaluation von Funktionen als Grundprinzip

```
Prelude> 2*2
4
Prelude> fac 5 where fac = product . enumFromTo 1
120
Prelude> ['h','e','l','l','o'] ++ " world"
"hello world"
Prelude> do_something 99 (\x -> x * x) where
                      do_something x fn = fn x
9801
```

Datentypen

Integer, Bool, String (eigtl. liste von Char)

Tupel:

```
type Tier = (String, String, Int)

name   :: Tier -> String
linne  :: Tier -> String
nummer :: Tier -> Int

name (n,l,u) = n
linne (n,l,u) = l
nummer (n,l,u) = u
```

Lazy Evaluation

```
Prelude> enumFrom 1
[1,2,3,4,5,6,7,...
```

endlose Ausgabe der Liste

Lazy Evaluation

```
Prelude> enumFrom 1
[1,2,3,4,5,6,7,...
```

endlose Ausgabe der Liste

```
Prelude> take 10 (enumFrom 1)
[1,2,3,4,5,6,7,8,9,10]
```

... nur die benötigten Elemente werden berechnet

Lambda-Operator, ...

Zur Definition anonymer Funktionen

```
Prelude> do_something 99 (\x -> x * x) where
                     do_something x fn = fn x
9801
msort = foldl (\s -> \t -> (\(u, v) -> u++[t]++v)
                           (break (>t) s)  ) []
Prelude> msort "ZyXVAbCdE1230"
"0123ACEVXZbdy"
```

tail recursion

Nur ein Stack Frame für die Rekursion notwendig, daher nicht weniger effizient und viel eleganter als eine iterative Variante:

```
fac :: [Integer] -> [Integer] -> Integer
fac c m  | m == 1 = c
         | otherwise = fac (c*m) (m-1)

fac> fac 1 99
933262154439441526816992388562667004907159682643816214685929
299156089414639761565182862536979208272237582511852109168640
000000
```

Currying

Syntaktik nur einstellige Funktionen (-> ...), durch Currying

Fallstricke/caveats

http://www.haskell.org/tutorial/pitfalls.html

- Typisierung: keine implizite Umwandlung
- explizite Rekursion wegen der high order functions meist nicht notwendig

```
raise :: Num a => a -> [a] -> [a]
raise _ [] = []
raise x (y:ys) = x+y : raise x ys
```

eleganter:

```
raise x ys = map (x+) ys
```

Iterative Programmeirung

nicht zur Nachahmung, von http://www.willamette.edu/ fruehr/haskell/evolu

```
fac n = result (for init next done)
        where init = (0,1)
              next   (i,m) = (i+1, m * (i+1))
              done   (i,_) = i==n
              result (_,m) = m

for i n d = until d n i
```

Varianten

- GHS, HUGS s. o.
- Objektorientierte Versionen: Haskell++, O´Haskell und Mondrian
- Distributed Haskell
- Eager Haskell
- Concurrent Clean: GUI-Integration, keine Monaden, sondern unique types
  http://www.cs.ru.nl/~clean/

Links, weitere Informationen

- Eleven Reasons to use Haskell as a Mathematician
  http://sigfpe.blogspot.com/2006/01/eleven-reasons-to-use-haskell-as.html
- Haskellwiki
  http://www.haskell.org/
- Haskell Reference (zvon.org)
  http://zvon.org/other/haskell/Outputglobal/index.html
- Functional Programming in the Real World
  http://homepages.inf.ed.ac.uk/wadler/realworld/index.html