PROGRAMMIERUNG

ÜBUNG 4: TYPPOLYMORPHIE & UNIFIKATION

Eric Kunze
eric.kunze@mailbox.tu-dresden.de

INHALT

Aufgabe 1

Typpolymorphie

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`).

- 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`.

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
```

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

AUFGABE 1 — TEIL (B)

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
```

```
depth :: BinTree a -> Int
depth (Leaf   _    ) = 1
depth (Branch _ l r) = 1 + min (depth l) (depth r)
```

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]
```

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)
```

Aufgabe 2

Paare

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])
```

```
unpairs :: [(a, b)] -> ([a], [b])
unpairs []          = ([], [])
unpairs ((a, b):xs) = let (as, bs) = unpairs xs
                      in (a:as, b:bs)
```

AUFGABE 2 — TEIL (B)

```
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)]`

AUFGABE 2 — TEIL (B)

```
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 : []
```

Aufgabe 3

Unifikation & Unifikationsalgorithmus

UNIFIKATION

Motivation: Typüberprüfung

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

g :: (Int, [u]) -> Int
g (...) = ...

h = g . f
```

UNIFIKATION

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?

UNIFIKATION

Typausdrücke

- `Int`, `Bool`, `Floar`, `Char`, `String`
- Typvariablen
- Listen, Tupel, Funktionen

Typterme

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

$$trans(\texttt{(t,[Char])}) = ()^2(t, [](Char))$$
$$trans(\texttt{(Int, [u])}) = ()^2(Int, [](u))$$

UNIFIKATION

Typausdrücke

- `Int`, `Bool`, `Floar`, `Char`, `String`
- Typvariablen
- Listen, Tupel, Funktionen

Typterme

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

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

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

Beide Typausdrücke können in Übereinstimmung gebracht werden, wenn die Typterme *trans*(`(t,[Char])`) und *trans*(`(Int, [u])`) unifizierbar sind.

UNIFIKATION

Typausdrücke

- `Int`, `Bool`, `Floar`, `Char`, `String`
- Typvariablen
- Listen, Tupel, Funktionen

Typterme

- Übersetzung *trans*: Typausdruck $\to$ Typterm
- z.B.

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

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

Beide Typausdrücke können in Übereinstimmung gebracht werden, wenn die Typterme *trans*(`(t,[Char])`) und *trans*(`(Int, [u])`) unifizierbar sind.
$\to$ `t = Int` und `u = Char`

UNIFIKATIONSALGORITHMUS

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

Wir notieren die beiden Typterme als Spalten:

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

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

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

UNIFIKATIONSALGORITHMUS – REGELN

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

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

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

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

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

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

- **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.

UNIFIKATIONSALGORITHMUS

Ende: keine Regel mehr anwendbar – Entscheidung:

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

$$\left\{ \binom{u_1}{t_1}, \binom{u_2}{t_2}, \ldots, \binom{u_k}{t_k} \right\} \qquad \begin{array}{c} \textit{„Variablen“} \\ \textit{„Terme ohne Variablen“} \end{array}$$

  wobei $u_1, u_2, \ldots, u_k$ paarweise verschiedene Variablen sind und nicht in $t_1, t_2, \ldots, t_k$ vorkommen.
  allgemeinster Unifikator φ:

$$\begin{aligned} \varphi(u_i) &= t_i && (i = 1, \ldots, k) \\ \varphi(x) &= x && \text{für alle nicht vorkommenden Variablen} \end{aligned}$$

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

AUFGABE 3

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

AUFGABE 3

allgemeinster Unifikator:

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

AUFGABE 3

allgemeinster Unifikator:

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

Teilaufgabe (b)

$$t_1 = (\texttt{a}\ ,\ [\texttt{a}])$$
$$t_2 = (\texttt{Int}\ ,\ [\texttt{Double}])$$
$$t_3 = (\texttt{b}\ ,\ \texttt{c})$$

AUFGABE 3

allgemeinster Unifikator:

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

Teilaufgabe (b)

$$t_1 = (\texttt{a} \;,\; [\texttt{a}])$$
$$t_2 = (\texttt{Int} \;,\; [\texttt{Double}])$$
$$t_3 = (\texttt{b} \;,\; \texttt{c})$$

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

AUFGABE 4

$$\left\{ \begin{pmatrix} \sigma(\sigma(\ \ x_1\ \ ,\alpha),\sigma(\gamma(x_3),x_3)) \\ \sigma(\sigma(\gamma(x_2),\alpha),\sigma(\ \ x_2\ \ ,x_3)) \end{pmatrix} \right\}$$

$$\xRightarrow{\text{Dek.}} \left\{ \begin{pmatrix} \sigma(\ \ x_1\ \ ,\alpha) \\ \sigma(\gamma(x_2),\alpha) \end{pmatrix}, \begin{pmatrix} \sigma(\gamma(x_3),x_3)) \\ \sigma(\ \ x_2\ \ ,x_3)) \end{pmatrix} \right\}$$

$$\xRightarrow{\text{Dek.}}{}^{2} \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}, \begin{pmatrix} \gamma(x_3) \\ x_2 \end{pmatrix}, \begin{pmatrix} x_3 \\ x_3 \end{pmatrix} \right\}$$

$$\xRightarrow{\text{El.}} \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}, \begin{pmatrix} \gamma(x_3) \\ x_2 \end{pmatrix} \right\}$$

$$\xRightarrow{\text{Dek.}} \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} \gamma(x_3) \\ x_2 \end{pmatrix} \right\}$$

$$\xRightarrow{\text{Vert.}} \left\{ \begin{pmatrix} x_1 \\ \gamma(x_2) \end{pmatrix}, \begin{pmatrix} x_2 \\ \gamma(x_3) \end{pmatrix} \right\}$$

x_2 kommt nicht in $\gamma(x_3)$ vor

$$\xRightarrow{\text{Subst.}} \left\{ \begin{pmatrix} x_1 \\ \gamma(\gamma(x_3)) \end{pmatrix}, \begin{pmatrix} x_2 \\ \gamma(x_3) \end{pmatrix} \right\}$$

AUFGABE 4

allgemeinster Unifikator:

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

AUFGABE 4

allgemeinster Unifikator:

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

weitere Unifikatoren:

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

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

AUFGABE 4

allgemeinster Unifikator:

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

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

$$t_1 = x_1$$
$$t_2 = \gamma(x_1)$$

ENDE

Fragen?