

## Heutiges Thema...

Datenstrukturen in Haskell...

- Algebraische Datentypen (`data Tree = ...`)
- Typsynonyme (`type Student = ...`)
- Spezialitäten (`newtype State = ...`)

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

1

## Datentypdeklarationen in Haskell

...selbstdefinierte (neue) Datentypen in Haskell!

~> Haskell's Vehikel dafür: *Algebraische Typen*

*Algebraische Typen* erlauben uns zu definieren...

- Summentypen
  - *Spezialfälle*
  - \* Produkttypen
  - \* Aufzählungstypen

In der Praxis besonders wichtige Varianten...

- Rekursive Typen (~> "unendliche" Datenstrukturen)
- Polymorphe Typen (~> Wiederverwendung): *Später!*

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

2

## Grundlegende Typmuster

Aufzählungs-, Produkt- und Summentypen:

- *Aufzählungstypen*
  - ~> Typen mit endlich vielen Werten
  - ...typisches Beispiel: Typ Jahreszeiten mit Werten `Fruehling, Sommer, Herbst und Winter`.
- *Produkttypen* (Synonym: *Verbundtypen*)
  - ~> Typen mit möglicherweise unendlich vielen Tupelwerten
  - ...typisches Beispiel: Typ Person mit Werten `(Adam, maennlich, 27)`, `(Eva, weiblich, 25)`, etc.
- *Summentypen* (Synonym: *Vereinigungstypen*)
  - ~> Vereinigung von Typen mit möglicherweise jeweils unendlich vielen Werten
  - ...typisches Beispiel: Typ Verkehrsmittel als Vereinigung der (Werte der) Typen `Auto`, `Schiff`, `Flugzeug`, etc.

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

3

## Zum Einstieg und Vergleich... (1)

Realisierung von Typdefinitionen in imperativen Sprachen

...hier am Bsp. von Pascal

- *Aufzählungstypen*

```
TYPE jahreszeiten = (fruehling, sommer, herbst, winter);
spielkartenfärben = (krenz, pik, herz, karo);
werkzeuge = (montag, dienstag, mittwoch,
             donnerstag, freitag);
transportmittel = (fahrrad, auto, schiff, flugzeug);
form = (kreis, rechteck, quadrat, dreieck);
```
- *Produkttypen*

```
TYPE person = RECORD
  name: ARRAY [1..42] OF char;
  geschlecht: (maennlich, weiblich);
  alter: integer
END;
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

4

## Zum Einstieg und Vergleich... (2)

- *Summentypen*

```
TYPE verkehrsmittel =
RECORD
CASE vkm: transportmittel OF
  fahrrad: (tandem: Boolean);
  auto: (hersteller: ARRAY [1..30] OF char;
        hubraum: real);
  schiff: (name: ARRAY [1..30] OF char;
          tiefgang: real;
          heimathafen: ARRAY [1..50] OF char);
  flugzeug: (reichweite: real;
            sitzplaetze: integer)
END;

geometrischefigur =
RECORD
CASE fgr: form OF
  kreis: (radius: real);
  rechteck: (breite, hoehe: real);
  quadrat: (seitenlaenge, diagonale: real);
  dreieck: (s1, s2, s3: real; rechteck: boolean);
END;
```

## Zum Einstieg und Vergleich... (4)

Obige Einstiegsdatentypbeispiele in Haskell...

- Aufzählungstyp Jahreszeiten

```
data Jahreszeiten = Fruehling | Sommer | Herbst | Winter
data Werktage    = Montag | Dienstag | Mittwoch | Donnerstag | Freitag
data Bool        = True | False
```
- Produkttyp Person

```
data Person = Pers Name Geschlecht Alter

mit

type Name = String
type Alter = Int
data Geschlecht = Maennlich | Weiblich
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

7

## Zum Einstieg und Vergleich... (3)

Aufzählungstypen, Produkttypen, Summentypen...

- In Pascal ...drei verschiedene Sprachkonstrukte
- In Haskell ...ein *einheitliches* Sprachkonstrukt  
~> die *algebraische Datentypdefinition*

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

6

## Zum Einstieg und Vergleich... (5)

- Summentyp Verkehrsmittel

```
data Verkehrsmittel = Fahrrad Bool |
                    Auto String Float |
                    Schiff String Float String |
                    Flugzeug Float Int
```

In obiger Form offenbar wenig transparent im Vergleich zu:

```
TYPE verkehrsmittel =
RECORD
CASE vkm: transportmittel OF
  fahrrad: (tandem: Boolean);
  auto: (hersteller: ARRAY [1..30] OF char;
        hubraum: real);
  schiff: (name: ARRAY [1..30] OF char;
          tiefgang: real;
          heimathafen: ARRAY [1..50] OF char);
  flugzeug: (reichweite: real;
            sitzplaetze: integer)
END;
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

8

## Zum Einstieg und Vergleich... (5)

- Summentyp Verkehrsmittel

```
data Verkehrsmittel = Fahrrad Tandem |
  Auto Hersteller Hubraum |
  Schiff Name Tiefgang Heimathafen |
  Flugzeug Spannweite Sitzplaeetze
```

mit

```
type Tandem      = Bool
type Hersteller  = String
type Hubraum     = Float
type Name        = String
type Tiefgang    = Float
type Heimathafen = String
type Reichweite  = Float
type Sitzplaeetze = Int
```

Man *erkennt*: Typsynonyme bringen *Transparenz* ins Programm!

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

0

## Algebraische Datentypen in Haskell

... das allg. Muster der algebraischen Datentypdefinition:

```
data Typname
  = Con1 t11 ... t1k1 |
  Con2 t21 ... t2k2 |
  ...
  Conn tn1 ... tnkn
```

*Sprechweisen*:

- Typname ... *Typname*-/identifikator

- $\text{Con}_i, i = 1..n$  ... *Konstruktor*(en)/-identifikatoren

- $k_j, j = 1..n$  ... *Stelligkeit* des Konstruktors  $\text{con}_i, k_i \geq 0, i = 1, \dots, n$

*Beachte*: Typ- und Konstruktordentifikatoren müssen mit einem Großbuchstaben beginnen (siehe z.B. True, False)!

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

0

## Konstruktoren...

... können als Funktionsdefinitionen gelesen werden:

```
Coni :: t11 -> ... -> t1k1 -> Typname
```

Konstruktion von Werten eines algebraischen Datentyps durch...

... Anwendung eines Konstruktors auf Werte "passenden" Typs, d.h. ....

```
Coni v11 ... v1k1 :: Typname
```

Wobei  $v_{11} :: t_{11} \dots v_{1k_1} :: t_{1k_1}, j = 1, \dots, k_1$

*Beispiele*:

- Pers "Adam" Maennlich 27 :: Person
- Schiff "Donaukönigin" 2.74 "Wien" :: Verkehrsmittel
- Flugzeug 8540.76 276 :: Verkehrsmittel

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

11

## Aufzählungstypen (2)

Funktionsdefinitionen über Aufzählungstypen...

~> üblicherweise mit Hilfe von Pattern-matching.

*Beispiele*:

```
hatecken :: Form -> Bool
hatecken Kreis = False
hatecken _     = True
```

```
istLandgebunden :: Verkehrsmittel -> Bool
```

```
istLandgebunden Fahrrad = True
```

```
istLandgebunden Auto    = True
```

```
istLandgebunden _      = False
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

13

## Summentypen (1)

Mehrere (null- oder mehrstellige) Konstruktoren führen auf Summentypen...

*Beispiel*:

```
type Radius      = Float
type Breite      = Float
type Hoehe       = Float
type Seite1      = Float
type Seite2      = Float
type Seite3      = Float
type Rechtwinklig = Bool
```

```
data XFigur = Kreis Radius |
  Rechteck Breite Hoehe |
  Quadrat Kantenlaenge |
  Dreieck Seite1 Seite2 Seite3 Rechtwinklig |
  Ebene
```

Die Varianten einer Summe werden durch "|" getrennt.

## Aufzählungstypen (1)

Nullstellige Konstruktoren führen auf *Aufzählungstypen*...

*Beispiele*:

```
data Spielfarbe = Kreuz | Pik | Herz | Karo
data Wochenende = Sonnabend | Sonntag
data Geschlecht = Maennlich | Weiblich
data Form       = Kreis | Rechteck | Quadrat | Dreieck
```

Insbesondere ist der Typ der Wahrheitswerte...

```
data Bool = True | False
```

... *Beispiel* eines in Haskell vorderfinierten Aufzählungstyps.

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

12

## Produkttypen

(Alternativenlose) mehrstellige Konstruktoren führen auf *Produkttypen*...

*Beispiel*:

```
type Name      = String
type Alter     = Int
data Geschlecht = Maennlich | Weiblich

data Person = Pers Name Geschlecht Alter
```

*Beispiele*: ...für Werte des Typs Person.

```
Pers "Paul Piffiff" Maennlich 23 :: Person
```

```
Pers "Paula Pfietsch" Weiblich 22 :: Person
```

*Beachte*: Funktionalität der Konstrukturfunktion ist hier...

```
Pers :: Name -> Geschlecht -> Alter -> Person
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

14

## Summentypen (2)

*Beispiele*: ...für Werte des Typs erweiterte Figur XFigur

```
Kreis 3.14      :: XFigur
Rechteck 17.0 4.0  :: XFigur
Quadrat 47.11   :: XFigur
Dreieck 3.0 4.0 5.0 True  :: XFigur
Ebene        :: XFigur
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

16

## Zwischenfazit

Somit ergibt sich die eingangs genannte Taxonomie algebraischer Datentypen...

Haskell offeriert...

- Summentypen

mit den beiden *Spezialfällen*

- Produkttypen
  - ~> nur ein Konstruktor, mehrstellig
- Aufzählungstypen
  - ~> ein oder mehrere Konstruktoren, alle nullstellig

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

17

## Rekursive Typen (1)

...der Schlüssel zu (potentiell) unendlichen Datenstrukturen.

*Technisch:*

...zu definierende Typnamen können rechtsseitig in der Definition benutzt werden.

*Beispiel:* ... (arithmetische) Ausdrücke

```
data Expr = Opd Int |
           Add Expr Expr |
           Sub Expr Expr |
           Squ Expr
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

18

## Rekursive Typen (2)

*Beispiele* ... für Ausdrücke (lies --> als "entspricht").

```
Opd 42 :: Expr           --> 42
Add (Opd 17) (Opd 4) :: Expr   --> 17+4
Add (Squ (Sub (Opd 42) (Squ (2)))) (Opd 12) :: Expr
--> square(42-square(2))+12
```

...rekursive Typen ermöglichen potentiell unendliche Datenstrukturen!

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

19

## Rekursive Typen (3)

*Weiteres Beispiel:*

Binärbäume, hier zwei verschiedene Varianten:

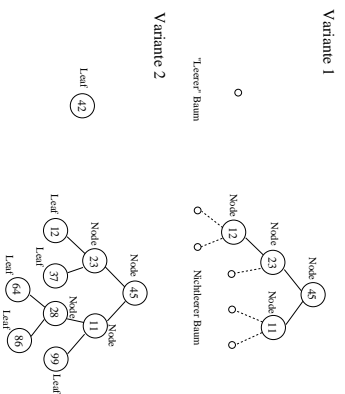
```
data BinTree1 = Nil | Node Int BinTree1 BinTree1
data BinTree2 = Leaf Int | Node Int BinTree2 BinTree2
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

20

## Rekursive Typen (4)

Veranschaulichung der Binärbaumvarianten 1&2 anhand eines Beispiels:



Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

21

## Rekursive Typen (5)

*Beispiele* ... für Funktionen über Binärbaumvariante 1.

```
valBinTree1 :: BinTree1 -> Int   == 0
valBinTree1 Nil                 == 0
valBinTree1 (Node n b1 b2)     = n + valBinTree1 b1 + valBinTree1 b2

depthBinTree1 :: BinTree1 -> Int
depthBinTree1 Nil = 0
depthBinTree1 (Node _ b1 b2)
  = 1 + max (depthBinTree1 b1) (depthBinTree1 b2)
```

Mit diesen Definitionen...

```
valBinTree1 Nil           == 0
valBinTree1 (Node 17 Nil (Node 4 Nil Nil)) == 21
depthBinTree1 (Node 17 Nil (Node 4 Nil Nil)) == 2
depthBinTree1 Nil       == 0
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

22

## Rekursive Typen (6)

*Beispiele* ... für Funktionen über Binärbaumvariante 2.

```
valBinTree2 :: BinTree2 -> Int
valBinTree2 (Leaf n)   = n
valBinTree2 (Node n b1 b2) = n + valBinTree2 b1 + valBinTree2 b2

depthBinTree2 :: BinTree2 -> Int
depthBinTree2 (Leaf _) = 1
depthBinTree2 (Node _ b1 b2)
  = 1 + max (depthBinTree2 b1) (depthBinTree2 b2)
```

Mit diesen Definitionen...

```
valBinTree2 (Leaf 3)           == 3
valBinTree2 (Node 17 (Leaf 4) (Node 4 (Leaf 12) (Leaf 5))) == 42
depthBinTree2 (Node 17 (Leaf 4) (Node 4 (Leaf 12) (Leaf 5))) == 3
depthBinTree2 (Leaf 3)       == 1
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

23

## Wechselweise rekursive Typen

...ein Spezialfall rekursiver Typen.

*Beispiel:*

```
data Individual = Adult Name Address Biography |
                Child Name

data Biography = Parent String [Individual] |
                NonParent String
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

24

## Typsynonyme (1)

...hätten wir bereits kennengelernt bei der Einführung von Tupeltypen:

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

...und auch in den Beispielen zu algebraischen Datentypen benutzt:

```
data Verkehrsmitel = Fahrrad Tandem |
                  Auto Hersteller Hubraum |
                  Schiff Name Tiefgang Heimathafen |
                  Flugzeug Spannweite Sitzplaeze

type Tandem      = Bool
type Hersteller  = String
type Hubraum     = Float
type Name        = String
type Tiefgang    = Float
type Heimathafen = String
type Reichweite  = Float
type Sitzplaeze  = Int
```

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

25

## Typsynonyme (2)

- Das Schlüsselwort `type` leitet die Deklaration von Typsynonymen ein

- Unbedingt zu beachten ist...

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

*Somit gilt:*

Durch `type`-Deklarationen eingeführte Typsynonyme...

- tragen zur Dokumentation bei und

- erleichtern (i.a.) das Programmverständnis

aber...

- führen nicht zu (zusätzlicher) Typsicherheit!

## Ein reales Beispiel

Anflugsteuerung einer Sonde zum Mars...

```
type Geschwindigkeit = Float
type Meilen           = Float
type Km               = Float
type Zeit             = Float
type Wegstrecke       = Meilen
type Distanz          = Km
```

```
geschwindigkeit :: Wegstrecke -> Zeit -> Geschwindigkeit
geschwindigkeit w z = (/) w z
```

```
verbleibendeFlugzeit :: Distanz -> Geschwindigkeit -> Zeit
verbleibendeFlugzeit d g = (/) d g
```

```
verbleibendeFlugzeit 18524.34 1523.79
```

...durch Typisierungsprobleme dieser Art ging vor einigen Jahren eine Marssonde im Wert von mehreren 100 Mill. USD verloren.

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

28

## Ein (pathologisches) Beispiel

```
type Euro = Float
type Yen  = Float
type Temperatur = Float
```

```
mpfi :: Float
dammen :: Float
maxTemp :: Float
mpfi = 3.14
dammen = 5.56
maxTemp = 43.2
```

```
currencyConverter :: Euro -> Yen
currencyConverter x = x + mpfi * dammen
```

Mit obigen Deklarationen...

```
currencyConverter maxTemp => 60.627
```

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

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

27

## Produkttypen vs. Tupeltypen (1)

Der Typ `Person` als...

- *Produkttyp*

```
data Person = Pers Name Geschlecht Alter
```

- *Tupeltyp*

```
type Person = (Name, Geschlecht, Alter)
```

*Vordergründiger Unterschied:*

...in der Tupeltypvariante fehlt der Konstruktor (in diesem Bsp.: `Pers`)

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

29

## Produkttypen vs. Tupeltypen (2)

...eine Abwägung von Vor- und Nachteilen.

*Produkttypen* und ihre typischen...

- *Vorteile* gegenüber *Tupeltypen*

– Objekte des Typs sind mit dem Konstruktor "markiert" (trägt zur Dokumentation bei)

– Tupel mit zufällig passenden Komponenten nicht irrtümlich als Elemente des Produkttyps manipulierbar (Typsicherheit! Vgl. früheres Beispiel zur Umrechnung Euro in Yen!)

– Aussagekräftigere (Typ-) Fehlermeldungen (Typsynonyme können wg. Expansion in Fehlermeldungen fehlen).

- *Nachteile* gegenüber *Tupeltypen*

– Produkttypen sind weniger kompakt, erfordern längere Definitionen (mehr Schreibarbeit)

– Auf Tupeln vorderfinierte polymorphe Funktionen (z.B. `fst`, `snd`, `zip`, `unzip`, ...) stehen nicht zur Verfügung.

– Der Code ist weniger effizient.

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

30

## Somit als kurzes Fazit... (1)

...unserer Überlegungen:

- *Typsynonyme* wie...

```
type Euro = Float
type Yen  = Float
type Temperatur = Float
```

...erben alle Operationen von `Float` und sind damit beliebig austauschbar – mit allen Annehmlichkeiten und Gefahren, sprich Fehlerquellen.

- *Produkttypen* wie...

```
data Euro = EUR Float
data Yen  = YEN Float
data Temperatur = Temp Float
```

...erben keinerlei Operationen von `Float`, bieten dafür aber um den Preis zusätzlicher Schreibarbeit und gewissen Performanzverlusts Typsicherheit!

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

31

---

## Somit als kurzes Fazit... (2)

In ähnlicher Weise...

```
data Miles      = Mi Float
data Km         = Km Float
type Distance   = Miles
type Wegstrecke = Km
```

...

...wäre auch der Verlust der Marssonde vermutlich vermeidbar gewesen.

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

33

---

## Spezialitäten

...die `newtype`-Deklaration:

```
newtype Miles = Mi Float
```

`newtype`-Deklarationen sind im Hinblick auf...

- Typsicherheit  
...mit `data`-Deklarationen vergleichbar
- Effizienz  
...mit `type`-Deklarationen vergleichbar

*Beachte:* `newtype`-Deklarationen sind auf Typen mit nur einem Konstruktor eingeschränkt.

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

34

---

## Vorschau auf die kommenden Aufgabenblätter...

Ausgabe des...

- vierten Aufgabenblatts: Di, 30.10.2007  
...Abgabetermine: Di, den 06.11.2007, und Di, den 13.11.2007, jeweils 15:00 Uhr
- fünften Aufgabenblatts: Di, den 06.11.2007  
...Abgabetermine: Di, den 13.11.2007, und Di, den 20.11.2007, jeweils 15:00 Uhr
- sechsten Aufgabenblatts: Di, den 13.11.2007  
...Abgabetermine: Di, den 20.11.2007, und Di, den 27.11.2007, jeweils 15:00 Uhr

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

36

---

## Polymorphe Typen

...demnächst!

Funktionale Programmierung (WS 2007/2008) / 4. Teil (30.10.2007)

35

---

## Vorschau auf die nächsten Vorlesungstermine...

- Do, 01.11.2006: *Keine Vorlesung!* (Allerheiligen)
- Do, 08.11.2007, Vorlesung von 16:30 Uhr bis 18:00 Uhr im Radlinger-Hörsaal
- Do, 15.11.2007: *Keine Vorlesung!* (Tag des Landespatriots, TU-weit LVA-frei)
- Di, 20.11.2007, Vorlesung von 13:00 Uhr bis 14:00 Uhr im Informatik-Hörsaal
- Do, 22.11.2007: *Keine Vorlesung!* (epilog, Diplomarbetspräsentation, FR-weit LVA-frei ab 15:00 Uhr)
- Di, 27.11.2007, Vorlesung von 13:00 Uhr bis 14:00 Uhr im Informatik-Hörsaal
- Do, 29.11.2007, Vorlesung von 16:30 Uhr bis 18:00 Uhr im Radlinger-Hörsaal