

PROGRAMMIERUNG

ÜBUNG 4: TYPPOLYMORPHIE & UNIFIKATION

Eric Kunze

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

1. Funktionale Programmierung
 - 1.1 Einführung in Haskell
 - 1.2 Listen & 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

Aufgabe 1

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

- ▶ **bisher:** Funktionen mit konkreten Datentypen
z.B. `length :: [Int] -> Int`
- ▶ **Problem:** Funktion würde auch auf anderen Datentypen funktionieren
z.B. `length :: [Bool] -> Int` oder `length :: String -> 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`).

- ▶ Der Aufruf `length [1,5,2,7]` liefert für die Typvariable `a = Int`.
- ▶ Der Aufruf `length [True, False, True, True, False]` liefert die Belegung `a = Bool`.
- ▶ Der Aufruf `length "hello"` impliziert `a = Char`.

AUFGABE 1 — TEIL (A)

Beispielbaum mit mindestens 5 Blättern

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

Beispielbaum mit mindestens 5 Blättern

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

```
testTree :: BinTree Int
testTree = Branch 5
           (Leaf 1)
           (Branch 12
            (Branch 4
             (Leaf 0)
             (Leaf 0))
            (Branch 12
             (Leaf 0)
             (Leaf 1)))
```

minimale Tiefe eines Baumes

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

minimale Tiefe eines Baumes

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

```
depth :: BinTree a -> Int
depth (Leaf _ _) = 1
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]
```

Baum mit Beschriftungsfolge neu beschriften

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

```
paths :: BinTree a -> BinTree [a]
paths = go []
  where
    go :: [a] -> BinTree a -> BinTree [a]
    go prefix (Leaf x) = Leaf (prefix ++ [x])
    go prefix (Branch x l r)
      = Branch (prefix ++ [x])
                (go (prefix ++ [x]) l)
                (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
```

```
tmap :: (a -> b) -> BinTree a -> BinTree b
tmap f (Leaf x) = Leaf (f x)
tmap f (Branch x l r) = Branch (f x) (tmap f l) (tmap f r)
```

EINSCHUB: AUSWERTUNG EINES FUNKTIONSAUFRUFS

```
map :: (a -> b) -> [a] -> [b]
map _ []          = []
map f (x : xs) = f x : map f xs
```

```
uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (x, y) = f x y
```

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

EINSCHUB: AUSWERTUNG EINES FUNKTIONSAUFRUFS

```
map :: (a -> b) -> [a] -> [b]
map _ []          = []
map f (x : xs) = f x : map f xs
```

```
uncurry :: (a -> b -> c) -> (a, b) -> c
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

Aufgabe 2

Motivation: Typüberprüfung

```
f :: (t, Char) -> (t, [Char])
```

```
f (...) = ...
```

```
g :: (Int, [u]) -> Int
```

```
g (...) = ...
```

```
h = g . f
```

Motivation: Typüberprüfung

```
f :: (t, Char) -> (t, [Char])  
f (...) = ...  
  
g :: (Int, [u]) -> Int  
g (...) = ...  
  
h = g . f
```

Wie müssen die Typvariablen t und u belegt werden, damit die Funktion h wohldefiniert ist, d.h. damit die Ergebnisse aus f wirklich in g eingesetzt werden dürfen?

Ziel: theoretischere Form von Typausdrücken

Typausdrücke

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

Typterme

- ▶ Übersetzung *trans*: Typausdruck \rightarrow Typterm

Ziel: theoretischere Form von Typausdrücken

Typausdrücke

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

Typterme

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

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

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

Ziel: theoretischere Form von Typausdrücken

Typausdrücke

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

Typterme

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

$$\mathit{trans}(\langle \mathit{t}, [\mathit{Char}] \rangle) = ()^2(\mathit{t}, [](\mathit{Char}))$$

$$\mathit{trans}(\langle \mathit{Int}, [\mathit{u}] \rangle) = ()^2(\mathit{Int}, [](\mathit{u}))$$

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

Ziel: theoretischere Form von Typausdrücken

Typausdrücke

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

Typterme

- ▶ Übersetzung *trans*: Typausdruck \rightarrow Typterm
- ▶ 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 Spalte:

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

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

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

UNIFIKATIONSALGORITHMUS – REGELN

- ▶ **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.

Occur Check: x kommt nicht in t vor

Dann ersetze in jedem anderen Term die Variable x durch t .

$$\begin{pmatrix} x \\ t \end{pmatrix}, \begin{pmatrix} y \\ s(x) \end{pmatrix} \rightsquigarrow \begin{pmatrix} x \\ t \end{pmatrix}, \begin{pmatrix} y \\ s(t) \end{pmatrix}$$

UNIFIKATIONSALGORITHMUS

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

OCCUR CHECK

Um endlose Rekursionen zu unterbinden, benötigen die Regeln zum Vertauschen und zur Substitution gewisse Einschränkungen.

Occur Check: Gegeben sei ein Termpaar $(\frac{x}{t})$, wobei x eine Variable und t ein Typterm sei.

- ▶ Kommt x in t vor, dann schlägt der Check fehl.
- ▶ Kommt x nicht in t vor, dann ist der Check okay.

Beispiel:

- ▶ $(\frac{x_1}{\gamma(x_1)}) \rightsquigarrow$ Fehlschlag, da x_1 in $\gamma(x_1)$ vorkommt
- ▶ $(\frac{x_1}{\gamma(x_2)}) \rightsquigarrow$ okay, da x_1 nicht in $\gamma(x_2)$ vorkommt

Was passiert, wenn wir substituieren obwohl der Occur Check fehlschlägt?

$$\left(\frac{x_1}{\gamma(x_1)}\right), \left(\frac{x_2}{\gamma(x_1)}\right) \Rightarrow \left(\frac{x_1}{\gamma(x_1)}\right), \left(\frac{x_2}{\gamma(\gamma(x_1))}\right) \Rightarrow \left(\frac{x_1}{\gamma(x_1)}\right), \left(\frac{x_2}{\gamma(\gamma(\gamma(x_1)))}\right)$$

AUFGABE 3

$$\begin{aligned}
 & \left\{ \left(\begin{array}{l} \sigma(\sigma(x_1, \alpha), \sigma(\gamma(x_3), x_3)) \\ \sigma(\sigma(\gamma(x_2), \alpha), \sigma(x_2, x_3)) \end{array} \right) \right\} \\
 \xRightarrow{\text{Dek.}} & \left\{ \left(\begin{array}{l} \sigma(x_1, \alpha) \\ \sigma(\gamma(x_2), \alpha) \end{array} \right), \left(\begin{array}{l} \sigma(\gamma(x_3), x_3) \\ \sigma(x_2, x_3) \end{array} \right) \right\} \\
 \xRightarrow{\text{Dek.}_2} & \left\{ \left(\begin{array}{l} x_1 \\ \gamma(x_2) \end{array} \right), \left(\begin{array}{l} \alpha \\ \alpha \end{array} \right), \left(\begin{array}{l} \gamma(x_3) \\ x_2 \end{array} \right), \left(\begin{array}{l} x_3 \\ x_3 \end{array} \right) \right\} \\
 \xRightarrow{\text{El.}} & \left\{ \left(\begin{array}{l} x_1 \\ \gamma(x_2) \end{array} \right), \left(\begin{array}{l} \alpha \\ \alpha \end{array} \right), \left(\begin{array}{l} \gamma(x_3) \\ x_2 \end{array} \right) \right\} \\
 \xRightarrow{\text{Dek.}} & \left\{ \left(\begin{array}{l} x_1 \\ \gamma(x_2) \end{array} \right), \left(\begin{array}{l} \gamma(x_3) \\ x_2 \end{array} \right) \right\} \\
 \xRightarrow{\text{Vert.}} & \left\{ \left(\begin{array}{l} x_1 \\ \gamma(x_2) \end{array} \right), \left(\begin{array}{l} x_2 \\ \gamma(x_3) \end{array} \right) \right\} \quad x_2 \text{ kommt nicht in } \gamma(x_3) \text{ vor} \\
 \xRightarrow{\text{Subst.}} & \left\{ \left(\begin{array}{l} x_1 \\ \gamma(\gamma(x_3)) \end{array} \right), \left(\begin{array}{l} x_2 \\ \gamma(x_3) \end{array} \right) \right\}
 \end{aligned}$$

AUFGABE 3

(a) **allgemeinster Unifikator:**

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

AUFGABE 3

(a) **allgemeinster Unifikator:**

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

(b) **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 3

(a) **allgemeinster Unifikator:**

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

(b) **weitere Unifikatoren:**

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

(c) **Fehlschlag beim occur-check:**

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

$$t_1 = x_1$$

$$t_2 = \gamma(x_1)$$

AUFGABE 3 — TEIL (D)

$t_1 = (a, [a])$

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

$t_3 = (b, c)$

AUFGABE 3 — TEIL (D)

$t_1 = (a, [a])$

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

$t_3 = (b, c)$

- ▶ t_1 und t_2 sind
- ▶ t_1 und t_3 sind
- ▶ t_2 und t_3 sind

AUFGABE 3 — TEIL (D)

$t_1 = (a, [a])$

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

$t_3 = (b, c)$

- ▶ t_1 und t_2 sind *nicht* unifizierbar
- ▶ t_1 und t_3 sind
- ▶ t_2 und t_3 sind

AUFGABE 3 — TEIL (D)

$t_1 = (a, [a])$

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

$t_3 = (b, c)$

- ▶ t_1 und t_2 sind *nicht* unifizierbar
- ▶ t_1 und t_3 sind unifizierbar mit $a \mapsto a, b \mapsto a, c \mapsto [a]$
- ▶ t_2 und t_3 sind

AUFGABE 3 — TEIL (D)

$t_1 = (a, [a])$

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

$t_3 = (b, c)$

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

Fragen?