Tutorium 08: Typisierung

Paul Brinkmeier

14. Dezember 2021

Tutorium Programmierparadigmen am KIT

ÜB 5, 6

5.1 — Klammerung im λ-Kalkül

$$c_0\ c_1\ (c_2\ c_3\ c_4) c_5 =$$

5.1 — Klammerung im λ-Kalkül

$$
\begin{aligned}
c_0\ c_1\ (c_2\ c_3\ c_4) c_5 &= \qquad \underline{(}\underline{(}c_0\ c_1\underline{)}\ (\underline{(}c_2\ c_3\underline{)}\ c_4)\underline{)}\ c_5 && (1)\\
(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &=
\end{aligned}
$$

5.1 — Klammerung im λ-Kalkül

$$
\begin{align}
c_0\ c_1\ (c_2\ c_3\ c_4) c_5 &= ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5 \tag{1}\\
(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &= ((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) \tag{2}\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6) &=
\end{align}
$$

5.1 — Klammerung im λ-Kalkül

$$c_0\ c_1\ (c_2\ c_3\ c_4) c_5 = \quad ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5 \tag{1}$$

$$(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) = \quad ((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) \tag{2}$$

$$c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6) = \quad ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ (c_5\ c_6) \tag{3}$$

$$c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5\ c_6 =$$

5.1 — Klammerung im λ-Kalkül

$$c_0\ c_1\ (c_2\ c_3\ c_4) c_5 = ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5 \tag{1}$$

$$(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) = ((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) \tag{2}$$

$$c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6) = ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ (c_5\ c_6) \tag{3}$$

$$c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5\ c_6 = (((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5)\ c_6 \tag{4}$$

$$c_0\ (c_1\ (c_2\ c_3\ c_4))\ c_5\ c_6 =$$

5.1 — Klammerung im λ-Kalkül

$$
\begin{aligned}
c_0\ c_1\ (c_2\ c_3\ c_4) c_5 &= ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5 && (1)\\
(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &= ((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) && (2)\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6) &= ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ (c_5\ c_6) && (3)\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5\ c_6 &= (((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5)\ c_6 && (4)\\
c_0\ (c_1\ (c_2\ c_3\ c_4))\ c_5\ c_6 &= ((c_0\ (c_1\ ((c_2\ c_3)\ c_4)))\ c_5)\ c_6 && (5)\\
(\lambda y.c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &=
\end{aligned}
$$

5.1 — Klammerung im λ-Kalkül

$$
\begin{aligned}
c_0\ c_1\ (c_2\ c_3\ c_4) c_5 &= \quad ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5 && (1)\\
(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &= \quad ((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) && (2)\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6) &= \quad ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ (c_5\ c_6) && (3)\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5\ c_6 &= \quad (((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5)\ c_6 && (4)\\
c_0\ (c_1\ (c_2\ c_3\ c_4))\ c_5\ c_6 &= \quad ((c_0\ (c_1\ ((c_2\ c_3)\ c_4)))\ c_5)\ c_6 && (5)\\
(\lambda y.c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &= \quad (\lambda y.(c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) && (6)\\
(\lambda y.(c_0\ (\lambda z.c_1\ c_2)))\ (c_3\ c_4\ c_5) &=
\end{aligned}
$$

5.1 — Klammerung im λ-Kalkül

$$
\begin{aligned}
c_0\ c_1\ (c_2\ c_3\ c_4) c_5 &= ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5 && (1)\\
(c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &= ((c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) && (2)\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ (c_5\ c_6) &= ((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ (c_5\ c_6) && (3)\\
c_0\ c_1\ (c_2\ c_3\ c_4)\ c_5\ c_6 &= (((c_0\ c_1)\ ((c_2\ c_3)\ c_4))\ c_5)\ c_6 && (4)\\
c_0\ (c_1\ (c_2\ c_3\ c_4))\ c_5\ c_6 &= ((c_0\ (c_1\ ((c_2\ c_3)\ c_4)))\ c_5)\ c_6 && (5)\\
(\lambda y.c_0\ c_1\ c_2)\ (c_3\ c_4\ c_5) &= (\lambda y.(c_0\ c_1)\ c_2)\ ((c_3\ c_4)\ c_5) && (6)\\
(\lambda y.(c_0\ (\lambda z.c_1\ c_2)))\ (c_3\ c_4\ c_5) &= (\lambda y.(c_0\ (\lambda z.(c_1\ c_2))))\ ((c_3\ c_4)\ c_5) && (7)
\end{aligned}
$$

- Funktionsanwendungen sind linksassoziativ, wie in Haskell
- Eigentlich: In Haskell sind Funktionsanwendungen linksassoziativ, wie im λ-Kalkül

5.1 — Klammerung im λ-Kalkül

$$(\lambda y.y)\ c_0 \stackrel{?}{=} \lambda y.y\ c_0 \tag{1}$$

$$\lambda y.(y\ c_0) \stackrel{?}{=} \lambda y.y\ c_0 \tag{2}$$

- Term 1 $\approx$ `App (Abs "y" (Var "y")) (Var "c0")`
- Term 2 $\approx$ `Abs "y" (App (Var "y") (Var "c0"))`
- Funktionsanwendung bindet am stärkesten $\rightsquigarrow$ (2)

5.1 — Klammerung im λ-Kalkül

$$(\lambda y.y)\ c_0 \stackrel{?}{=} \lambda y.y\ c_0 \tag{1}$$

$$\lambda y.(y\ c_0) \stackrel{?}{=} \lambda y.y\ c_0 \tag{2}$$

- Term 1 $\approx$ `app(abs(y, y), c0)`
- Term 2 $\approx$ `abs(y, app(y, c0))`
- Funktionsanwendung bindet am stärkesten $\rightsquigarrow$ (2)

1.3 — Klammerung im λ-Kalkül

$$((x)\ c_0)\,[x \to \lambda y.y] = (\lambda y.y)\ c_0 \tag{1}$$

$$(x\ c_0)\,[x \to (\lambda y.y)] = (\lambda y.y)\ c_0 \tag{2}$$

$$(x\ c_0)\,[x \to \lambda y.y] = (\lambda y.y)\ c_0 \tag{3}$$

- Alle drei Substitutionen führen zum selben Ergebnis
- Klammern auf der rechten Seite haben nichts mit der Substution zu tun
- „Für beliebiges t repräsentieren t und (t) den gleichen λ-Term" stimmt

1.4 — Klammerung im λ-Kalkül

Angenommen, $x = c_0\ c_1$.
Welche der folgenden Aussagen gelten?

$$c_0\ c_1\ c_2 = x\ c_2 \tag{1}$$

$$c_2\ c_0\ c_1 = c_2\ x \tag{2}$$

$$c_2\ (c_3\ c_4)\ c_0\ c_1 = c_2\ (c_3\ c_4)\ x \tag{3}$$

$$c_2\ (c_0\ c_1\ c_3) c_4 = c_2\ (x\ c_3)\ c_4 \tag{4}$$

1.4 — Klammerung im λ-Kalkül

Angenommen, $x = c_0\ c_1$.

Welche der folgenden Aussagen gelten?

$$c_0\ c_1\ c_2 = x\ c_2 \tag{1}$$

$$c_2\ c_0\ c_1 = c_2\ x \tag{2}$$

$$c_2\ (c_3\ c_4)\ c_0\ c_1 = c_2\ (c_3\ c_4)\ x \tag{3}$$

$$c_2\ (c_0\ c_1\ c_3) c_4 = c_2\ (x\ c_3)\ c_4 \tag{4}$$

- 1 und 4 gelten
- $c_2\ c_0\ c_1 = \underline{(}c_2\ c_0\underline{)}\ c_1 \neq c_2\ \underline{(}c_0\ c_1\underline{)} = c_2\ x$
- $c_2\ (c_3\ c_4)\ c_0\ c_1 \overset{*}{\neq} c_2\ (c_3\ c_4)\ x$

1.4 — Klammerung im λ-Kalkül

$$\underline{(\lambda a.\, a)}\,(\lambda b.\, b)\,(\underline{(\lambda c.\, c)}\,(\underline{(\lambda d.\, d)}\,(\lambda e.\, e)\,(\lambda f.\, f)))\,(\lambda g.\, g)\,(\underline{(\lambda h.\, h)}\,(\lambda i.\, i))$$

- Gute Übung um Redex zu finden
- Warum ist $\lambda g.\, g$ nicht die linke Seite eines Redex?

Wiederholung: Auswertungsstrategien

$$\overbrace{\underline{\text{succ}}\ \overbrace{(\underline{\text{succ}}\ c_0)}^{\text{Redex 2}}}^{\text{Redex 1}}$$

mit

$$\begin{aligned} c_0 &= \lambda s.\, \lambda z.\, z \\ \text{succ} &= \lambda n.\, \lambda s.\, \lambda z.\, s\ (n\ s\ z) \end{aligned}$$

- Welcher Redex soll zuerst ausgewertet werden?
- $\rightsquigarrow$ verschiedene Auswertungsstrategien

Wiederholung: Normalreihenfolge

$$
\begin{array}{ll}
 & \underline{\mathsf{succ}}\ (\underline{\mathsf{succ}}\ c_0) \\
\Rightarrow_\beta & \lambda s.\, \lambda z.\, s\ ((\underline{\mathsf{succ}}\ c_0)\ s\ z) \\
\Rightarrow_\beta & \lambda s.\, \lambda z.\, s\ (\underline{(\lambda s.\, \lambda z.\, s\ (\underline{c_0}\ s\ z))}\ s\ z) \\
\Rightarrow_\beta^2 & \lambda s.\, \lambda z.\, s\ (s\ (\underline{c_0}\ s\ z)) \\
\Rightarrow_\beta^2 & \lambda s.\, \lambda z.\, s\ (s\ z)\ \not\Rightarrow
\end{array}
$$

Normalreihenfolge: Linkester Redex zuerst.

Wiederholung: Call-by-Name

$$\underline{\mathsf{succ}}\ (\underline{\mathsf{succ}}\ c_0)$$

$$\Rightarrow_\beta \quad \lambda s.\, \lambda z.\, s\ ((\underline{\mathsf{succ}}\ c_0)\ s\ z) \not\Rightarrow_{\mathsf{CbN}}$$

Call-by-Name: Linkester Redex zuerst, aber:

- Funktions*inhalte* werden nicht weiter reduziert
- $\rightsquigarrow$ Betrachte nur Redexe, die nicht von einem λ umgeben sind
- So (ähnlich) funktioniert auch Laziness in Haskell

Wiederholung: Call-by-Value

$$
\begin{array}{rc}
& \underline{\mathrm{succ}}\ (\underline{\mathrm{succ}}\ c_0) \\
\Rightarrow_\beta & \underline{\mathrm{succ}}\ (\lambda s.\,\lambda z.\, s\ (\underline{c_0}\ s\ z)) \\
\Rightarrow_\beta & \lambda s.\,\lambda z.\, \underline{(\lambda s.\,\lambda z.\, s\ (\underline{c_0}\ s\ z))}\ s\ z \not\Rightarrow_{\mathrm{CbV}}
\end{array}
$$

Call-by-Value: Linkester Redex zuerst, aber:

- Funktionsinhalte werden nicht weiter reduziert
- $\rightsquigarrow$ Betrachte nur Redexe, die nicht von einem λ umgeben sind
- Berechne *Argumente vor dem Einsetzen*
- $\rightsquigarrow$ Betrachte nur Redexe, deren Argument *unter CbV* nicht weiter reduziert werden muss

5.2 — Redexe, Auswertungsstrategie

$$\overbrace{(\lambda t.\,\lambda f.\,f)\;((\lambda y.\,(\lambda x.\,x\;x)\;(\lambda x.\,x\;x))\;(\overbrace{(\lambda x.\,x)\;(\lambda x.\,x)}^{\text{CbV}}))}^{\text{NRF, CbN}}\;(\lambda t.\,\lambda f.\,f)$$

- NRF: Linkester Redex
- CbN: Linkester Redex, nicht in Lambda
- CbV: Linkester Redex, nicht in Lambda, Argumente zuerst

5.2 — Redexe, Auswertungsstrategie

$$\lambda y.\, \overbrace{\underline{(\lambda z.\, \underline{(\lambda x.\, x)}\, (\lambda x.\, x)\; z)}\; y}^{\text{NRF}}$$

- NRF: Linkester Redex
- CbN: Linkester Redex, nicht in Lambda
- CbV: Linkester Redex, nicht in Lambda, Argumente zuerst

5.3 — Church-Zahlen in Haskell

```haskell
module ChurchNumbers where

type Church t = (t -> t) -> t -> t

int2church 0 = \s z -> z
int2church n = \s z -> s (int2church (n - 1) s z)

church2int cn = cn (+1) 0
```

- Alte Klausuraufgabe
- Typisch: Lambda-Wissen hilft beim Lösen

5.4 — Church-Paare

$$pair = \lambda a.\, \lambda b.\, \lambda p.\, p\ a\ b$$
$$(3, 5) \approx pair\ c_3\ c_5 \Rightarrow_\beta^2 \lambda p.\, p\ c_3\ c_5$$
$$fst = \lambda p.\, p\ (\lambda a.\, \lambda b.\, a)$$

- „Fertiges“ Paar nimmt Funktion p als Argument
- p wählt ersten oder zweiten Eintrag

5.4 — Church-Paare

$$\begin{aligned} pair &= \lambda a.\, \lambda b.\, \lambda p.\, p\ a\ b \\ (3,5) \approx pair\ c_3\ c_5 &\Rightarrow_\beta^2 \lambda p.\, p\ c_3\ c_5 \\ fst &= \lambda p.\, p\ (\lambda a.\, \lambda b.\, a) \end{aligned}$$

- $snd = \lambda p.\, p\ (\lambda a.\, \lambda b.\, b)$
- $next$: Berechne zu (n, m): $(m, m+1)$
- $next = \lambda p.\, pair\ (snd\ p)\ (succ\ (snd\ p))$
- $pred = \lambda n.\, fst\ (n\ next\ (pair\ c_0\ c_0))$
- $sub = \lambda m.\, \lambda n.\, n\ pred\ m$

Typisierung

Klausuraufgabe WS16/18 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

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

typisierbar? Begründen Sie ihre Antwort *kurz*.

Klausuraufgabe WS16/18 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.

$$\begin{aligned}
\sigma_1 &= \texttt{unify}(\{\alpha_9 = \alpha_{10} \to \alpha_8, \alpha_9 = \alpha_4, \alpha_{10} = \texttt{bool}\}) \\
&= ... = [\alpha_9 \Rightarrow \texttt{bool} \to \alpha_8, \alpha_4 \Rightarrow \texttt{bool} \to \alpha_8, \alpha_{10} \Rightarrow \texttt{bool}] \\
\sigma_2 &= \texttt{unify}(\{\alpha_{12} = \alpha_{13} \to \alpha_{11}, \alpha_{12} = \alpha_4, \alpha_{13} = \texttt{int}\}) \\
&= ... = [\alpha_{12} \Rightarrow \texttt{int} \to \alpha_{11}, \alpha_4 \Rightarrow \texttt{int} \to \alpha_{11}, \alpha_{13} \Rightarrow \texttt{int}]
\end{aligned}$$

Klausuraufgabe WS16/18 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 WS16/18 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

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

typisierbar? Begründen Sie ihre Antwort *kurz*.

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

Cheatsheet: Typisierter Lambda-Kalkül

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

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

- 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 $:$?

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

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

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

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

- Γ: Enthält Typen für freie Variablen.
- ... $\vdash$ ... : ... — Notation für Typisierungsproblem.

Γ in Aktion

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

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

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

Γ in Aktion

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

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

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

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

Γ in Aktion

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

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

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

- $\Gamma = a : \texttt{int}, + : \texttt{int} \to \texttt{int} \to \texttt{int}$
- Allgemeiner: $\Gamma = a : \alpha, + : \alpha \to \texttt{int} \to \texttt{int}$

Typisierungsregel für Lambdas

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

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

- Daraus folgt:
- „$\lambda p.\, b$ 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.“

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

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

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."

Typisierung: Beispiel

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

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

Typisierung: Beispiel

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

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

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

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

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

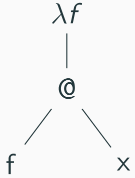
$$\underline{\rho} = \underline{\alpha}$$

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

Typisierung: Beispiel

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

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



$\lambda f.\, f\ x$

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:

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

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:

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

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
  - I.d.R.: Allgemeinster Unifikator (mgu)

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

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

Herleitungsbaum: Beispiel

$$\vdash \lambda x.\, \lambda y.\, x : \alpha_1$$

Beispielhafte Aufgabenstellung: Finde den Typen α_1.

Herleitungsbaum: Beispiel

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

Typgleichungen:

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

Herleitungsbaum: Beispiel

$$\dfrac{\dfrac{\underline{x : \alpha_2}, \underline{y} : \alpha_4 \vdash \underline{x} : \alpha_5}{\underline{x : \alpha_2} \vdash \lambda \underline{y}.\, \underline{x} : \alpha_3}\text{ABS}}{\vdash \lambda x.\, \lambda y.\, x : \alpha_1}\text{ABS}$$

Typgleichungen:

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

Herleitungsbaum: Beispiel

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

Typgleichungen:

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

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

Findet den Typen α_1. Teilpunkte gibt es für:

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

Herleitungsbaum: Aufgabe

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

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

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

Let-Polymorphismus

Let-Polymorphismus: Motivation

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

- 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 *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:

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

- *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

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

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