

Tutorium 07: Unifikation

David Kaufmann

14. Dezember 2022

Tutorium Programmierparadigmen am KIT

Unifikation

Welche Probleme löst die Unifikation?

$$\begin{aligned}\text{state}(1, 1, 1, 1) &\stackrel{!}{=} \text{state}(M, W, Z, K) \\ \text{opposite}(M, M_2) &\stackrel{!}{=} \text{opposite}(1, r) \\ Z &\stackrel{!}{=} K\end{aligned}$$

- Unifikation löst Gleichungssysteme von baumförmigen Termen (hier: Prolog-Terme).
- Eingabe: Menge $C = \{\theta_l^1 \stackrel{!}{=} \theta_r^1, \dots, \theta_l^n \stackrel{!}{=} \theta_r^n\}$
 - Alle $\theta_{\{l,r\}}^i$ sind Bäume und können Variablen enthalten.
- Ausgabe: *Unifikator* σ , sodass $\sigma(\theta_l^i) = \sigma(\theta_r^i)$.
 - Wenn C nicht unifizierbar (bspw. $X = f(X)$): fail

Welche Probleme löst die Unifikation?

$$\begin{aligned}\text{state}(1, 1, 1, 1) &\stackrel{!}{=} \text{state}(M, W, Z, K) \\ \text{opposite}(M, M_2) &\stackrel{!}{=} \text{opposite}(1, r) \\ Z &\stackrel{!}{=} K\end{aligned}$$

Mehrere mögliche Lösungen:

$$\sigma_1 = [M_2 \mapsto r] \circ [K \mapsto 1] \circ [Z \mapsto 1] \circ [W \mapsto 1] \circ [M \mapsto 1]$$

$$\sigma_2 = [M_2 \mapsto r] \circ [K \mapsto 1] \circ [Z \mapsto K] \circ [W \mapsto 1] \circ [M \mapsto 1]$$

- I.d.R. suchen wir nach *einem* allgemeinsten Unifikator (mgu).
- $\text{mgu} \approx$ minimaler Unifikator, der C löst.

Unifiziert:

$$A \stackrel{!}{=} x$$

$$B \stackrel{!}{=} f(X)$$

$$C \stackrel{!}{=} g(C)$$

$$f(x, D, z) \stackrel{!}{=} f(x, y, E)$$

$$\text{func}(F, \text{func}(G, z)) \stackrel{!}{=} \text{func}(x, \text{func}(y, F))$$

$$g(x, H, z) \stackrel{!}{=} f(x, H, H)$$

$$f(g(z)) \stackrel{!}{=} f(J)$$

Ergebnis: Entweder **fail** oder ein Unifikator.

Unifikationsalgorithmus: $\text{unify}(C) =$

```
if  $C == \emptyset$  then []  
else let  $\{\theta_l = \theta_r\} \cup C' = C$  in  
  if  $\theta_l == \theta_r$  then  $\text{unify}(C')$   
  else if  $\theta_l == Y$  and  $Y \notin FV(\theta_r)$  then  $\text{unify}([Y \dot{=} \theta_r] C') \circ [Y \dot{=} \theta_r]$   
  else if  $\theta_r == Y$  and  $Y \notin FV(\theta_l)$  then  $\text{unify}([Y \dot{=} \theta_l] C') \circ [Y \dot{=} \theta_l]$   
  else if  $\theta_l == f(\theta_l^1, \dots, \theta_l^n)$  and  $\theta_r == f(\theta_r^1, \dots, \theta_r^n)$   
    then  $\text{unify}(C' \cup \{\theta_l^1 = \theta_r^1, \dots, \theta_l^n = \theta_r^n\})$   
  else fail
```

$Y \in FV(\theta)$ **occur check**, verhindert zyklische Substitutionen

Korrektheitstheorem

$\text{unify}(C)$ terminiert und gibt *mgu* für C zurück, falls C unifizierbar, ansonsten **fail**.

Beweis: Siehe [Pie02]

Robinson-Algorithmus: Zeile für Zeile

```
if  $C == \emptyset$  then []  
else let  $\{\theta_l \stackrel{!}{=} \theta_r\} \cup C' = C$  in
```

- Ist das Gleichungssystem C leer, ist es schon gelöst
 \leadsto wir brauchen nichts zu ersetzen.
- Andernfalls betrachten wir *eine* der Gleichungen: $\theta_l \stackrel{!}{=} \theta_r$.
 - *Beliebige Auswahl möglich.*
 - Die restlichen Gleichungen merken wir uns als C' .
- Beispiel:
 $C = \{X \stackrel{!}{=} a, Y \stackrel{!}{=} f(X), f(Z) \stackrel{!}{=} Y\}$
 $\theta_l = X, \theta_r = a, C' = \{Y \stackrel{!}{=} f(X), f(Z) \stackrel{!}{=} Y\}$

if $\theta_l == \theta_r$ then unify(C')

- Wenn die Gleichung trivial ist (auf beiden Seiten steht schon das gleiche), brauchen wir auch nichts zu ersetzen.
- Wir müssen also nur C' unifizieren.
- Verschiedene Gleichheitsrelationen:
 - $A \stackrel{!}{=} B$: Element von C , behandeln wir wie eine Datenstruktur.
 - $A == B$: Vergleichsoperator
 - $A = B$: Meta-Gleichheitsoperator, Notation für Pattern-Matching

Robinson-Algorithmus: Zeile für Zeile

else if $\theta_l == Y$ and $Y \notin FV(\theta_r)$
then $\text{unify}([Y \dot{\mapsto} \theta_r] C') \circ [Y \dot{\mapsto} \theta_r]$

- Steht auf der linken Seite eine Variable, so wird diese ersetzt.
 - $\theta_l == Y$: „Ist der Term θ_l eine Variable Y ?“
 - $Y \notin FV(\theta_r)$: *occurs check*, Y darf sich nicht selbst einsetzen.
- Wir ersetzen in C' dann Y durch θ_r .
- Substitution $[Y \dot{\mapsto} \theta_r]$ wird als Ergebnis vorgemerkt.
- Beispiel: $\theta_l = X$ ✓, $X \notin FV(\theta_r) = FV(a) = \emptyset$ ✓, d.h.
Ergebnis: $\text{unify}(\{Y \stackrel{!}{=} f(a), f(Z) \stackrel{!}{=} Y\}) \circ [A \dot{\mapsto} a]$

Robinson-Algorithmus: Zeile für Zeile

else if $\theta_r == Y$ and $Y \notin FV(\theta_l)$
then $\text{unify}([Y \dot{=} \theta_l] C') \circ [Y \dot{=} \theta_l]$

- Auch wenn rechts eine Variable steht muss sie ersetzt werden.
- Beispiel: $\theta_r = Y$ ✓, $Y \notin FV(\theta_l) = FV(f(Z)) = \{Z\}$ ✓, d.h.
Ergebnis: $\text{unify}(\{f(Z) \stackrel{!}{=} f(a)\}) \circ [Y \dot{=} f(Z)]$

Robinson-Algorithmus: Zeile für Zeile

else if $\theta_l == f(\theta_l^1, \dots, \theta_l^n)$ and $\theta_r == f(\theta_r^1, \dots, \theta_r^n)$
then unify($C' \cup \{\theta_l^1 \stackrel{!}{=} \theta_r^1, \dots, \theta_l^n \stackrel{!}{=} \theta_r^n\}$)

- Steht auf beiden Seiten ein Funktor, extrahieren wir paarweise neue Gleichungen und unifizieren diese mitsamt C' .
 - Namen der Funktoren *müssen identisch sein!* (hier: f)
 - Parameterzahlen der Funktoren *müssen identisch sein!*
 - Für Atome: $n = 0$, aber schon abgedeckt durch den ersten Fall.
- Beispiel: $\theta_l = f(Z), \theta_r = f(a) \checkmark, C' = \emptyset$
Ergebnis: unify($\emptyset \cup \{Z \stackrel{!}{=} a\}$) = $[Z \mapsto a]$ (links Variable)

Gegeben sei die einelementige Menge von Gleichungen

$$C = \{f(X_1, X_1) = f(f(X_2, X_3), f(X_4, g(X_4)))\}$$

Führen Sie den Unifikationsalgorithmus nach Robinson durch.
Geben Sie bei jedem rekursiven Aufruf die erzeugte Substitution
sowie die noch zu unifizierende Menge an.

Geben Sie das Endergebnis in Listenform an.

Typinferenz

Lambda-Terme bestehen aus

- Lambdas $\lambda p. b$
- Funktionsanwendungen $x y$
- Variablen x , Konstanten `true`, `17`

$\lambda a. \lambda f. f (a \text{ true})$

Struktur von Lambda-Termen

Lambda-Terme bestehen aus

- **Lambdas** $\lambda p. b$
- Funktionsanwendungen $x y$
- Variablen x , Konstanten `true`, `17`

$$\underbrace{\lambda a. \underbrace{\lambda f. f (a \text{ true})}_{\text{Lambda } p=f, b=f (a \text{ true})}}_{\text{Lambda } p=a, b=\lambda f. f (a \text{ true})}$$

Struktur von Lambda-Termen

Lambda-Terme bestehen aus

- Lambdas $\lambda p. b$
- **Funktionsanwendungen** $x y$
- Variablen x , Konstanten `true`, `17`

$$\lambda a. \lambda f. f \underbrace{(a \text{ true})}_{\substack{\text{Anwendung} \\ x=a, y=\text{true}}} \underbrace{\phantom{(a \text{ true})}}_{\substack{\text{Anwendung} \\ x=f, y=a \text{ true}}}$$

Struktur von Lambda-Termen

Lambda-Terme bestehen aus

- Lambdas $\lambda p. b$
- Funktionsanwendungen $x y$
- **Variablen** x , Konstanten `true`, `17`

$$\lambda a. \lambda f. \underbrace{f}_{\text{Variable}} \left(\underbrace{a}_{\text{Variable}} \text{ true} \right)$$

Struktur von Lambda-Termen

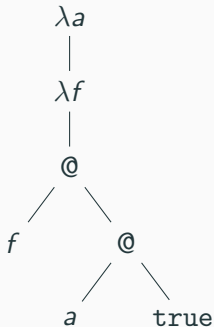
Lambda-Terme bestehen aus

- Lambdas $\lambda p. b$
- Funktionsanwendungen $x y$
- Variablen x , **Konstanten** `true`, `17`

$\lambda a. \lambda f. f (a \underbrace{\text{true}}_{\text{Konstante}})$

Lambda-Terme als Bäume

Wir können Lambda-Terme also als Bäume mit Lambda- und Anwendungsknoten und Variablen- und Konstantenblättern betrachten, um ihre Struktur zu untersuchen:



$\lambda a. \lambda f. f (a \text{ true})$

Cheatsheet: Typisierter Lambda-Kalkül

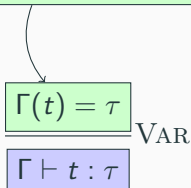
$$\frac{\Gamma(t) = \tau}{\Gamma \vdash t : \tau} \text{VAR} \qquad \frac{\Gamma \vdash f : \phi \rightarrow \alpha \quad \Gamma \vdash x : \phi}{\Gamma \vdash f x : \alpha} \text{APP}$$

$$\frac{\Gamma, p : \pi \vdash b : \rho}{\Gamma \vdash \lambda p. b : \pi \rightarrow \rho} \text{ABS}$$

- Typvariablen: τ, α, π, ρ
- Funktionstypen: $\tau_1 \rightarrow \tau_2$, rechtsassoziativ
- (Weitere Typen: Listen, Tupel, etc.)
- *Typisierungsregeln sind eindeutig*: Eine Regel pro Termform

(Allgemeine) Typisierungsregel für Variablen

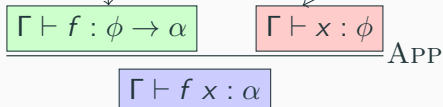
- „Der Typkontext Γ enthält einen Typ τ für t .“



- Daraus folgt:
- „Variable t hat im Kontext Γ den Typ τ .“

Typisierungsregel für Funktionsanwendungen

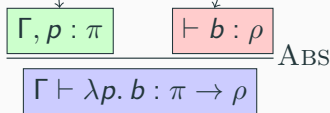
- „ f ist im Kontext Γ eine Funktion, die ϕ s auf α s abbildet.“
- „ x ist im Kontext Γ ein Term des Typs ϕ .“



- Daraus folgt:
- „ x eingesetzt in f ergibt einen Term des Typs α .“

Typisierungsregel für Lambdas

- „Unter Einfügung des Typs π von p in den Kontext...“
- „... ist b als Funktion von p typisierbar.“



- Daraus folgt:
- „ $\lambda p. b$ ist eine Funktion, die π s auf ρ s abbildet“

Vorgehensweise zur Typinferenz:

- Stelle Typherleitungsbaum auf
 - In jedem Schritt werden neue Typvariablen α_i angelegt
 - Statt die Typen direkt im Baum einzutragen, werden Gleichungen in einem Constraint-System eingetragen
- Unifiziere Constraint-System zu einem Unifikator
 - Robinson-Algorithmus, im Grunde wie bei Prolog
 - I.d.R.: Allgemeinster Unifikator (findet man per Robinson)

Unifikationsalgorithmus: $\text{unify}(C) =$

```
if  $C == \emptyset$  then []  
else let  $\{\theta_l = \theta_r\} \cup C' = C$  in  
  if  $\theta_l == \theta_r$  then  $\text{unify}(C')$   
  else if  $\theta_l == Y$  and  $Y \notin FV(\theta_r)$  then  $\text{unify}([Y \dot{=} \theta_r] C') \circ [Y \dot{=} \theta_r]$   
  else if  $\theta_r == Y$  and  $Y \notin FV(\theta_l)$  then  $\text{unify}([Y \dot{=} \theta_l] C') \circ [Y \dot{=} \theta_l]$   
  else if  $\theta_l == f(\theta_l^1, \dots, \theta_l^n)$  and  $\theta_r == f(\theta_r^1, \dots, \theta_r^n)$   
    then  $\text{unify}(C' \cup \{\theta_l^1 = \theta_r^1, \dots, \theta_l^n = \theta_r^n\})$   
  else fail
```

$Y \in FV(\theta)$ **occur check**, verhindert zyklische Substitutionen

Korrektheitstheorem

$\text{unify}(C)$ terminiert und gibt *mgu* für C zurück, falls C unifizierbar, ansonsten **fail**.

Beweis: Siehe [Pie02]

Unifikationsalgorithmus: $\text{unify}(C) =$

```
if  $C == \emptyset$  then []  
else let  $\{\tau_1 = \tau_2\} \cup C' = C$  in  
  if  $\tau_1 == \tau_2$  then  $\text{unify}(C')$   
  else if  $\tau_1 == \alpha$  and  $\alpha \notin FV(\tau_2)$  then  $\text{unify}([\alpha \dot{\vdash} \tau_2] C') \circ [\alpha \dot{\vdash} \tau_2]$   
  else if  $\tau_2 == \alpha$  and  $\alpha \notin FV(\tau_1)$  then  $\text{unify}([\alpha \dot{\vdash} \tau_1] C') \circ [\alpha \dot{\vdash} \tau_1]$   
  else if  $\tau_1 == (\tau'_1 \rightarrow \tau''_1)$  and  $\tau_2 == (\tau'_2 \rightarrow \tau''_2)$   
    then  $\text{unify}(C' \cup \{\tau'_1 = \tau'_2, \tau''_1 = \tau''_2\})$   
  else fail
```

$\alpha \in FV(\tau)$ **occur check**, verhindert zyklische Substitutionen

Korrektheitstheorem

$\text{unify}(C)$ terminiert und gibt *mgu* für C zurück, falls C unifizierbar, ansonsten **fail**.

Beweis: Siehe Literatur

Typen kann man auch als Funktoren darstellen:

$$\begin{array}{lll} \tau_1 \rightarrow \tau_2 & \equiv & \text{func}(\tau_1, \tau_2) \\ [\tau] & \equiv & \text{list}(\tau) \\ & \text{etc.} & \end{array}$$

$$\frac{\dots}{f : \text{int} \rightarrow \beta \vdash \lambda x. f \ x : \alpha_1} \text{ABS}$$

- „Finde den allgemeinsten Typen α_1 von $\lambda x. f \ x$ “

Erinnerung:

- Baum mit durchnummerierten α_i aufstellen
- Constraints sammeln:

$$\frac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_j} \text{VAR}$$

Constraint:

$$\{\alpha_i = \alpha_j\}$$

$$\frac{\Gamma \vdash f : \alpha_j \quad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f \ x : \alpha_i} \text{APP}$$

Constraint:

$$\{\alpha_j = \alpha_k \rightarrow \alpha_i\}$$

$$\frac{\Gamma, p : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda p. b : \alpha_i} \text{ABS}$$

Constraint:

$$\{\alpha_i = \alpha_j \rightarrow \alpha_k\}$$

- Constraint-System auflösen

$$\frac{\dots}{\vdash \lambda f. \lambda x. (f \ x) \ x : \alpha_1} \text{ABS}$$

- „Finde den allgemeinsten Typen α_1 von $\lambda f. \lambda x. (f \ x) \ x$ “

Erinnerung:

- Baum mit durchnummerierten α_i aufstellen
- Constraints sammeln:

$$\frac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_j} \text{VAR}$$

$$\frac{\Gamma \vdash f : \alpha_j \quad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f \ x : \alpha_i} \text{APP}$$

$$\frac{\Gamma, p : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda p. b : \alpha_i} \text{ABS}$$

Constraint:

$$\{\alpha_i = \alpha_j\}$$

Constraint:

$$\{\alpha_j = \alpha_k \rightarrow \alpha_i\}$$

Constraint:

$$\{\alpha_i = \alpha_j \rightarrow \alpha_k\}$$

- Constraint-System auflösen

- Mehr...
 - ...Prolog?
 - ...Unifikation?
 - ...Typherleitung?
 - ...Haskell?