

1.1.3 Patterns

Dienstag, 19. April 2016 15:30

Patterns are special expressions that are used to describe the forms of expected arguments.

$$\begin{array}{l|l} \text{len} :: [\text{Int}] \rightarrow \text{Int} & \text{append} :: [\text{Int}] \rightarrow [\text{Int}] \rightarrow [\text{Int}] \\ \text{len} [] = 0 & \text{append} [] \text{ ys} = \text{ys} \\ \text{len} (x:xs) = 1 + \text{len} xs & \text{append} (x:xs) \text{ ys} = x : \text{append} xs \text{ ys} \end{array}$$

append is pre-defined as an infix-function called ++ :

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

$$\text{len} (\text{append} \underbrace{[1]}_{1:[]} [2]) =$$

← to evaluate len, we need to evaluate append a few steps until we know whether the first defining equation of len matches

$$\text{len} (1 : \text{append} [] [2]) =$$

$$1 + \text{len} (\text{append} [] [2]) = \dots = 2$$

$$\text{zeros} :: [\text{Int}]$$

$$\text{zeros} = 0 : \text{zeros}$$

$$f :: [\text{Int}] \rightarrow [\text{Int}] \rightarrow [\text{Int}]$$

$$f [] \text{ ys} = []$$

$$f \text{ xs} [] = []$$

$$f [] \text{ zeros} = [] \quad \text{terminates}$$

$$f \text{ zeros} [] = f (0 : \text{zeros}) [] = [] \quad \text{terminates}$$

↑

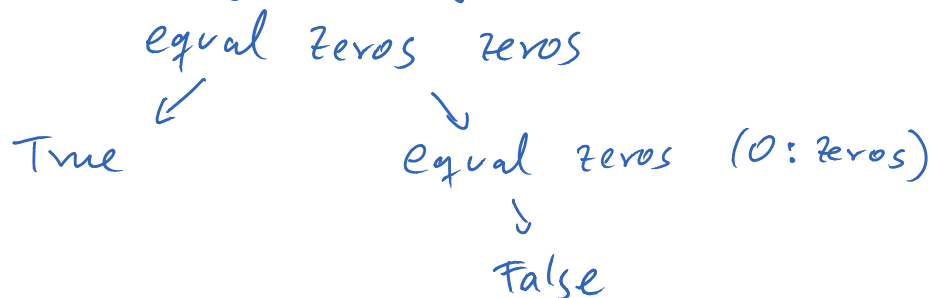
at this point, one notices that the pattern [] in the first f-equation does not match.

Patterns have to be linear, i. e., no variable may occur twice on the lhs of a defining equation.

Reason: $\text{equal} :: [\text{Int}] \rightarrow [\text{Int}] \rightarrow \text{Bool}$ } not allowed in Haskell
 $\text{equal } xs \ xs = \text{True}$
 $\text{equal } xs \ (x:xs) = \text{False}$

Up to now: evaluation strategy influences termination and efficiency, but not the result.

Here: evaluation strategy does influence result:



For every form of pattern (Slide 12), we describe which arguments are matched by this pattern and how the variables of the pattern are instantiated.

- var : a variable matches any expression, matching instantiates the variable by the expression

square $x = x * x$
pattern

square $11 = 11 * 11 = 121$

- _ : joker pattern, matches any expression, but no variable is instantiated. "_" may occur multiple times, stands for possibly different values

stands for possibly different values

und True $y = y$

und $_ _ = \text{False}$

und False True = False

- integer, float, char, string: these patterns only match themselves

$\text{is_5} :: \text{Int} \rightarrow \text{Bool}$

$\text{is_5 } 5 = \text{True}$

$\text{is_5 } _ = \text{False}$

- $(\text{constr } \underline{\text{pat}}_1 \dots \underline{\text{pat}}_n)$, $n \geq 0$ matches all expressions $(\text{constr } \underline{\text{exp}}_1 \dots \underline{\text{exp}}_n)$ where $\underline{\text{pat}}_i$ matches $\underline{\text{exp}}_i$
↑
data constructor (e.g. True, [], :, ...)

e.g. $\text{len } (\underbrace{x : xs}_{\text{pattern}}) = 1 + \text{len } xs$

- $[\underline{\text{pat}}_1, \dots, \underline{\text{pat}}_n]$, $n \geq 0$ matches all lists

$[\underline{\text{exp}}_1, \dots, \underline{\text{exp}}_n]$ where $\underline{\text{pat}}_i$ matches $\underline{\text{exp}}_i$

$\text{has_length_three} :: [\text{Int}] \rightarrow \text{Bool}$

$\text{has_length_three } [-, -, -] = \text{True}$

$\text{has_length_three } _ = \text{False}$

- $(\underline{\text{pat}}_1, \dots, \underline{\text{pat}}_n)$ where $n \geq 0$ matches all tuples

$(\underline{\text{exp}}_1, \dots, \underline{\text{exp}}_n)$ where $\underline{\text{pat}}_i$ matches $\underline{\text{exp}}_i$.

$$\text{maxi} :: (\text{Int}, \text{Int}) \rightarrow \text{Int}$$

$$\text{maxi } (0, y) = y$$

$$\text{maxi } (x, 0) = x$$

$$\text{maxi } (x, y) = 1 + \text{maxi } (x-1, y-1)$$