Auswertung

Peter Thiemann

November 26, 2013

Contents

1 Wie wird eine Funktion strikt in einem Parameter?

Indem der Parameter „zwangsläufig benutzt" wird:

- Pattern-Matching auf dem Parameter:

```
(++) [] ys     = ys
(++) (x:xs) ys = x : (xs ++ ys)

(&&) False _ = False
(&&) True  b = b

-- strikt in beiden Parametern
(&&') False False = False
(&&') b     False = False
(&&') True  b     = b
```

2 Wie wird eine Funktion strikt in einem Parameter?

- Zwangsläufige Anwendung von strikten Funktionen auf den Parameter (unter LO)

```
inc x = x + 5
incOrDec True  x = x + 5
incOrDec False x = x - 5
```

- Zwangsläufige Anwendung des Parameters als Funktion. bzw Rückgabe des Parameters (unter LO):

```
id x = x

const x y = y

(.) f g x = f (g x)
```

(siehe auch: Striktheit von ($))

q* Konstruktoren

- sind auch nicht-strikt
- (Konstruktorargumente werden auch nicht ausgewertet)
- Wert: ...

  „Partiell oder vollständig angewendete **Konstruktoren**"
- (`fib 2000):(3:[])`, `Just undefined`, `undefined:5:undefined:[]`, `(undefined:)`

3 Representation of a constructor

3.1 vector in memory

3.2 first cell: identification of constructor

3.3 remaining cells: arguments (unevaluated)

3.4 must be initialized in one step

4 Haskell wraps this vector into as many lambdas as there are constructor argument

4.1 C = 1 ... xn -> (C#, x1, ..., xn)

4.2 C e1 = 2 ... xn -> (C#, e1, ..., xn)

5 Einschub: Pattern-Matching

Bisher habe wir Funktionen durch Fallunterscheidung geschrieben:

```
take1 []     = []
take1 (x:xs) = [x]
```

Wie verarbeiten die Reduktionsregeln `take1 [1,2]`?

6 Einschub: Pattern-Matching

```
take1 []     = []
take1 (x:xs) = [x]
```

ist eine andere Schreibweise für:

```
take1 a = case a of
    []     -> []
    (x:xs) -> [x]
```

7 Einschub: Pattern-Matching

Reduktionsregeln für `case e of ps`, wobei `ps = pattern_1 -> e1 ... pattern_n -> en`

```
           C x1 ... xn -> e'
   ist erstes passendes Pattern in ps
-------------------------------------------------
case C e1 ... en of ps ====> e'[x1->e1,...,xn->en]
```

```
case 1:(2:[]) of
  []     -> []
  (x:xs) -> [x]

====>

[1]
```

8 Einschub: Pattern-Matching

Reduktionsregeln für `case e of ps`, wobei `ps = pattern_1 -> e1 ... pattern_n -> en`

```
             x -> e'
   ist erstes passendes Pattern in ps
-------------------------------------
      case e of ps ====> e'[x/e]
```

Beispiel

9 Einschub: Pattern-Matching

Reduktionsregeln für `case e of ps`, wobei `ps = pattern_1 -> e1 ... pattern_n -> en`

```
           e ====> e'
---------------------------------
case e of ps ====> case e' of ps
```

10 Vorteile Nicht-Strikter Konstruktoren:

Es ist möglich unendlich große Werte zu definieren und (partiell) zu verwenden.

```
take1 xs  = case xs of
     []    -> []
     (x:_) -> [x]

iterate f z = z : iterate f (f z)

take1 (iterate (+1) 0)              ====>
case (iterate (+1) 0) of ..         ====>
case (0:iterate (+1) (0 + 1)) of .. ====>
[0]
```