Tutorium 08: Typisierung & Typinferenz

Paul Brinkmeier

20. Dezember 2022

Tutorium Programmierparadigmen am KIT

Typisierung

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

Klausuraufgabe WS16/17 A3 a) (6P.)

$$C_1 = \{\alpha_9 = \alpha_{10} \to \alpha_8, \alpha_9 = \alpha_4, \alpha_{10} = \texttt{bool}\}$$
$$C_2 = \{\alpha_{12} = \alpha_{13} \to \alpha_{11}, \alpha_{12} = \alpha_4, \alpha_{13} = \texttt{int}\}$$

i. Geben Sie allgemeinste Unifikatoren σ_1 für C_1 und σ_2 für C_2 an.

ii. Ist auch $C_1 \cup C_2$ unifizierbar?

iii. Ist der Ausdruck

$$\lambda \texttt{a}.\, \lambda \texttt{f}.\, \texttt{f}\ (\texttt{a}\ \mathit{true})\ (\texttt{a}\ 17)$$

typisierbar? Begründen Sie ihre Antwort *kurz*.

Klausuraufgabe WS16/17 A3 a) (6P.)

$$C_1 = \{\alpha_9 = \alpha_{10} \to \alpha_8, \alpha_9 = \alpha_4, \alpha_{10} = \texttt{bool}\}$$

$$C_2 = \{\alpha_{12} = \alpha_{13} \to \alpha_{11}, \alpha_{12} = \alpha_4, \alpha_{13} = \texttt{int}\}$$

Geben Sie allgemeinste Unifikatoren σ_1 für C_1 und σ_2 für C_2 an.

$$\sigma_1 = \texttt{unify}(\{\alpha_9 = \alpha_{10} \to \alpha_8, \alpha_9 = \alpha_4, \alpha_{10} = \texttt{bool}\})$$

$$= ... = [\alpha_9 \Rightarrow \texttt{bool} \to \alpha_8, \alpha_4 \Rightarrow \texttt{bool} \to \alpha_8, \alpha_{10} \Rightarrow \texttt{bool}]$$

$$\sigma_2 = \texttt{unify}(\{\alpha_{12} = \alpha_{13} \to \alpha_{11}, \alpha_{12} = \alpha_4, \alpha_{13} = \texttt{int}\})$$

$$= ... = [\alpha_{12} \Rightarrow \texttt{int} \to \alpha_{11}, \alpha_4 \Rightarrow \texttt{int} \to \alpha_{11}, \alpha_{13} \Rightarrow \texttt{int}]$$

Klausuraufgabe WS16/17 A3 a) (6P.)

$$C_1 = \{\alpha_9 = \alpha_{10} \to \alpha_8, \alpha_9 = \alpha_4, \alpha_{10} = \texttt{bool}\}$$

$$C_2 = \{\alpha_{12} = \alpha_{13} \to \alpha_{11}, \alpha_{12} = \alpha_4, \alpha_{13} = \texttt{int}\}$$

Ist auch $C_1 \cup C_2$ unifizierbar?

$$\sigma_1 = ... = \left[\alpha_9 \Rightarrow \texttt{bool} \to \alpha_8, \underline{\alpha_4 \Rightarrow \texttt{bool} \to \alpha_8}, \alpha_{10} \Rightarrow \texttt{bool}\right]$$

$$\sigma_2 = ... = \left[\alpha_{12} \Rightarrow \texttt{int} \to \alpha_{11}, \underline{\alpha_4 \Rightarrow \texttt{int} \to \alpha_{11}}, \alpha_{13} \Rightarrow \texttt{int}\right]$$

A: Nein, da die *allgemeinsten Unifikatoren* σ_1 und σ_2 einen Konflikt für α_4 enthalten: $\texttt{unify}(\{\texttt{bool} = \texttt{int}\}) = \texttt{fail}$

Klausuraufgabe WS16/17 A3 a) (6P.)

$$C_1 = \{\alpha_9 = \alpha_{10} \to \alpha_8, \alpha_9 = \alpha_4, \alpha_{10} = \texttt{bool}\}$$
$$C_2 = \{\alpha_{12} = \alpha_{13} \to \alpha_{11}, \alpha_{12} = \alpha_4, \alpha_{13} = \texttt{int}\}$$

Ist der Ausdruck

$$\lambda \texttt{a}.\, \lambda \texttt{f}.\, \texttt{f}\ (\texttt{a}\ \textit{true})\ (\texttt{a}\ 17)$$

typisierbar? Begründen Sie ihre Antwort *kurz*.

A: Nein, da a mit zwei verschiedenen Typen verwendet wird.

Cheatsheet: Typisierter Lambda-Kalkül

$$\frac{\Gamma, \mathtt{p} : \pi \vdash b : \rho}{\Gamma \vdash \lambda \mathtt{p}.\, b : \pi \to \rho}\textsc{Abs} \qquad \frac{\Gamma \vdash f : \phi \to \alpha \qquad \Gamma \vdash x : \phi}{\Gamma \vdash f\ x : \alpha}\textsc{App}$$

$$\frac{\Gamma(\mathtt{t}) = \tau}{\Gamma \vdash \mathtt{t} : \tau}\textsc{Var} \qquad \frac{c \in \textsc{Const}}{\Gamma \vdash c : \tau_c}\textsc{Const}$$

- Typvariablen: τ, α, π, ρ
- Funktionstypen: $\tau_1 \to \tau_2$, rechtsassoziativ
- *Typisierungsregeln sind eindeutig*: Eine Regel pro Termform

Was bedeuten eigentlich $\vdash$, Γ und $:$?

$$\lambda \texttt{a}.\, \lambda \texttt{f}.\, \texttt{f}\ (\texttt{a}\ \textit{true})$$

Um zu einem solchen Term ein Typisierungsproblem zu beschreiben, notieren wir:

$$\Gamma \vdash \lambda \texttt{a}.\, \lambda \texttt{f}.\, \texttt{f}\ (\texttt{a}\ \textit{true}) : \tau$$

„Im *Typkontext* Γ hat der Term den Typen τ."

- Γ: Enthält Typen für freie Variablen.
- Liste von Paaren aus Variablen und deren Typen
- Liste $\rightsquigarrow$ Reihenfolge ist wichtig

Γ in Aktion

$$\Gamma \vdash \mathtt{a} + 42 : \mathtt{int}$$

$$\textsc{Const} = \{42\}, \tau_{42} = \mathtt{int}$$

Damit die Aussage „$\mathtt{a} + 42$ hat in Γ den Typen `int`“ stimmt, müssen wir für Γ wählen:

Γ in Aktion

$$\Gamma \vdash \mathtt{a} + 42 : \mathtt{int}$$

$$\textsc{Const} = \{42\}, \tau_{42} = \mathtt{int}$$

Damit die Aussage „$\mathtt{a} + 42$ hat in Γ den Typen $\mathtt{int}$“ stimmt, müssen wir für Γ wählen:

- $\Gamma = \mathtt{a} : \mathtt{int}, + : \mathtt{int} \to \mathtt{int} \to \mathtt{int}$

Γ in Aktion

$$\Gamma \vdash \mathtt{a} + 42 : \mathtt{int}$$

$$\textsc{Const} = \{42\}, \tau_{42} = \mathtt{int}$$

Damit die Aussage „$\mathtt{a} + 42$ hat in Γ den Typen $\mathtt{int}$“ stimmt, müssen wir für Γ wählen:

- $\Gamma = \mathtt{a} : \mathtt{int}, + : \mathtt{int} \rightarrow \mathtt{int} \rightarrow \mathtt{int}$
- Allgemeiner: $\Gamma = \mathtt{a} : \alpha, + : \alpha \rightarrow \mathtt{int} \rightarrow \mathtt{int}$

Typisierungsregel für Lambdas

- „Unter Einfügung des Typs π von $\mathtt{p}$ in den Kontext…“
- „… ist b als Funktion von $\mathtt{p}$ typisierbar.“

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

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

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.“

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

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

Einfache Typisierungsregel für Variablen

- $\boxed{\text{„Der Typkontext } \Gamma \text{ enthält einen Typ } \tau \text{ für } \mathtt{t}.\text{“}}$

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

- Daraus folgt:
- $\boxed{\text{„Variable } \mathtt{t} \text{ hat im Kontext } \Gamma \text{ den Typ } \tau.\text{“}}$

Typisierung: Beispiel

$$\mathtt{x} : \mathtt{bool} \vdash \lambda \mathtt{f}.\, \mathtt{f}\ \mathtt{x} : (\mathtt{bool} \to \alpha) \to \alpha$$

„Unter der Annahme, dass $\mathtt{x}$ den Typ $\mathtt{bool}$ hat, hat $\lambda \mathtt{f}.\, \mathtt{f}\ \mathtt{x}$ den Typ $(\mathtt{bool} \to \alpha) \to \alpha$."

Typisierung: Beispiel

$$\frac{\underline{\texttt{x} : \texttt{bool}}, \underline{\texttt{f}} : \underline{\texttt{bool} \to \alpha} \vdash \underline{\texttt{f x}} : \underline{\alpha}}{\underline{\texttt{x} : \texttt{bool}} \vdash \lambda \underline{\texttt{f}}.\, \underline{\texttt{f x}} : (\underline{\texttt{bool} \to \alpha}) \to \underline{\alpha}} \textsc{Abs}$$

„Pattern-Matching“: Der äußerste Term ist ein Lambda, also wenden wir die $\textsc{Abs}$-Regel an.

$\underline{\Gamma} = \underline{\texttt{x} : \texttt{bool}}$

$\underline{p} = \underline{\texttt{f}}, \underline{\texttt{b}} = \underline{\texttt{f x}}$

$\underline{\pi} = \underline{\texttt{bool} \to \alpha}$

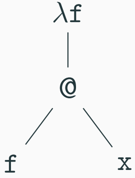
$\underline{\rho} = \underline{\alpha}$

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

Typisierung: Beispiel

$$\dfrac{\dfrac{\dfrac{\Gamma(\mathtt{f}) = \mathtt{bool} \to \alpha}{\Gamma \vdash \mathtt{f} : \mathtt{bool} \to \alpha}\,\textsc{Var} \qquad \dfrac{\Gamma(\mathtt{x}) = \mathtt{bool}}{\Gamma \vdash \mathtt{x} : \mathtt{bool}}\,\textsc{Var}}{\mathtt{x} : \mathtt{bool}, \mathtt{f} : \mathtt{bool} \to \alpha \vdash \mathtt{f}\ \mathtt{x} : \alpha}\,\textsc{App}}{\mathtt{x} : \mathtt{bool} \vdash \lambda \mathtt{f}.\, \mathtt{f}\ \mathtt{x} : (\mathtt{bool} \to \alpha) \to \alpha}\,\textsc{Abs}$$

$$\Gamma = x : \mathtt{bool}, f : \mathtt{bool} \to \alpha$$



$\lambda \mathtt{f}.\, f\ x$

Typinferenz

Problemstellung bei Typinferenz: Zu einem gegebenen Term den passenden Typ finden.

- Struktur des Terms erkennen. Wo sind:
  - Lambdas?
  - Funktionsanwendungen?
  - Variablen/Konstanten?
- Entsprechenden Baum aufstellen.
- Typgleichungen finden.
- Gleichungssystem unifizieren.

Von Typisierungsregeln zu Typinferenz

Beim Inferieren wird das Pattern-matching der Typen durch die *Unifikation* übernommen. Deswegen schreiben wir anstelle von konkreten Typen immer α_i und merken uns die Gleichungen für später:

$$\frac{\Gamma, \mathtt{p} : \pi \vdash b : \rho}{\Gamma \vdash \lambda \mathtt{p}.\, b : \pi \to \rho}\textsc{Abs} \quad \rightsquigarrow \quad \begin{array}{c} \dfrac{\Gamma, \mathtt{p} : \alpha_j \vdash b : \alpha_k}{\Gamma \vdash \lambda \mathtt{p}.\, b : \alpha_i}\textsc{Abs} \\ \{\alpha_i = \alpha_j \to \alpha_k\} \end{array}$$

Von Typisierungsregeln zu Typinferenz

Beim Inferieren wird das Pattern-matching der Typen durch die *Unifikation* übernommen. Deswegen schreiben wir anstelle von konkreten Typen immer α_i und merken uns die Gleichungen für später:

$$\frac{\Gamma \vdash f : \phi \to \alpha \qquad \Gamma \vdash x : \phi}{\Gamma \vdash f\ x : \alpha}\textsc{App} \quad \rightsquigarrow \quad \frac{\Gamma \vdash f : \alpha_j \qquad \Gamma \vdash x : \alpha_k}{\begin{array}{c}\Gamma \vdash f\ x : \alpha_i \\ \{\alpha_j = \alpha_k \to \alpha_i\}\end{array}}\textsc{App}$$

Von Typisierungsregeln zu Typinferenz

Beim Inferieren wird das Pattern-matching der Typen durch die *Unifikation* übernommen. Deswegen schreiben wir anstelle von konkreten Typen immer α_i und merken uns die Gleichungen für später:

$$\frac{\Gamma(\mathtt{t}) = \tau}{\Gamma \vdash \mathtt{t} : \tau}\textsc{Var} \quad \rightsquigarrow \quad \begin{array}{c} \dfrac{\Gamma(\mathtt{t}) = \alpha_j}{\Gamma \vdash \mathtt{t} : \alpha_i}\textsc{Var} \\ \{\alpha_i = \alpha_j\} \end{array}$$

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
  - Allgemeinster Unifikator (mgu)

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

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

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

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

Herleitungsbaum: Beispiel

$$\vdash \lambda \mathtt{x}.\, \lambda \mathtt{y}.\, \mathtt{x} : \alpha_1$$

Beispielhafte Aufgabenstellung: Finde den Typen α_1.

Herleitungsbaum: Beispiel

$$\frac{\underline{\mathtt{x}} : \alpha_2 \vdash \underline{\lambda \mathtt{y}.\, x} : \alpha_3}{\vdash \lambda \underline{\mathtt{x}}.\, \underline{\lambda \mathtt{y}.\, x} : \alpha_1}\textsc{Abs}$$

Typgleichungen:

$$C = \{\underline{\alpha_1 = \alpha_2 \to \alpha_3}\}$$

Herleitungsbaum: Beispiel

$$\dfrac{\dfrac{\underline{\mathtt{x} : \alpha_2}, \underline{\mathtt{y}} : \alpha_4 \vdash \underline{\mathtt{x}} : \alpha_5}{\underline{\mathtt{x} : \alpha_2} \vdash \lambda \underline{\mathtt{y}}.\, \underline{\mathtt{x}} : \alpha_3}\,\textsc{Abs}}{\vdash \lambda \mathrm{x}.\, \lambda \mathrm{y}.\, \mathrm{x} : \alpha_1}\,\textsc{Abs}$$

Typgleichungen:

$$\begin{aligned} C = \{&\alpha_1 = \alpha_2 \to \alpha_3 \\ &,\underline{\alpha_3 = \alpha_4 \to \alpha_5}\} \end{aligned}$$

Herleitungsbaum: Beispiel

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

Typgleichungen:

$$\begin{aligned} C = \{&\alpha_1 = \alpha_2 \to \alpha_3 \\ ,&\alpha_3 = \alpha_4 \to \alpha_5 \\ ,&\underline{\alpha_5 = \alpha_2}\} \end{aligned}$$

Herleitungsbaum: Aufgabe

$$\frac{\dots}{\vdash \lambda \texttt{f}.\, \texttt{f}\ (\lambda \texttt{x}.\, \texttt{x}) : \alpha_1}\textsc{Abs}$$

Findet den Typen α_1. Teilpunkte gibt es für:

- Herleitungsbaum,
- Typgleichungsmenge C,
- Unifikation per Robinsonalgorithmus.

Herleitungsbaum: Aufgabe

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

$$\Gamma = \texttt{f} : \alpha_2, \texttt{x} : \alpha_6$$

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

Let-Polymorphismus

Let-Polymorphismus: Motivation

$$\lambda\texttt{f}.\,\texttt{f}\ \texttt{f}$$

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

Let-Polymorphismus: Motivation

$$\lambda \texttt{f}.\, \texttt{f f}$$

- Diese Funktion verwendet `f` auf zwei Arten:
  - $\alpha \to \alpha$: Rechte Seite.
  - $(\alpha \to \alpha) \to (\alpha \to \alpha)$: Linke Seite, nimmt $\texttt{f} : \alpha \to \alpha$ 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 *neue Typvariablen*, 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:

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

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 einem $\textsc{Let}$-Term wird ein Typschema eingeführt:

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

$$\begin{aligned}
\sigma_{let} &= mgu(C_{let}) \\
\Gamma' &= \sigma_{let}(\Gamma), \texttt{x} : ta(\sigma_{let}(\alpha_i), \sigma_{let}(\Gamma)) \\
ta(\tau, \Gamma) &= \forall \alpha_1. \dots \forall \alpha_n.\tau \qquad \{\alpha_1, \dots, \alpha_n\} = FV(\tau) \setminus FV(\Gamma) \\
C'_{let} &= \{\alpha_n = \sigma_{let}(\alpha_n) \mid \sigma_{let}(\alpha_n) \text{ ist definiert}\}
\end{aligned}$$

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

Beispiel: Let-Polymorphismus

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

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

`flip flip`

$$\frac{\Gamma \vdash \lambda \mathtt{f}.\, \lambda \mathtt{x}.\, \lambda \mathtt{y}.\, \mathtt{f\ y\ x} : \alpha_2 \qquad \Gamma' \vdash \mathtt{flip}\, \mathtt{flip}\, 17\ \mathtt{(+)}\ 25 : \alpha_3}{\Gamma \vdash \mathtt{let\ \ flip} = \lambda \mathtt{f}.\, \lambda \mathtt{x}.\, \lambda \mathtt{y}.\, \mathtt{f\ y\ x\ \ in\ \ flip}\, \mathtt{flip}\, 17\ \mathtt{(+)}\ 25 : \alpha_1}\textsc{Let}$$

$$\tau_{17}, \tau_{25} = \mathtt{int}, \Gamma = \mathtt{(+)} : \mathtt{int} \to \mathtt{int} \to \mathtt{int}$$

- Aus `Prelude`: `flip f x y = f y x`
- Welchen Typ hat `flip`?
- Welchen Typ hat `flip flip`?

flip flip

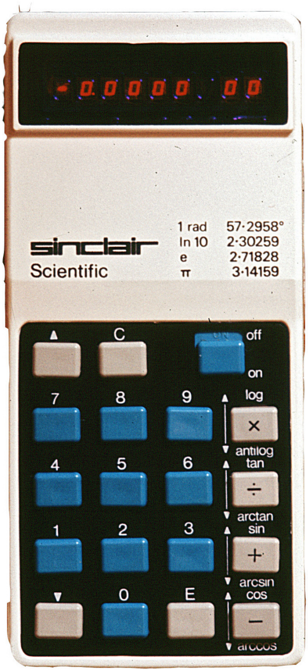
$$\frac{\Gamma \vdash \lambda \mathtt{f}.\, \lambda \mathtt{x}.\, \lambda \mathtt{y}.\, \mathtt{f\ y\ x} : \alpha_2 \qquad \Gamma' \vdash \mathtt{flip}\, \mathtt{flip}\, 17\ \mathtt{(+)}\ 25 : \alpha_3}{\Gamma \vdash \mathtt{let\ \ flip} = \lambda \mathtt{f}.\, \lambda \mathtt{x}.\, \lambda \mathtt{y}.\, \mathtt{f\ y\ x\ \ in\ \ flip}\, \mathtt{flip}\, 17\ \mathtt{(+)}\ 25 : \alpha_1}\,\textsc{Let}$$

$$\tau_{17}, \tau_{25} = \mathtt{int}, \Gamma = \mathtt{(+)} : \mathtt{int} \to \mathtt{int} \to \mathtt{int}$$

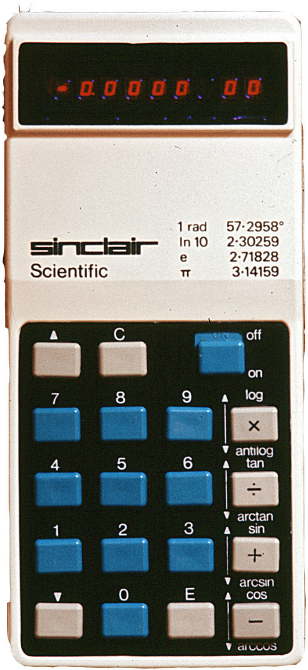
- Aus `Prelude`: `flip f x y = f y x`
- Welchen Typ hat `flip`?
- Welchen Typ hat `flip flip`?
- Überlegt euch den Typ von `flip` und überprüft ihn in GHCi.
- Was ist Γ'?
- Führt Typinferenz für den rechten Teilbaum durch.

Prolog

- Welche Taste fehlt hier?



Sinclair Scientific (1974)



- Welche Taste fehlt hier?
- Eingegebene Zahlen landen auf einem Stack
- $\times / \div / + / -$ wenden ihre Operation auf die obersten Stackelemente an
- Ergebnis landet wieder auf dem Stack
- Beispiel:
  $2 \times (7 + 14) \rightsquigarrow 2\ 7\ 14 + \times$
- *Umgekehrte Polnische Notation*

Sinclair Scientific (1974)

UPN-Rechner

```
eval(W, OutStack) :- evalRec([], W, OutStack1),
                     reverse(OutStack1, OutStack).
evalRec(Stack, [], Stack).
evalRec(Stack, [N | T], OutStack) :-
    number(N),
    evalRec([N | Stack], T, OutStack).
evalRec(Stack, [Op | T], OutStack) :-
    evalOp(Op, Stack, OutStack1),
    evalRec(OutStack1, T, OutStack).

evalOp(plus, [X, Y | T], [Z | T]) :-
    Z is X + Y.
```

- Beispiel: `eval([15, 27, plus], [42]).`
- Erweitert `evalOp` um $\times / \div / -$!

Übersetzung in UPN

```
% compile übersetzt einen mathematischen
% Ausdruck nach UPN
compile(N, [N]) :- number(N).
compile(X + Y, L) :-
    compile(X, Xc),
    compile(Y, Yc),
    ...
```

- Vervollständigt `compile/2`!
- Beispiel:
  `compile(2 * (7 + 14), [2, 7, 14, plus, mul])`

Haskell

Pixelflut

Mit dem Pixelflut-Protokoll können wir einzelne Pixel per TCP auf eine Leinwand malen. Es gibt zwei einfache Befehle:

- `SIZE`: Fragt die Größe der Leinwand ab
- `PX x y rrggbb`: Zeichnet einen Pixel der Farbe `rrggbb`

Mit `nc` oder `telnet` könnt ihr euch mit meinem Pixelflut-Server verbinden:

- `nc 172.17.8.144 3000`
- `telnet 172.17.8.144 3000`

Verwendet die Vorlage in `demos/Pixelflut.hs` um per Haskell Pixel zu malen!