Tutorium 11: Typinferenz

Paul Brinkmeier

02. Februar 2021

Tutorium Programmierparadigmen am KIT

Heutiges Programm

Programm

- Unifikation
  - Robinson-Algorithmus
  - Allgemeinster Unifikator (mgu)
- Typinferenz
  - Regelsystem mit Ableitungsbäumen
  - Constraint-System
  - Unifikation zu einem mgu

Unifikation

Welche Probleme löst die Unifikation?

$$\texttt{state}(\texttt{l},\texttt{l},\texttt{l},\texttt{l}) \overset{!}{=} \texttt{state}(M, W, Z, K)$$
$$\texttt{opposite}(M, M_2) \overset{!}{=} \texttt{opposite}(\texttt{l},\texttt{r})$$
$$Z \overset{!}{=} K$$

- Unifikation löst Gleichungssysteme von baumförmigen Termen (hier: Prolog-Terme).
- Eingabe: Menge $C = \{\theta_l^1 \overset{!}{=} \theta_r^1, ..., \theta_l^n \overset{!}{=} \theta_r^n\}$
  - Alle $\theta^i_{\{l,r\}}$ sind Bäume und können Variablen enthalten.
- Ausgabe: <u>Unifikator</u> σ, sodass $\sigma(\theta_l^i) = \sigma(\theta_r^i)$.
  - Wenn C nicht unifizierbar (bspw. $X = \texttt{f}(X)$): `fail`

Welche Probleme löst die Unifikation?

$$\texttt{state}(\texttt{l},\texttt{l},\texttt{l},\texttt{l}) \stackrel{!}{=} \texttt{state}(M, W, Z, K)$$
$$\texttt{opposite}(M, M_2) \stackrel{!}{=} \texttt{opposite}(\texttt{l},\texttt{r})$$
$$Z \stackrel{!}{=} K$$

Mehrere mögliche Lösungen:

$$\sigma_1 = [M_2 \Rightarrow \texttt{r}] \circ [K \Rightarrow \texttt{l}] \circ [Z \Rightarrow \texttt{l}] \circ [W \Rightarrow \texttt{l}] \circ [M \Rightarrow \texttt{l}]$$
$$\sigma_2 = [M_2 \Rightarrow \texttt{r}] \circ [K \Rightarrow \texttt{l}] \circ [Z \Rightarrow \texttt{K}] \circ [W \Rightarrow \texttt{l}] \circ [M \Rightarrow \texttt{l}]$$

- I.d.R. suchen wir nach <u>einem</u> allgemeinsten Unifikator (mgu).
- mgu $\approx$ minimaler Unifikator, der C löst.

Unifikation (Robinson, [Rob65])

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

$$
\begin{array}{l}
\texttt{if}\ C\ \texttt{==}\ \emptyset\ \texttt{then}\ [] \\
\texttt{else let}\ \{\theta_l = \theta_r\} \cup C' = C\ \texttt{in} \\
\quad \texttt{if}\ \theta_l\ \texttt{==}\ \theta_r\ \texttt{then unify}(C') \\
\quad \texttt{else if}\ \theta_l\ \texttt{==}\ \texttt{Y and Y} \notin FV(\theta_r)\ \texttt{then unify}([\texttt{Y} \Rightarrow \theta_r]\, C') \circ [\texttt{Y} \Rightarrow \theta_r] \\
\quad \texttt{else if}\ \theta_r\ \texttt{==}\ \texttt{Y and Y} \notin FV(\theta_l)\ \texttt{then unify}([\texttt{Y} \Rightarrow \theta_l]\, C') \circ [\texttt{Y} \Rightarrow \theta_l] \\
\quad \texttt{else if}\ \theta_l\ \texttt{==}\ \texttt{f}(\theta_l^1, \ldots, \theta_l^n)\ \texttt{and}\ \theta_r\ \texttt{==}\ \texttt{f}(\theta_r^1, \ldots, \theta_r^n) \\
\qquad\quad \texttt{then unify}(C' \cup \{\theta_l^1 = \theta_r^1, \ldots, \theta_l^n = \theta_r^n\}) \\
\quad \texttt{else}\ \textbf{fail}
\end{array}
$$

$\texttt{Y} \in FV(\theta)$ occur check, verhindert zyklische Substitutionen

Korrektheitstheorem

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

Beweis: Siehe [Pie02]

Robinson-Algorithmus: Zeile für Zeile

$$\texttt{if } C == \emptyset \texttt{ then } []$$

$$\texttt{else let } \{\theta_l \stackrel{!}{=} \theta_r\} \cup C' = C \texttt{ in}$$

- Ist das Gleichungssystem C leer, ist es schon gelöst
  $\rightsquigarrow$ 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:
  $C = \{X \stackrel{!}{=} \mathtt{a}, Y \stackrel{!}{=} \mathtt{f}(X), \mathtt{f}(Z) \stackrel{!}{=} Y\}$
  $\theta_l = X, \theta_r = \mathtt{a}, C' = \{Y \stackrel{!}{=} \mathtt{f}(X), \mathtt{f}(Z) \stackrel{!}{=} Y\}$

Robinson-Algorithmus: Zeile für Zeile

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

- 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:
  - $A \overset{!}{=} B$: Element von C, behandeln wir wie eine Datenstruktur.
  - $A == B$: Vergleichsoperator
  - $A = B$: Meta-Gleichheitsoperator, Notation für Pattern-Matching

Robinson-Algorithmus: Zeile für Zeile

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

- 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 $[Y \Rightarrow \theta_r]$ wird als Ergebnis vorgemerkt.
- Beispiel: $\theta_l = X$ ✓, $X \notin FV(\theta_r) = FV(\mathtt{a}) = \emptyset$ ✓, d.h.
  Ergebnis: $\texttt{unify}(\{Y \overset{!}{=} \mathtt{f}(\mathtt{a}), \mathtt{f}(Z) \overset{!}{=} Y\}) \circ [A \Rightarrow \mathtt{a}]$

Robinson-Algorithmus: Zeile für Zeile

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

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

Robinson-Algorithmus: Zeile für Zeile

$$\texttt{else if } \theta_l == \texttt{f}(\theta_l^1, ..., \theta_r^n) \texttt{ and } \theta_r == \texttt{f}(\theta_r^1, ..., \theta_r^n)$$

$$\texttt{then unify}(C' \cup \{\theta_l^1 \stackrel{!}{=} \theta_r^1, ..., \theta_l^n \stackrel{!}{=} \theta_r^n\})$$

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

Eine vollständige Ausführung

$$C = \{X \stackrel{!}{=} \mathtt{a}, Y \stackrel{!}{=} \mathtt{f}(X), \mathtt{f}(Z) \stackrel{!}{=} Y\}$$

$$\begin{aligned}
\mathtt{unify}(C) &= \mathtt{unify}(\{\underline{X \stackrel{!}{=} \mathtt{a}}, Y \stackrel{!}{=} \mathtt{f}(X), \mathtt{f}(Z) \stackrel{!}{=} Y\}) \\
&= \mathtt{unify}(\{Y \stackrel{!}{=} \mathtt{f}(\mathtt{a}), \underline{\mathtt{f}(Z) \stackrel{!}{=} Y}\}) \circ [X \mapsto \mathtt{a}] \\
&= \mathtt{unify}(\{\underline{\mathtt{f}(Z) \stackrel{!}{=} \mathtt{f}(\mathtt{a})}\}) \circ [Y \mapsto \mathtt{f}(Z)] \circ [X \mapsto \mathtt{a}] \\
&= \mathtt{unify}(\{\underline{Z \stackrel{!}{=} \mathtt{a}}\}) \circ [Y \mapsto \mathtt{f}(Z)] \circ [X \mapsto \mathtt{a}] \\
&= [Z \mapsto \mathtt{a}] \circ [Y \mapsto \mathtt{f}(Z)] \circ [X \mapsto \mathtt{a}] \\
&= [Z \mapsto \mathtt{a}, Y \mapsto \mathtt{f}(\mathtt{a}), X \mapsto \mathtt{a}]
\end{aligned}$$

- Der Robinson-Algorithmus liefert einen möglichen mgu.
- Letzte Zeile: Unifikator angewandt auf jede Variable, also keine „Ketten“-Ersetzungen mehr nötig.

Übungsaufgabe: Unifikation

$$C = \{A = \mathtt{f}(B, C), C = \mathtt{f}(D, E), B = E\}$$

Wendet den Robinson-Algorithmus an, um einen mgu für C zu finden!

Übungsaufgabe: Unifikation

$$C = \{A = \mathtt{f}(B, C), C = \mathtt{f}(D, E), B = E\}$$

$$\begin{aligned}
\mathtt{unify}(C) &= \mathtt{unify}(\{\underline{A = \mathtt{f}(B, C)}, C = \mathtt{f}(D, E), B = E\}) \\
&= \mathtt{unify}(\{\underline{C = \mathtt{f}(D, E)}, B = E\}) \circ [A \Rightarrow \mathtt{f}(B, C)] \\
&= \mathtt{unify}(\{\underline{B = E}\}) \circ [C \Rightarrow \mathtt{f}(D, E)] \circ [A \Rightarrow \mathtt{f}(B, C)] \\
&= [B \Rightarrow E] \circ [C \Rightarrow \mathtt{f}(D, E)] \circ [A \Rightarrow \mathtt{f}(B, C)] \\
&= [A \Rightarrow \mathtt{f}(E, \mathtt{f}(D, E)), C \Rightarrow \mathtt{f}(D, E), B \Rightarrow E]
\end{aligned}$$

Eure Lösung sollte so oder so ähnlich aussehen, da ihr die Reihenfolge der Gleichungen beliebig wählen dürft.

Typinferenz

Cheatsheet: Lambda-Kalkül/Basics

- Terme t: Variable (x), Funktion ($\lambda x.t$), Anwendung ($t\ t$)
- α-Äquivalenz: Gleiche Struktur
- η-Äquivalenz: Unterversorgung
- Freie Variablen, Substitution, RedEx
- β-Reduktion:
  $(\lambda p.b)\ t \Rightarrow b\,[p \to t]$

Cheatsheet: Typisierter Lambda-Kalkül

$$\frac{\Gamma(t) = \tau}{\Gamma \vdash t : \tau}\textsc{Var} \qquad \frac{\Gamma \vdash f : \phi \to \alpha \qquad \Gamma \vdash x : \phi}{\Gamma \vdash f\ x : \alpha}\textsc{App}$$

$$\frac{\Gamma, p : \pi \vdash b : \rho}{\Gamma \vdash \lambda p.\, b : \pi \to \rho}\textsc{Abs}$$

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

Einfache Typisierungsregel für Variablen

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

$$\frac{\boxed{\Gamma(t) = \tau}}{\boxed{\Gamma \vdash t : \tau}}\ \textsc{Var}$$

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

Typisierungsregel für Funktionsanwendungen

- „f ist im Kontext Γ eine Funktion, die ϕs auf αs abbildet."
- „x ist im Kontext Γ ein Term des Typs ϕ."

$$\frac{\boxed{\Gamma \vdash f : \phi \to \alpha} \qquad \boxed{\Gamma \vdash x : \phi}}{\boxed{\Gamma \vdash f\ x : \alpha}}\ \textsc{App}$$

- 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 typisiertbar.“

$$\dfrac{\boxed{\Gamma, p : \pi} \qquad \boxed{\vdash b : \rho}}{\boxed{\Gamma \vdash \lambda p.\, b : \pi \to \rho}}\ \textsc{Abs}$$

- Daraus folgt:
- „$\lambda p.\, b$ ist eine Funktion, die πs auf ρs abbildet“

Algorithmus 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 (mgu)

$$\frac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_i}\textsc{Var} \qquad \frac{\Gamma \vdash f : \alpha_j \qquad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f\ x : \alpha_i}\textsc{App} \qquad \frac{\Gamma, p : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda p.\, b : \alpha_i}\textsc{Abs}$$

Constraint:
$\{\alpha_i = \alpha_j\}$

Constraint:
$\{\alpha_j = \alpha_k \to \alpha_i\}$

Constraint:
$\{\alpha_i = \alpha_j \to \alpha_k\}$

Herleitungsbaum: Beispiel

Gegebener Term: $\lambda f.\, \lambda x.\, f\ x\ x$

$$\dfrac{\dfrac{\dfrac{\Gamma'(f) = \alpha_2}{\Gamma' \vdash f : \alpha_8}\textsc{Var} \qquad \dfrac{\Gamma'(x) = \alpha_4}{\Gamma' \vdash x : \alpha_9}\textsc{Var}}{\Gamma' \vdash f\ x : \alpha_6}\textsc{App} \qquad \dfrac{\Gamma'(x) = \alpha_4}{\Gamma' \vdash x : \alpha_7}\textsc{Var}}{\dfrac{f : \alpha_2, x : \alpha_4 \vdash f\ x\ x : \alpha_5}{\dfrac{f : \alpha_2 \vdash \lambda x.\, f\ x\ x : \alpha_3}{\vdash \lambda f.\, \lambda x.\, f\ x\ x : \alpha_1}\textsc{Abs}}\textsc{Abs}}\textsc{App}$$

$$\Gamma' = f : \alpha_2, x : \alpha_4$$

Herleitungsbaum: Beispiel

Gegebener Term: $\lambda f.\, \lambda x.\, f\ x\ x$

$$
\dfrac{
  \dfrac{
    \dfrac{\Gamma'(f) = \alpha_2}{\Gamma' \vdash f : \alpha_8}\textsc{Var}
    \qquad
    \dfrac{\Gamma'(x) = \alpha_4}{\Gamma' \vdash x : \alpha_9}\textsc{Var}
  }{
    \dfrac{
      \dfrac{}{\Gamma' \vdash f\ x : \alpha_6}\textsc{App}
      \qquad
      \dfrac{\Gamma'(x) = \alpha_4}{\Gamma' \vdash x : \alpha_7}\textsc{Var}
    }{
      f : \alpha_2, x : \alpha_4 \vdash f\ x\ x : \alpha_5
    }\textsc{App}
  }
}{}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma'(f) = \alpha_2}{\Gamma' \vdash f : \alpha_8}\textsc{Var} \qquad \dfrac{\Gamma'(x) = \alpha_4}{\Gamma' \vdash x : \alpha_9}\textsc{Var}}{\Gamma' \vdash f\ x : \alpha_6}\textsc{App} \qquad \dfrac{\Gamma'(x) = \alpha_4}{\Gamma' \vdash x : \alpha_7}\textsc{Var}}{f : \alpha_2, x : \alpha_4 \vdash f\ x\ x : \alpha_5}\textsc{App}}{f : \alpha_2 \vdash \lambda x.\, f\ x\ x : \alpha_3}\textsc{Abs}}{\vdash \lambda f.\, \lambda x.\, f\ x\ x : \alpha_1}\textsc{Abs}
$$

$$\Gamma' = f : \alpha_2, x : \alpha_4$$

Constraintmenge:

$$
\begin{aligned}
C = \{&\alpha_1 = \alpha_2 \to \alpha_3, \alpha_3 = \alpha_4 \to \alpha_5, \\
&\alpha_6 = \alpha_7 \to \alpha_5, \alpha_8 = \alpha_9 \to \alpha_6, \\
&\alpha_2 = \alpha_8, \alpha_4 = \alpha_9, \alpha_4 = \alpha_7 \\
\}&
\end{aligned}
$$

Übungsaufgabe: Herleitungsbäume aufstellen

$$\frac{\dots}{\vdash \lambda x.\, \lambda y.\, x : \alpha_1}\textsc{Abs}$$

Stellt den Herleitungsbaum für $\lambda x.\, \lambda y.\, x$ auf!

$$\frac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_i}\textsc{Var}$$

Constraint:
$\{\alpha_i = \alpha_j\}$

$$\frac{\Gamma \vdash f : \alpha_j \qquad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f\ x : \alpha_i}\textsc{App}$$

Constraint:
$\{\alpha_j = \alpha_k \to \alpha_i\}$

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

Constraint:
$\{\alpha_i = \alpha_j \to \alpha_k\}$

Übungsaufgabe: Herleitungsbäume aufstellen

$$\dfrac{\ldots}{\vdash \lambda f.\, f\ (\lambda x.\, x) : \alpha_1}\textsc{Abs}$$

Stellt den Herleitungsbaum für $\lambda f.\, f\ (\lambda x.\, x)$ auf!

$$\dfrac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_i}\textsc{Var}$$

Constraint:
$\{\alpha_i = \alpha_j\}$

$$\dfrac{\Gamma \vdash f : \alpha_j \qquad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f\ x : \alpha_i}\textsc{App}$$

Constraint:
$\{\alpha_j = \alpha_k \to \alpha_i\}$

$$\dfrac{\Gamma, p : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda p.\, b : \alpha_i}\textsc{Abs}$$

Constraint:
$\{\alpha_i = \alpha_j \to \alpha_k\}$

Übungsaufgabe: Constraint-System aufstellen

Stellt die Constraint-Systeme C_1 und C_2 zu den Herleitungsbäumen auf!

$$\dfrac{\dfrac{\dfrac{(x : \alpha_2, y : \alpha_4)(x) = \alpha_2}{x : \alpha_2, y : \alpha_4 \vdash x : \alpha_5}\textsc{Var}}{x : \alpha_2 \vdash \lambda y.\, x : \alpha_3}\textsc{Abs}}{\vdash \lambda x.\, \lambda y.\, x : \alpha_1}\textsc{Abs}$$

$$\dfrac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_i}\textsc{Var}$$

Constraint:
$\{\alpha_i = \alpha_j\}$

$$\dfrac{\Gamma \vdash f : \alpha_j \qquad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f\ x : \alpha_i}\textsc{App}$$

Constraint:
$\{\alpha_j = \alpha_k \to \alpha_i\}$

$$\dfrac{\Gamma, p : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda p.\, b : \alpha_i}\textsc{Abs}$$

Constraint:
$\{\alpha_i = \alpha_j \to \alpha_k\}$

Übungsaufgabe: Constraint-System aufstellen

Stellt die Constraint-Systeme C_1 und C_2 zu den Herleitungsbäumen auf!

$$\dfrac{\dfrac{(f : \alpha_2)(f) = \alpha_2}{f : \alpha_2 \vdash f : \alpha_4}\textsc{Var} \qquad \dfrac{\dfrac{\Gamma'(x) = \alpha_6}{\Gamma' \vdash x : \alpha_7}\textsc{Var}}{f : \alpha_2 \vdash \lambda x.\, x : \alpha_5}\textsc{Abs}}{\dfrac{f : \alpha_2 \vdash f\ (\lambda x.\, x) : \alpha_3}{\vdash \lambda f.\, f\ (\lambda x.\, x) : \alpha_1}\textsc{Abs}}\textsc{App}$$

$$\Gamma' = f : \alpha_2, x : \alpha_6$$

$$\dfrac{\Gamma(t) = \alpha_j}{\Gamma \vdash t : \alpha_i}\textsc{Var} \qquad \dfrac{\Gamma \vdash f : \alpha_j \qquad \Gamma \vdash x : \alpha_k}{\Gamma \vdash f\ x : \alpha_i}\textsc{App} \qquad \dfrac{\Gamma, p : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda p.\, b : \alpha_i}\textsc{Abs}$$

Constraint: $\{\alpha_i = \alpha_j\}$

Constraint: $\{\alpha_j = \alpha_k \to \alpha_i\}$

Constraint: $\{\alpha_i = \alpha_j \to \alpha_k\}$

Übungsaufgabe: Constraint-System aufstellen

$$
\begin{aligned}
C_1 &= \{\alpha_1 = \alpha_2 \rightarrow \alpha_3, \alpha_3 = \alpha_4 \rightarrow \alpha_5, \alpha_2 = \alpha_5\} \\
C_2 &= \{\alpha_1 = \alpha_2 \rightarrow \alpha_3, \alpha_4 = \alpha_5 \rightarrow \alpha_3, \\
&\quad \alpha_2 = \alpha_4, \\
&\quad \alpha_5 = \alpha_6 \rightarrow \alpha_7, \alpha_6 = \alpha_7 \\
&\quad \}
\end{aligned}
$$

- Lösung hängt von Variablenbenennung ab $\Rightarrow$ manchmal schwierig, mit Musterlösung zu vergleichen.
- Aber: Eure Lösung muss α-äquivalent sein (selbe Struktur).

Unifikation

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

$$
\begin{array}{l}
\texttt{if } C \texttt{ == } \emptyset \texttt{ then } [] \\
\texttt{else let } \{\tau_1 = \tau_2\} \cup C' = C \texttt{ in} \\
\quad \texttt{if } \tau_1 \texttt{ == } \tau_2 \texttt{ then unify}(C') \\
\quad \texttt{else if } \tau_1 \texttt{ == } \alpha \texttt{ and } \alpha \notin FV(\tau_2) \texttt{ then unify}([\alpha \Rightarrow \tau_2]\, C') \circ [\alpha \Rightarrow \tau_2] \\
\quad \texttt{else if } \tau_2 \texttt{ == } \alpha \texttt{ and } \alpha \notin FV(\tau_1) \texttt{ then unify}([\alpha \Rightarrow \tau_1]\, C') \circ [\alpha \Rightarrow \tau_1] \\
\quad \texttt{else if } \tau_1 \texttt{ == } (\tau_1' \to \tau_1'') \texttt{ and } \tau_2 \texttt{ == } (\tau_2' \to \tau_2'') \\
\qquad\quad \texttt{then unify}(C' \cup \{\tau_1' = \tau_2', \tau_1'' = \tau_2''\}) \\
\quad \texttt{else } \textbf{fail}
\end{array}
$$

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

Korrektheitstheorem

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

Beweis: Siehe Literatur

Unifikation für Typinferenz

- Wir können den Robinson-Algorithmus 1:1 für die Typinferenz übernehmen.
- Statt Prolog-Termen verwenden wir Typterme:
    - Konkrete Typen: int, string, char, etc. $\approx$ `int`, `string`, ...
    - Typvariablen: α, $\tau \approx A, X, \ldots$
    - Funktionstypen: $\tau_1 \rightarrow \tau_2 \approx \mathtt{f}(A, B)$
- Wir könnten den Algorithmus noch erweitern um:
    - Tupel, Listen, Dictionaries, etc.
    - Machen wir aber nicht weil auch diese als Funktoren darstellbar sind $\rightsquigarrow$ Robinson für Prolog reicht

Aufgabe: Unifikation für Typinferenz

$$C_1 = \{\alpha_1 = \alpha_2 \to \alpha_3, \alpha_3 = \alpha_4 \to \alpha_5, \alpha_2 = \alpha_5\}$$

$$\begin{aligned} C_2 = \{&\alpha_1 = \alpha_2 \to \alpha_3, \alpha_4 = \alpha_5 \to \alpha_3, \\ &\alpha_2 = \alpha_4, \\ &\alpha_5 = \alpha_6 \to \alpha_7, \alpha_6 = \alpha_7 \\ &\} \end{aligned}$$

$$\sigma_1 = \texttt{unify}(C_1) = [\alpha_1 \Rightarrow \alpha_5 \to \alpha_4 \to \alpha_5, \alpha_2 \Rightarrow \alpha_5, \alpha_3 \Rightarrow \alpha_4 \to \alpha_5]$$

Der Typ von $\lambda x.\, \lambda y.\, x$ ist also $\sigma_1(\alpha_1) = \alpha_5 \to \alpha_4 \to \alpha_5$.

Findet den Typen von $\lambda f.\, f\ (\lambda x.\, x)$!

Aufgabe: Unifikation für Typinferenz

Findet den Typen von $\lambda f.\, f\ (\lambda x.\, x)$!

$$
\begin{aligned}
&\sigma_2 = \texttt{unify}(C_2) \\
&= \texttt{unify}(\{\alpha_1 = \alpha_2 \to \alpha_3, ...\}) \\
&= \texttt{unify}(\{\alpha_4 = \alpha_5 \to \alpha_3, ...\}) \circ [\alpha_1 \Rightarrow \alpha_2 \to \alpha_3] \\
&= \texttt{unify}(\{\alpha_2 = \alpha_5 \to \alpha_3, ...\}) \circ [\alpha_4 \Rightarrow \alpha_5 \to \alpha_3] \circ [\alpha_1 \Rightarrow \alpha_2 \to \alpha_3] \\
&= ... \\
&= \ [\alpha_1 \Rightarrow ((\alpha_7 \to \alpha_7) \to \alpha_3) \to \alpha_3, ...]
\end{aligned}
$$

$\rightsquigarrow$ Der Typ von $\lambda f.\, f\ (\lambda x.\, x)$ ist $((\alpha_7 \to \alpha_7) \to \alpha_3) \to \alpha_3$.

Let-Polymorphismus: Motivation

$$\lambda f.\, f\ f$$

- Diese Funktion verwendet f auf zwei Arten:
  - $\alpha \to \alpha$: Rechte Seite.
  - $(\alpha \to \alpha) \to (\alpha \to \alpha)$: Linke Seite, nimmt f als Argument und gibt es zurück.

Let-Polymorphismus: Motivation

$$\lambda f.\, f\ f$$

- Diese Funktion verwendet f auf zwei Arten:
  - $\alpha \to \alpha$: Rechte Seite.
  - $(\alpha \to \alpha) \to (\alpha \to \alpha)$: Linke Seite, nimmt f als Argument und gibt es zurück.
- Problem: $\alpha \to \alpha$ und $(\alpha \to \alpha) \to (\alpha \to \alpha)$ sind nicht unifizierbar!
  - „occurs check“: α darf sich nicht selbst einsetzen.
- Idee: Bei jeder Verwendung eines polymorphen Typen erzeugen wir <u>neue Typvariablen</u>, um diese Beschränkung zu umgehen.

Typschemata und Instanziierung

- Idee: Bei jeder Verwendung eines polymorphen Typen erzeugen wir neue Typvariablen, um diese Beschränkung zu umgehen.
- Ein Typschema ist ein Typ, in dem manche Typvariablen allquantifiziert sind:

$$\phi = \forall \alpha_1. \dots \forall \alpha_n. \tau$$
$$\alpha_i \in FV(\tau)$$

- Typschemata kommen bei uns immer nur in Kontexten vor!
- Beispiele:
  - $\forall \alpha . \alpha \to \alpha$
  - $\forall \alpha . \alpha \to \beta \to \alpha$

Typschemata und Instanziierung

- Ein Typschema spannt eine Menge von Typen auf, mit denen es instanziiert werden kann:

$$\forall\alpha.\alpha \to \alpha \succeq \mathsf{int} \to \mathsf{int}$$

$$\forall\alpha.\alpha \to \alpha \succeq \tau \to \tau$$

$$\forall\alpha.\alpha \to \alpha \not\succeq \tau \to \sigma$$

$$\forall\alpha.\alpha \to \alpha \not\succeq \tau \to \tau \to \tau$$

$$\forall\alpha.\alpha \to \alpha \succeq (\tau \to \tau) \to (\tau \to \tau)$$

Let-Polymorphismus

Um Typschemata bei der Inferenz zu verwenden, müssen wir zunächst die Regel für Variablen anpassen:

$$\frac{\Gamma(x) = \phi \qquad \phi \succeq_{\text{frische } \alpha_i} \tau}{\Gamma \vdash x : \alpha_j}\textsc{Var}$$

Constraint: $\{\alpha_j = \tau\}$

- $\succeq_{\text{frische } \alpha_i}$ instanziiert ein Typschema mit α_i, die noch nicht im Baum vorkommen.
- Jetzt brauchen wir noch eine Möglichkeit, Typschemata zu erzeugen.

Let-Polymorphismus

Mit einen Let-Term wird ein Typschema eingeführt:

$$\frac{\Gamma \vdash t_1 : \alpha_i \qquad \Gamma' \vdash t_2 : \alpha_j}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : \alpha_k}\textsc{Let}$$

$$\begin{aligned}
\sigma_{let} &= mgu(C_{let}) \\
\Gamma' &= \sigma_{let}(\Gamma), x : ta(\sigma_{let}(\alpha_i), \sigma_{let}(\Gamma)) \\
C'_{let} &= \{\alpha_n = \sigma_{let}(\alpha_n) \mid \sigma_{let}(\alpha_n) \text{ ist definiert}\}
\end{aligned}$$

Constraints: $C'_{let} \cup C_{body} \cup \{a_j = a_k\}$

Beispiel: Let-Polymorphismus

$$
\dfrac{
\dfrac{\dots}{\vdash \lambda x.\, x : \alpha_2}\,\textsc{Abs}
\qquad
\dfrac{
\dfrac{\begin{array}{c}\Gamma'(f) = \forall \alpha_5.\alpha_5 \rightarrow \alpha_5 \\ \succeq \alpha_8 \rightarrow \alpha_8\end{array}}{\Gamma' \vdash f : \alpha_6}\,\textsc{Var}
\qquad
\dfrac{\begin{array}{c}\Gamma'(f) = \forall \alpha_5.\alpha_5 \rightarrow \alpha_5 \\ \succeq \alpha_9 \rightarrow \alpha_9\end{array}}{\Gamma' \vdash f : \alpha_7}\,\textsc{Var}
}{\Gamma' \vdash f\ f : \alpha_3}\,\textsc{App}
}{\vdash \texttt{let}\ f = \lambda x.\, x\ \texttt{in}\ f\ f : \alpha_1}\,\textsc{Let}
$$

$$
\begin{aligned}
C_{let} &= \{\alpha_2 = \alpha_4 \rightarrow \alpha_5, \alpha_4 \rightarrow \alpha_5\} \\
\sigma_{let} &= [\alpha_2 \Rightarrow \alpha_5 \rightarrow \alpha_5, \alpha_4 \Rightarrow \alpha_5] \\
\Gamma' &= x : \forall \alpha_5.\alpha_5 \rightarrow \alpha_5 \\
C'_{let} &= \{\alpha_2 = \alpha_5 \rightarrow \alpha_5, \alpha_4 = \alpha_5\} \\
C_{body} &= \{\alpha_6 = \alpha_7 \rightarrow \alpha_3, \alpha_6 = \alpha_8 \rightarrow \alpha_8, \alpha_7 = \alpha_9 \rightarrow \alpha_9\} \\
C &= C'_{let} \cup C_{body} \cup \{\alpha_3 = \alpha_1\}
\end{aligned}
$$

Ende

Ende

Von heute mitnehmen:

- Robinson-Unifikation für Prolog-Terme und Lambda-Typen
- Typinferenz:
  - Baum aufstellen
  - Constraints aufsammeln
  - mgu per Robinson berechnen

Nächste Tuts:

- Compilerbau-Grundlagen
- Wir schreiben einen Compiler für Brainfuck:
  `++++++[>+++++++<-]>.`