

PROGRAMMIERUNG

ÜBUNG 5: UNIFIKATION & INDUKTION AUF LISTEN

Eric Kunze

`eric.kunze@tu-dresden.de`

TU Dresden, 11. Mai 2022

letzte Änderung:
11.05.2022, 09:32

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

Unifikation & Unifikationsalgorithmus

Aufgabe 1

ERINNERUNG: TYPOLYMPHIE

- ▶ **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`.

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?

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.

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

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

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

Weitere Unifikatoren ψ erhält man durch Anwendung einer Substitution σ , sodass $\psi = \sigma \circ \varphi$.

OCCUR CHECK

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

Occur Check: Gegeben sei ein Termpaar $(\overset{x}{t})$, 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:

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

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

$$\left(\overset{x_1}{\gamma(x_1)} \right), \left(\overset{x_2}{\gamma(\textcolor{brown}{x}_1)} \right) \Rightarrow \left(\overset{x_1}{\gamma(x_1)} \right), \left(\overset{x_2}{\gamma(\gamma(\textcolor{brown}{x}_1))} \right) \Rightarrow \left(\overset{x_1}{\gamma(x_1)} \right), \left(\overset{x_2}{\gamma(\gamma(\gamma(\textcolor{brown}{x}_1)))} \right)$$

AUFGABE 1

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

AUFGABE 1

(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:**

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

$$t_1 = x_1$$

$$t_2 = \gamma(x_1)$$

AUFGABE 1 — 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 $b \mapsto \text{Int}, c \mapsto [\text{Double}]$

Induktionsbeweise

Aufgabe 2

VOLLSTÄNDIGE INDUKTION AUF \mathbb{N}

Definition: natürliche Zahlen $\mathbb{N} := \{0, 1, \dots\}$

Basisfall: $0 \in \mathbb{N}$

Rekursionsfall: $x + 1 \in \mathbb{N}$ für $x \in \mathbb{N}$

Beweis von Eigenschaften: Eigenschaft = Prädikat P

zu zeigen: für alle $x \in \mathbb{N}$ gilt $P(x)$
--

vollständige Induktion:

- ▶ **Induktionsanfang:**

zeige $P(x)$ für $x = 0$

- ▶ **Induktionsvoraussetzung:**

Sei $x \in \mathbb{N}$, sodass $P(x)$ gilt.

$P(x)$ gilt noch nicht für *alle* $x \in \mathbb{N}$

- ▶ **Induktionsschritt:**

zeige $P(x + 1)$ unter Nutzung der Induktionsvoraussetzung

INDUKTION AUF LISTEN

Erinnerung: Rekursion über Listen xs

Basisfall: $xs = []$

Rekursionsfall: $xs = (y:ys)$ für $ys :: [a]$

Beweis von Programmeigenschaften: Eigenschaft = Prädikat P

zu zeigen: für alle $xs :: [a]$ gilt $P(xs)$
--

Induktion auf Listen:

- ▶ **Induktionsanfang:**
zeige $P(xs)$ für $xs == []$
- ▶ **Induktionsvoraussetzung:**
Sei $xs :: [a]$ eine Liste für die $P(xs)$ gilt.
- ▶ **Induktionsschritt:**
zeige $P(x:xs)$ für alle $x :: a$ unter Nutzung der Induktionsvoraussetzung

<i>Allgemeiner Hinweis:</i> Es müssen immer alle Variablen quantifiziert werden!

STRUKTURELLE INDUKTION

Erinnerung: Rekursion über Bäume

Basisfall: Nil oder Leaf x für $x :: a$

Rekursionsfall: Branch x l r für $x :: a$ und $l, r :: \text{BinTree } a$

zu zeigen: für alle $t :: \text{BinTree } a$ gilt $P(t)$
--

strukturelle Induktion:

► **Induktionsanfang:**

zeige $P(t)$ für $t == \text{Nil}$ oder $t == \text{Leaf } x$ für alle $x :: a$

► **Induktionsvoraussetzung:**

Seien $l, r :: \text{BinTree } a$ zwei Bäume, sodass $P(l)$ und $P(r)$ gilt.

► **Induktionsschritt:**

zeige $P(\text{Branch } x \ l \ r)$ für alle $x :: a$ unter Nutzung der Induktionsvoraussetzung

<i>Allgemeiner Hinweis:</i> Es müssen immer alle Variablen quantifiziert werden!

FEHLERQUELLEN

- ▶ kein Induktionsprinzip
- ▶ IV wird im Induktionsschritt nicht verwendet
- ▶ fehlende Quantifizierung (nur Gleichungen bringen kaum Punkte)
- ▶ *Missachtung freier Variablen*
- ▶ zu beweisende Eigenschaft P wird für xs angenommen, um sie dann im Induktionsschritt nochmal für xs zu beweisen — eine Tautologie
- ▶ Annahme, dass P bereits für alle Listen gilt, um es dann für $x:xs$ nochmal zu zeigen

AUFGABE 2

Zu zeigen ist die Gleichung

$$\text{sum (foo xs)} = 2 * \text{sum xs} - \text{length xs} \quad \text{für alle } xs :: \text{Int}$$

mittels Induktion über Listen.

Induktionsanfang: Sei $xs == []$.

linke Seite:

$$\text{sum (foo [])} \stackrel{(2)}{=} \text{sum []} \stackrel{(6)}{=} 0$$

rechte Seite:

$$2 * \text{sum []} - \text{length []} \stackrel{(10)}{=} 2 * \text{sum []} - 0 \stackrel{(6)}{=} 2 * 0 - 0 = 0$$

Induktionsvoraussetzung: Sei $xs :: [\text{Int}]$, sodass

$$\text{sum (foo xs)} = 2 * \text{sum xs} - \text{length xs}$$

gilt.

AUFGABE 2 (FORTSETZUNG)

Induktionsschritt: Sei $x :: \text{Int}$. Es gilt

$$\begin{aligned} \text{sum (foo (x:xs))} &\stackrel{(3)}{=} \text{sum (x : x : (-1) : foo xs)} \\ &\stackrel{3.(7)}{=} x + x + (-1) + \text{sum (foo xs)} \\ &\stackrel{(IV)}{=} x + x + (-1) + 2 * \text{sum xs} - \text{length xs} \\ &\stackrel{(\text{Komm.})}{=} 2 * x + 2 * \text{sum xs} - 1 - \text{length xs} \\ &\stackrel{(\text{Dist.})}{=} 2 * (x + \text{sum xs}) - (1 + \text{length xs}) \\ &\stackrel{(7)}{=} 2 * \text{sum (x:xs)} - (1 + \text{length xs}) \\ &\stackrel{(11)}{=} 2 * \text{sum (x:xs)} - \text{length (x:xs)} \end{aligned}$$

ENDE

Fragen?