III.4 Muster (Patterns)

Freitag, 20. Januar 2017 16:30

Patterns ≙ spezielle Ausdrücke, um Formen der erwarteten Argumente einer Funktion zu beschreiben

Pattern Matching: teste definierende Gleichungen von oben nach unten u. verwende die erste Gleichung, bei der die Patterns auf die aktuellen Elemente passen.

Pattern ≈ Ausdruck aus Variablen u. Datenkonstruktoren

```
f :: [Int] -> Bool      <- testet, ob Liste mit 5 beginnt
f [ ]      = False
f (5 : xs) = True
f  xs      = False
```

weiteres Bsp: app (append, Listkonkatenation)

```
app [1,2] [3,4,5]  = [1,2,3,4,5]
```

Ist in Haskell unter dem Namen ++ vordefiniert:

```
[1,2] ++ [3,4,5]  = [1,2,3,4,5]
```

Haskells Auswertungsstrategie verfolgt das Leftmost-Outermost Prinzip.

Wenn bei f (g ...) nicht klar ist, welche def.

Gleichung von f anzuwenden ist, dann wertet man g so lange aus, bis man weiß, welche f-Gleichung anwendbar ist. Anschließend wird f ausgewertet.

```
  len (app [1] [2])
      1:[]
= len (1 : app [] [2])
= 1 + len (app [] [2])
= 1 + len [2]
          2:[]
= 1 + (1 + len [])
= 1 + (1+0)
= 1 + 1
= 2
```

Weiteres Bsp:

```
f :: [Int] -> [Int] -> [Int]
f [] ys = []
f xs [] = []
```

```
zeros :: [Int]
zeros = 0 : zeros
```

```
f [] zeros = []                terminiert
f zeros [] = f (0:zeros) [] = []
                                terminiert
```

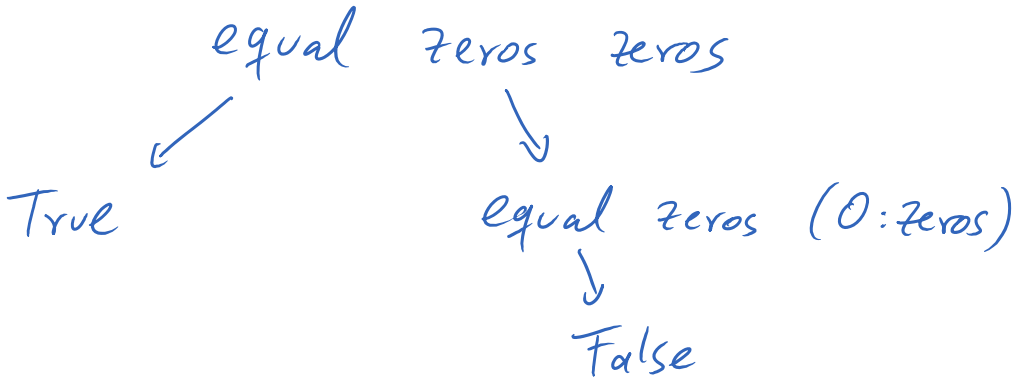
Es wird nur solange das Argument ausgewertet, bis man seine obersten Datenkonstruktoren kennt.

Einschränkung: Patterns müssen linear sein

Genauer: Auf der linken Seite einer definierenden Gleichung darf keine Variable mehrfach auftreten.

Grund: Man möchte, dass das Ergebnis der Auswertung nicht von der Auswertungsstrategie abhängt.

```
zeros = 0 : zeros
```



Syntaxdiagramm für Patterns:

Beschreibe, auf welche Ausdrücke der Pattern passt und wie die Variablen beim Pattern Matching instantiiert werden.

- Var

```
Square x = x * x
       ↑
    Pattern
```

passt auf alle Ausdrücke, beim Pattern Matching wird die Variable mit dem entsprechenden Wert instantiert

- _ (underscore) Joker-Pattern

  passt auch auf alle Ausdrücke, aber beim Pattern Matching wird keine Variable instantiert.
  Mehrfache Vorkommen von _ dürfen für unterschiedliche Ausdrücke stehen.

- integer, float, char, string

  Diese Pattern passen nur auf sich selbst u. es findet keine Instantiierung von Variablen statt.

```
is_7 :: Int -> Bool

is_7  7 = True
is_7  _ = False
```

- Constr $pat_1 \dots pat_n$, wobei Constr ein n-stelliger Datenkonstruktor ist

  (z.B. True, [] sind 0-stellige Datenkonstruktoren
  : ist 2-stelliger " ")

Constr $pat_1 \dots pat_n$ passt auf alle Werte, die mit dem Datenkonstruktor Constr gebildet wurden, falls pat_1 auf das 1. Arg passt, pat_2 auf

das 2. Arg. passt, ...

- $[pat_1, .., pat_n]$, $n \geq 0$

  passt auf alle Ausdrücke der Form $[exp_1, \ldots, exp_n]$, falls pat_1 auf exp_1 passt, ..., pat_n auf exp_n passt.

- $(pat_1, .., pat_n)$, $n \geq 0$

  passt auf alle Tupel $(exp_1, .., exp_n)$, falls pat_1 auf exp_1 passt, .., pat_n auf exp_n passt.