

PROGRAMMIERUNG

ÜBUNG 4: TYPPOLYMORPHIE & UNIFIKATION

Eric Kunze

`eric.kunze@mailbox.tu-dresden.de`

1. Funktionale Programmierung
 - 1.1 Einführung in Haskell: Listen
 - 1.2 Algebraische Datentypen
 - 1.3 Funktionen höherer Ordnung
 - 1.4 **Typpolymorphie & Unifikation**
 - 1.5 Beweis von Programmeigenschaften
 - 1.6 λ – Kalkül
2. Logikprogrammierung
3. Implementierung einer imperativen Programmiersprache
4. Verifikation von Programmeigenschaften
5. H_0 – ein einfacher Kern von Haskell

Typpolymorphie

- ▶ **bisher:** Funktionen mit konkreten Datentypen
z.B. `length :: [Int] -> Int`
- ▶ **Problem:** Funktion würde auch auf anderen Datentypen funktionieren
z.B. `length :: [Bool] -> Int`
- ▶ **Lösung:** Typvariablen und polymorphe Funktionen
z.B. `length :: [a] -> Int`

Bei konkreter Instanziierung wird Typvariable an entsprechenden Typbezeichner gebunden (z.B. `a = Int` oder `a = Bool`).

Übungsblatt 4

Aufgabe 1

AUFGABE 1 — TEIL (A)

Beispielbaum mit mindestens 5 Blättern

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
```

AUFGABE 1 — TEIL (A)

Beispielbaum mit mindestens 5 Blättern

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
```

```
1 testTree :: BinTree Int
2 testTree = Branch 5
3             (Leaf 1)
4             (Branch 12
5                 (Branch 4
6                     (Leaf 0)
7                     (Leaf 0))
8                 (Branch 12
9                     (Leaf 0)
10                    (Leaf 1)))
```

minimale Tiefe eines Baumes

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
depth :: BinTree a -> Int
```


AUFGABE 1 — TEIL (B)

minimale Tiefe eines Baumes

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
depth :: BinTree a -> Int
```

```
1  -- Aufgabe 1 (b)
2  depth :: BinTree a -> Int
3  depth (Leaf _ ) = 1
4  depth (Branch _ l r) = 1 + min (depth l) (depth r)
```

Baum mit Beschriftungsfolge neu beschriften

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
paths :: BinTree a -> BinTree [a]
```

AUFGABE 1 — TEIL (C)

Baum mit Beschriftungsfolge neu beschriften

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
paths :: BinTree a -> BinTree [a]
```

```
1  -- Aufgabe 1 (c)
2  paths :: BinTree a -> BinTree [a]
3  paths = go []
4      where
5      go :: [a] -> BinTree a -> BinTree [a]
6      go prefix (Leaf x) = Leaf (prefix ++ [x])
7      go prefix (Branch x l r)
8          = Branch (prefix ++ [x])
9                  (go (prefix ++ [x]) l)
10                 (go (prefix ++ [x]) r)
```

Map für Bäume

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
tmap :: (a -> b) -> BinTree a -> BinTree b
```

Map für Bäume

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
tmap :: (a -> b) -> BinTree a -> BinTree b
```

```
1  -- Aufgabe 1 (d)
2  tmap :: (a -> b) -> BinTree a -> BinTree b
3  tmap f (Leaf x) = Leaf (f x)
4  tmap f (Branch x l r) = Branch (f x) (tmap f l) (tmap
    f r)
```

Übungsblatt 4

Aufgabe 2

AUFGABE 2 — TEIL (A)

Liste von Paaren \rightarrow Paare von Listen

`unpairs :: [(a,b)] -> ([a], [b])`

AUFGABE 2 — TEIL (A)

Liste von Paaren \rightarrow Paare von Listen

`unpairs :: [(a,b)] -> ([a], [b])`

```
1  -- Aufgabe 2 (a)
2  unpairs :: [(a, b)] -> ([a], [b])
3  unpairs []           = ([], [])
4  unpairs ((a, b):xs) = let (as, bs) = unpairs xs
5                        in (a:as, b:bs)
```


AUFGABE 2 — TEIL (B)

```
1 map :: (a -> b) -> [a] -> [b]
2 map _ []          = []
3 map f (x : xs) = f x : map f xs
4
5 uncurry :: (a -> b -> c) -> (a, b) -> c
6 uncurry f (x, y) = f x y
```

```
map (uncurry (+)) [(1,2), (3,4)]
```

AUFGABE 2 — TEIL (B)

```
1 map :: (a -> b) -> [a] -> [b]
2 map _ []          = []
3 map f (x : xs) = f x : map f xs
4
5 uncurry :: (a -> b -> c) -> (a, b) -> c
6 uncurry f (x, y) = f x y
```

```
map (uncurry (+)) [(1,2), (3,4)]
= map (uncurry (+)) ((1,2):[(3,4)])
= uncurry (+) (1,2) : map (uncurry (+)) [(3,4)]
= (1 + 2) : map (uncurry (+)) [(3,4)]
= 3 : map (uncurry (+)) [(3,4)]
= 3 : map (uncurry (+)) ((3,4);[])
= 3 : uncurry (+) (3,4) : map (uncurry (+)) []
= 3 : (3 + 4) : map (uncurry (+)) []
= 3 : 7 : map (uncurry (+)) []
= 3 : 7 : []
= [3, 7]
```

AUFGABE 2 — TEIL (B)

```
1 map :: (a -> b) -> [a] -> [b]
2 map _ []          = []
3 map f (x : xs) = f x : map f xs
4
5 uncurry :: (a -> b -> c) -> (a, b) -> c
6 uncurry f (x, y) = f x y
```

```
map (uncurry (+)) [(1,2), (3,4)]
= map (uncurry (+)) ((1,2):[(3,4)])
= uncurry (+) (1,2) : map (uncurry (+)) [(3,4)]
= (1 + 2) : map (uncurry (+)) [(3,4)]
= 3 : map (uncurry (+)) [(3,4)]
= 3 : map (uncurry (+)) ((3,4);[])
= 3 : uncurry (+) (3,4) : map (uncurry (+)) []
= 3 : (3 + 4) : map (uncurry (+)) []
= 3 : 7 : map (uncurry (+)) []
= 3 : 7 : []
= [3, 7]
```

Unifikation & Unifikationsalgorithmus

Motivation: Typüberprüfung

```
1 f :: (t, Char) -> (t, [Char])
2 f (...) = ...
3
4 g :: (Int, [u]) -> Int
5 g (...) = ...
6
7 h = g . f
```

Typausdrücke

- ▶ `Int`, `Bool`, `Float`, `Char`, `String`
- ▶ Typvariablen
- ▶ Listen, Tupel, Funktionen

Typterme

- ▶ Übersetzung *trans*: Typausdruck \rightarrow Typterme
- ▶ z.B.

$$\text{trans}((t, [\text{Char}])) = ()^2(t, [](\text{Char}))$$

$$\text{trans}((\text{Int}, [u])) = ()^2(\text{Int}, [](u))$$

Typausdrücke

- ▶ `Int`, `Bool`, `Float`, `Char`, `String`
- ▶ Typvariablen
- ▶ Listen, Tupel, Funktionen

Typterme

- ▶ Übersetzung *trans*: Typausdruck \rightarrow Typterme
- ▶ z.B.

$$\text{trans}((t, [\text{Char}])) = ()^2(t, [](\text{Char}))$$

$$\text{trans}((\text{Int}, [u])) = ()^2(\text{Int}, [](u))$$

Beide Typausdrücke können in Übereinstimmung gebracht werden, wenn die Typterme $\text{trans}((t, [\text{Char}]))$ und $\text{trans}((\text{Int}, [u]))$ unifizierbar sind

Typausdrücke

- ▶ `Int`, `Bool`, `Float`, `Char`, `String`
- ▶ Typvariablen
- ▶ Listen, Tupel, Funktionen

Typterme

- ▶ Übersetzung *trans*: Typausdruck \rightarrow Typterme
- ▶ z.B.

$$\text{trans}((t, [\text{Char}])) = ()^2(t, [](\text{Char}))$$

$$\text{trans}((\text{Int}, [u])) = ()^2(\text{Int}, [](u))$$

Beide Typausdrücke können in Übereinstimmung gebracht werden, wenn die Typterme $\text{trans}((t, [\text{Char}]))$ und $\text{trans}((\text{Int}, [u]))$ unifizierbar sind $\rightarrow t = \text{Int}$ und $u = \text{Char}$

UNIFIKATIONSALGORITHMUS

- **gegeben:** zwei Typterme t_1, t_2
- **Ziel:** entscheide, ob t_1 und t_2 unifizierbar sind

Wir notieren die beiden Typterme als Spalten:

$$\begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \quad \text{bzw.} \quad \begin{pmatrix} ()^2(t, [](\text{Char})) \\ ()^2(\text{Int}, [](u)) \end{pmatrix}$$

Unifikationsalgorithmus erstellt eine Folge von Mengen M_i , wobei die M_{i+1} aus M_i hervorgeht, indem eine der vier Regeln angewendet wird.

$$M_1 := \left\{ \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \right\} \quad \text{bzw.} \quad M_1 := \left\{ \begin{pmatrix} ()^2(t, [](\text{Char})) \\ ()^2(\text{Int}, [](u)) \end{pmatrix} \right\}$$

- **Dekomposition.** Sei $\delta \in \Sigma$ ein k -stelliger Konstruktor, $s_1, \dots, s_k, t_1, \dots, t_k$ Terme über Konstruktoren und Variablen.

$$\begin{pmatrix} \delta(s_1, \dots, s_k) \\ \delta(t_1, \dots, t_k) \end{pmatrix} \rightsquigarrow \begin{pmatrix} s_1 \\ t_1 \end{pmatrix}, \dots, \begin{pmatrix} s_k \\ t_k \end{pmatrix}$$

- **Elimination.** Sei x eine *Variable* !

$$\begin{pmatrix} x \\ x \end{pmatrix} \rightsquigarrow \emptyset$$

- **Vertauschung.** Sei t *keine* Variable.

$$\begin{pmatrix} t \\ x \end{pmatrix} \rightsquigarrow \begin{pmatrix} x \\ t \end{pmatrix}$$

- **Substitution.** Sei x eine Variable, t keine Variable und x kommt nicht in t vor (occur check). Dann ersetze in jedem anderen Term die Variable x durch t .

Ende: keine Regel mehr anwendbar – Entscheidung:

- ▶ t_1, t_2 **unifizierbar:** M ist von der Form

$$\left\{ \begin{pmatrix} u_1 \\ t_1 \end{pmatrix}, \begin{pmatrix} u_2 \\ t_2 \end{pmatrix}, \dots, \begin{pmatrix} u_k \\ t_k \end{pmatrix} \right\} \quad \begin{array}{l} \text{„Variablen“} \\ \text{„Terme ohne Variablen“} \end{array}$$

wobei u_1, u_2, \dots, u_k paarweise verschiedene Variablen sind und nicht in t_1, t_2, \dots, t_k vorkommen.

allgemeinster Unifikator φ :

$$\varphi(u_i) = t_i \quad (i = 1, \dots, k)$$

$$\varphi(x) = x \quad \text{für alle nicht vorkommenden Variablen}$$

- ▶ t_1, t_2 **sind nicht unifizierbar:** M hat nicht diese Form und keine Regel ist anwendbar

Übungsblatt 4

Aufgabe 3

AUFGABE 3

$$\begin{aligned}
 & \left\{ \left(\delta(\alpha, \sigma(x_1, \alpha), \sigma(x_2, x_3)), \delta(\alpha, \sigma(x_1, x_2), \sigma(x_2, \gamma(x_2))) \right) \right\} \\
 \xRightarrow{\text{Dek.}} & \left\{ \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}, \begin{pmatrix} \sigma(x_1, \alpha) \\ \sigma(x_1, x_2) \end{pmatrix}, \begin{pmatrix} \sigma(x_2, x_3) \\ \sigma(x_2, \gamma(x_2)) \end{pmatrix} \right\} \\
 \xRightarrow{\text{Dek.}}^3 & \left\{ \begin{pmatrix} x_1 \\ x_1 \end{pmatrix}, \begin{pmatrix} \alpha \\ x_2 \end{pmatrix}, \begin{pmatrix} x_2 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_3 \\ \gamma(x_2) \end{pmatrix} \right\} \\
 \xRightarrow{\text{El.}}^2 & \left\{ \begin{pmatrix} \alpha \\ x_2 \end{pmatrix}, \begin{pmatrix} x_3 \\ \gamma(x_2) \end{pmatrix} \right\} \\
 \xRightarrow{\text{Vert.}} & \left\{ \begin{pmatrix} x_2 \\ \alpha \end{pmatrix}, \begin{pmatrix} x_3 \\ \gamma(x_2) \end{pmatrix} \right\} \\
 \xRightarrow{\text{Subst.}} & \left\{ \begin{pmatrix} x_2 \\ \alpha \end{pmatrix}, \begin{pmatrix} x_3 \\ \gamma(\alpha) \end{pmatrix} \right\}
 \end{aligned}$$

allgemeinster Unifikator:

$$x_1 \mapsto x_1 \quad x_2 \mapsto \alpha \quad x_3 \mapsto \gamma(\alpha)$$

allgemeinster Unifikator:

$$X_1 \mapsto X_1 \quad X_2 \mapsto \alpha \quad X_3 \mapsto \gamma(\alpha)$$

Teilaufgabe (b)

$$t_1 = (a, [a])$$

$$t_2 = (\text{Int}, [\text{Double}])$$

$$t_3 = (b, c)$$

allgemeinster Unifikator:

$$x_1 \mapsto x_1 \quad x_2 \mapsto \alpha \quad x_3 \mapsto \gamma(\alpha)$$

Teilaufgabe (b)

$$t_1 = (a, [a])$$

$$t_2 = (\text{Int}, [\text{Double}])$$

$$t_3 = (b, c)$$

- ▶ t_1 und t_2 sind *nicht* unifizierbar
- ▶ t_1 und t_3 sind unifizierbar mit
$$a \mapsto a \quad b \mapsto a \quad c \mapsto [a]$$
- ▶ t_2 und t_3 sind unifizierbar mit $b \mapsto \text{Int} \quad c \mapsto [\text{Double}]$

Übungsblatt 4

Aufgabe 4

AUFGABE 4

$$\begin{aligned}
 & \left\{ \begin{pmatrix} \sigma(\sigma(x_1, \alpha), \sigma(\gamma(x_3), x_3)) \\ \sigma(\sigma(\gamma(x_2), \alpha), \sigma(x_2, x_3)) \end{pmatrix} \right\} \\
 \xRightarrow{\text{Dek.}} & \left\{ \begin{pmatrix} \sigma(x_1, \alpha) \\ \sigma(\gamma(x_2), \alpha) \end{pmatrix}, \begin{pmatrix} \sigma(\gamma(x_3), x_3) \\ \sigma(x_2, x_3) \end{pmatrix} \right\} \\
 \xRightarrow[2]{\text{Dek.}} & \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}, \begin{pmatrix} \gamma(x_3) \\ x_2 \end{pmatrix}, \begin{pmatrix} x_3 \\ x_3 \end{pmatrix} \right\} \\
 \xRightarrow{\text{El.}} & \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}, \begin{pmatrix} \gamma(x_3) \\ x_2 \end{pmatrix} \right\} \\
 \xRightarrow{\text{Dek.}} & \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} \gamma(x_3) \\ x_2 \end{pmatrix} \right\} \\
 \xRightarrow{\text{Vert.}} & \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} x_2 \\ \gamma(x_3) \end{pmatrix} \right\} \quad x_2 \text{ kommt nicht in } \gamma(x_3) \text{ vor} \\
 \xRightarrow{\text{Subst.}} & \left\{ \begin{pmatrix} x_1 \\ \gamma(\gamma(x_3)) \end{pmatrix}, \begin{pmatrix} x_2 \\ \gamma(x_3) \end{pmatrix} \right\}
 \end{aligned}$$

allgemeinster Unifikator:

$$x_1 \mapsto \gamma(\gamma(x_3)) \quad x_2 \mapsto \gamma(x_3) \quad x_3 \mapsto x_3$$

AUFGABE 4

allgemeinster Unifikator:

$$x_1 \mapsto \gamma(\gamma(x_3)) \quad x_2 \mapsto \gamma(x_3) \quad x_3 \mapsto x_3$$

weitere Unifikatoren:

$$\begin{array}{lll} x_1 \mapsto \gamma(\gamma(\alpha)) & x_2 \mapsto \gamma(\alpha) & x_3 \mapsto \alpha \\ x_1 \mapsto \gamma(\gamma(\gamma(\alpha))) & x_2 \mapsto \gamma(\gamma(\alpha)) & x_3 \mapsto \gamma(\alpha) \end{array}$$

AUFGABE 4

allgemeinster Unifikator:

$$x_1 \mapsto \gamma(\gamma(x_3)) \quad x_2 \mapsto \gamma(x_3) \quad x_3 \mapsto x_3$$

weitere Unifikatoren:

$$\begin{array}{lll} x_1 \mapsto \gamma(\gamma(\alpha)) & x_2 \mapsto \gamma(\alpha) & x_3 \mapsto \alpha \\ x_1 \mapsto \gamma(\gamma(\gamma(\alpha))) & x_2 \mapsto \gamma(\gamma(\alpha)) & x_3 \mapsto \gamma(\alpha) \end{array}$$

Fehlschlag beim occur-check:

Alphabet: $\Sigma = \{\gamma^{(1)}\}$

$$t_1 = x_1$$

$$t_2 = \gamma(x_1)$$

Fragen?