Mehr über Haskell, Teil 2

...von Basistypen (Int, Bool, Char,...)
zu zusammengesetzten Datentypen

$\rightsquigarrow$ Tupel und Listen

...und zu Funktionen darauf.

Tupel

Tupel ...fassen eine festgelegte Zahl von Werten möglicherweise verschiedener Typen zusammen.

Beispiele:

- ...Modellierung von Studentendaten

```
("MaxMuster","e0123456@student.tuwien.ac.at",534) ::
                    Num a => ([Char], [Char], a)
```

- ...Modellierung von Bibliotheksdaten

```
("PeytonJones","Implementing Funct. Lang.",1987,True) ::
                    Num a => ([Char], [Char], a, Bool)
```


Tupel...

- Allgemeines Muster

```
(v1,v2,...,vk) :: (T1,T2,...,Tk)
```

wobei

```
v1 :: T1, v2 :: T2,..., vk :: Tk
```

mit `v1,...,vk` Werten und `T1,...,Tk` Typen

- Standardkonstruktor

```
(. , . , ... , . )
```


Spezialfall: Paare (“Zweitupel”)

- Beispiele

```
type Point = (Float, Float)

(0.0,0.0) :: Point
(3.14,17.4) :: Point
```

- Standardselektoren (für Paare)

```
fst (x,y) = x
snd (x,y) = y
```

- Anwendung der Standardselektoren

```
fst (0.0,0.0) = 0.0
snd (3.14,17.4) = 17.4
```


Hilfreich...

Typsynonyme

```
type Student = (String, String, Int)
type Buch = (String, String, Int, Int)
```

...erhöhen die Transparenz in Programmen.

Tupel...

Selbstdefinierte Selektorfunktionen...

```
type Student = (String, String, Int)

name  :: Student -> String
email :: Student -> String
kennzahl :: Student -> Int

name (n,e,k)     = n
email (n,e,k)    = e
kennZahl (n,e,k) = k
```

...auch hier mittels Mustererkennung (engl. pattern matching)

Selbstdefinierte Selektorfunktionen...

Ein weiteres Beispiel...

```
autor :: Buch -> String
kurzTitel :: Buch -> String
erscheinungsJahr :: Buch -> Int
ausgeliehen :: Buch -> Bool

autor (a,t,j,b)  = a
kurzTitel (a,t,j,b)  = t
erscheinungsJahr (a,t,j,b)  = j
ausgeliehen (a,t,j,b)  = b

autErsch (a,t,j,b) = (autor (a,t,j,b), ausgeliehen (a,t,j,b))
```

...mittels Mustererkennung

Nebenbei bemerkt...

Die Deklaration...

```
binom :: (Integer,Integer) -> Integer
binom (n,k)
  | k==0 || n==k   = 1
  | otherwise      = binom (n-1,k-1) + binom (n-1,k)
```

...ist Beispiel einer musterbasierten Funktionsdefinition.

Zum Vergleich...

mit Standardselektoren ohne Muster...

```
binom :: (Integer,Integer) -> Integer
binom p
  | snd(p)==0 || snd(p)==fst(p)  = 1
  | otherwise  = binom (fst(p)-1,snd(p)-1)
                         + binom (fst(p)-1,snd(p))
```


Typsynonyme: Beachte...

- `type` ...führt neue Namen für bereits existierende Typen ein (Typsynonyme!), keine neuen Typen.

Daher...

Typsyonyme...

- tragen zur Dokumentation bei und
- erleichtern (i.a.) das Programmverständnis

aber...

- führen nicht zu (zusätzlicher) Typsicherheit!

Ein Beispiel

```
type Euro        = Float
type Yen         = Float
type Temperature = Float

myPi    :: Float
daumen  :: Float
maxTemp :: Temperature
myPi    = 3.14
daumen  = 5.55
maxTemp = 43.2

currencyConverter :: Euro -> Yen
currencyConverter x = x + myPi * daumen
```

Mit obigen Deklarationen...

```
currencyConverter maxTemp => 60.627
```

...werden 43.2 °C in 60.627 Yen umgerechnet. Typsicher?

Listen

Listen ...fassen eine beliebige/unbestimmte Zahl von Werten gleichen Typs zusammen.

Beispiele:

- ...Modellierung von Vorlesungshörern

```
[("MaxMuster","e0123456@student.tuwien.ac.at",534),
 ("MoritzMuster","e0987654@student.tuwien.ac.at",538),
 ("MimiMuster","e0987654@student.tuwien.ac.at",536)] :: [Student]
```

- ...Modellierung von Bibliotheken

```
[("PeytonJones","Implementing Funct. Lang.",1987,True),
 ("Pepper","Funkt. Prog.",2003,False)] :: [Buch]
```


Listen

Bereits vordefiniert...

`Type String = [Char]`

Auch möglich...

- Listen von Listen
  `[[2,4,23,2,5],[3,4],[],[56,7,6,]] :: [ [Int] ]`
- Listen von Paaren
  `[(3.14,42.0),56.1,51.3),(1.12,2.22)] :: [ Point ]`
- ...
- Listen von Funktionen
  `[fact, fib, fibFix] :: [ Integer -> Integer ]`

Erinnerung aus der Mathematik...

Eine Funktion f ist ein Tripel (D,W,G) mit einer Definitionsmenge (-bereich) D, einer Wertemenge (-bereich) W und einer linkseindeutigen Relation G mit $G \subseteq D \times W$, dem sog. Funktionsgraphen von f.

Mithin...

Funktionen sind spezielle Relationen; spezielle Teilmengen eines kartesischen Produkts

Damit intuitiv naheliegend...

Listen von Funktionen ... "Listen von Listen von Paaren"

Schlagwort...

Funktionen als *"first-class citizens"*

Schlaglichter über Listen (1)

length :: [a] –> Integer
length [] = 0
length (x:xs) = 1 + length xs

length [1, 2, 3] ⇒ 3 ([1,2,3] = 1:2:3:[])
length ['a','b','c'] ⇒ 3
length [[1],[2],[3]] ⇒ 3

head :: [a] –> a
head (x:xs) = x

tail :: [a] –> [a]
tail (x:xs) = xs

['h','e','l','l','o'] = "hello"
"hello" ++ " world" ⇒ "hello world"

Schlaglichter über Listen (2)

- Spezialfälle (i.w. für Listen über Zahlen und Zeichen)
  ...`[2 .. 6]` kurz für `[2,3,4,5,6]`
  ...`[11,9 .. 2]` kurz für `[11,9,7,5,3]`
  ...`['a','d' .. 'j']` kurz für `['a','d','g','j']`
  ...`[0.0,0.3 .. 1.0]` kurz für `[0.0,0.3,0.6,0.9]`
- Listenkomprehension
  ...ein erstes Beispiel:
  `[3*n | n <- list]` kurz für `[3,6,9,12]`, falls `list` vom Wert `[1,2,3,4]` ist.

  ⇝ Listenkomprehension ist ein sehr ausdruckskräftiges Sprachkonstrukt!

Wiederholung&Ausblick auf weiteres...

Muster, Wild Cards und Typvariablen...

```
takeFirst m ys = case (m,ys) of
                   (0,_)      -> []
                   (_,[])     -> []
                   (n,x:xs)   -> x : takeFirst (n - 1) xs

if then else :: Bool -> a -> a -> a
if c then t else e = case c of True  -> t
                               False -> e
```


Vorschau auf die nächsten Vorlesungstermine...

- Ab Di, 25.11.2003, jeden Dienstag von 13-14 Uhr im Treitl-Hörsaal