Funktionale Programmierung Mitschrieb

Finn Ickler

16. Oktober 2015

„Avoid sucess at all cost “

Simon Peyton Jones

List of Listings

Vorlesung 1

```haskell
-- Hello World Haskell
main :: IO ()
main = putStrLn "Chewie, we're home"
```

Codebeispiel 1: Hello World

Functional Programming (FP)

A programming language is a medium for expressive ideas (not to get a computer to perform operations). Thus programs must be written for people to read, and only incidentally for machines.

Computational Model in FP : ***Reduction***

Replace expressions by their value.
IN FP, expressions are formed by applying functions to values.

1. Function as in maths: $x = y \rightarrow f(x) = f(y)$

2. Functions are values like numbers or text

	FP	Imperative
construction	function application and composition	statement sequencing
execution	reduction (expression evaluation)	state changes
sementics	λ-calculus	denotational

$n \in \mathbb{N}, n \geq 2$ is a prime number $\Leftrightarrow$ the set of non-trivial factors of n is empty.
n is prime $\Leftrightarrow \{m \mid m \in m \in \{2, \ldots, n-1\}, nmodm = 0\} = \{\}$

```c
int IsPrime(int n)
{
    int m;
    int found_factor;
    found_factor
    for (m = 2; m <= n -1; m++)
    {
        if (n % m == 0)
        {
            found_factor = 1 ;
            break;
        }
    }
    return !found_factor;
}
```

Codebeispiel 2: isPrime in C

```haskell
isPrime :: Integer -> Bool
isPrime n = factors n == []
  where
    factors :: Integer -> [Integer]
    factors n = [ m ] | m <- [2..n-1], mod n m == 0]

main :: IO ()
main = do
  let n =42
  print (isPrime n)
```

Codebeispiel 3: isPrime in Haskell

```
let xs = [ x+1 | x <- [0..9] ]
:sprint xs = _
length xs
:sprint xs = [_,_,_,_,_,_,_,_,_]
```

Codebeispiel 4: Lazy Evaluation in der ghci REPL

Haskell Ramp Up

Read $\equiv$ as ”denotes the same value as”
Apply f to value e: f ␣e (juxtaposition, ”apply”, binary operator ␣, Haskell speak: infixL 10 ␣) = ␣has max precedere (10): f e_1 +e_2 $\equiv$(f e_1) + e_2 ␣associates to the left g ␣f ␣e $\equiv$ (g f) e Fonction composition:

- g (f e)

- Operator ”.” (”after”) : (g.f) e (. = $\circ$)

- Alternative ”apply” operator \$ (lowest precedure, associates to the right), infix 0\$): f\$$e_1$+ e_2 = f ($e_1 + e_2$)