III.4 Patterns

Dienstag, 19. Dezember 2017 09:00

Patterns sind Ausdrücke, die beschreiben, welche Argumente eine Fkt. erwartet.

Im wesentlichen:

Patterns ≃ Ausdrücke aus Variablen u. Datenkonstruktoren

Zur Auswertung einer Fkt wird überprüft, welcher Pattern aus der Fkt-Deklaration auf das aktuelle Argument passt (Pattern Matching).

Alg. append berechnet Listenkonkatenation. Ist in Haskell als Infix-Funktion ++ vordefiniert:

[1,2] ++ [3,4,5] ergist

[1,2,3,4,5]

Auswertungsstrategie:
Lazy Evaluation, d.h.
Auswertung ganz außen.
Wenn die äußere Fkt nicht
ausgewertet werden kann, weil
unklar ist, welcher Pattern ihrer
Funktionsdeklaration passt,
dann werte Argument ein paar
Schritte aus, bis man ent-
scheiden kann, welche defi-
nierende Gleichung der
äußeren Fkt. angewendet
werden muss.
(Def. Gleichungen werden
v. oben u. unten überprüft.)

len (app [1] [2]) ← werte app aus, bis man

len (app [1] [2]) ← werte app aus, bis man weiß, welche len-Gleichung angewendet werden muss
        1:[]

= len (1 : app [] [2])

= 1 + len (app [] [2]) ← vordef. arithmet. Operationen wie +, -, ..., >, <, ... können nur ausgew. werden, wenn beide Argumente Zahlen sind

= 1 + len [2]
        2:[]

= 1 + (1 + len [])

= 1 + (1 + 0)

= 1 + 1

= 2

Weiteres Bsp:

```
f [] ys = []
f xs [] = []
```

Auswertung v. zeros

```
zeros = 0 : zeros
```

← terminiert nicht. Liefert ∞ Liste [0, 0, ...]

Auswertung von:

f [] zeros ergist []

f zeros []

= f (0 : zeros) []

= []

terminiert ebenfalls

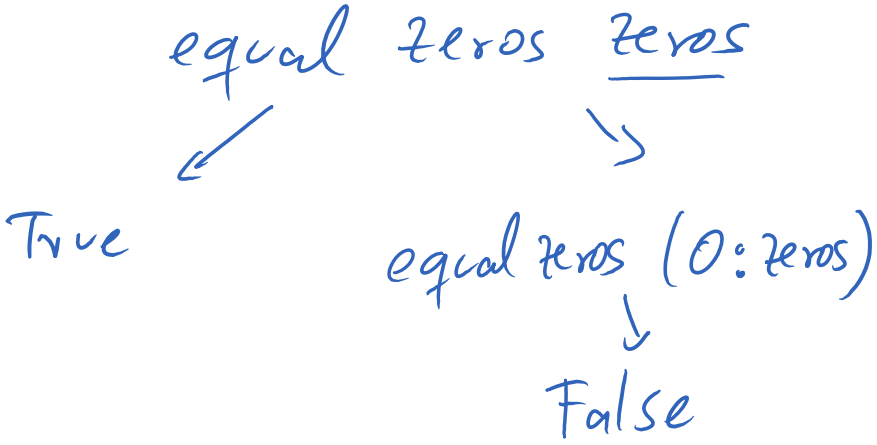
Einschränkung:

Patterns müssen linear sein (d.h: keine Variable darf in den Patterns einer linken Seite mehrfach vorkommen).

Grund: Sonst könnte das Resultat der Auswertung von der Auswertungsstrategie abhängen.

zeros = 0 : zeros

Auswertung von:



Grammatik für Patterns

Gib jeweils an:

- auf welche Ausdrücke passt ein Pattern?
- wie werden die Variablen des Patterns beim Pattern Matching instantiiert?

- Variable z.B.:

  square x = x * x

  passt auf jeden Wert, Variable wird dann mit dem jeweiligen Wert instantiiert

  square 5 = 5 * 5 = 25

- _ (underscore) ist der Joker-Pattern.
  Passt auch auf jeden Wert, aber es erfolgt keine Instantiierung von Variablen.
  Mehrfache Vorkommen von _ in einer linken Seite stehen für bel. (potentiell unterschiedliche) Werte.
- integer, float, char, string
  Diese Patterns passen nur auf sich selbst und es findet keine Instantiierung von Variablen statt.

```
is_5 :: Int -> Bool
is_5  5  = True
is_5  _  = False
```

- $(constr\ pat_1 \dots pat_n)$, $n \geq 0$

Hierbei ist constr ein n-stelliger Datenkonstruktor (z.B. True, :).

Constr $pat_1 \dots pat_n$ passt auf Ausdrücke der Art Constr $exp_1 \dots exp_n$, falls pat_1 auf exp_1 passt und ... ... pat_n auf exp_n passt.

```
f :: [Int] -> Bool
f (5:_) = True
f _     = False
```

(Zur Erkennung aller Listen, die mit 5 beginnen)

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

passt auf Listen $(exp_1, .., exp_n)$
der Länge n, falls pat_1 auf
exp_1 passt und ... und
pat_n auf exp_n passt.

- $(pat_1, ..., pat_n)$, $n \geq 0$
passt auf Tupel $(exp_1, .., exp_n)$
mit n Komponenten, falls
pat_1 auf exp_1 passt und ...
und pat_n auf exp_n passt.