Polymorphie

Polymorphismus

Polymorphe Funktionen: Verhalten hängt nicht vom konkreten Typ ab!

- Operationen auf Containern, z.B. Listen
- Hängen nicht vom Typ τ der Elemente ab
- Beispiel: Aneinanderfügen von Listen

```
app []     right  = right
app (x:xs) right  = x:(app xs right)
```

Polymorphe Funktionen: Haben unendlich viele Typen, z.B.:

```
app :: [Int] -> [Int] -> [Int]         app :: [Bool] -> [Bool] -> [Bool]
app :: [[Int]] -> [[Int]] -> [[Int]] app ::  ...

app :: [Bool -> Int] -> [Bool -> Int] -> [Bool -> Int]
```

Allgemeinster Typ: `app ::` $[\alpha]$ `->` $[\alpha]$ `->` $[\alpha]$

- Für beliebige Typen $\alpha \Rightarrow \alpha$ implizit $\forall$ quantifiziert
- Polymorphismus im λ-Kalkül: `let`-Polymorphismus

`let`-Polymorphismus

Beispielprogramm: $P = \ \mathbf{let}\ \mathtt{f} = \lambda\mathtt{x}.\,\mathtt{2}\ \mathbf{in}\ \mathtt{f}\,(\mathtt{f}\ \mathtt{true})$

- f polymorphe Hilfsfunktion: Anwendung auf `true`, dann auf `2`
- Ziel: Wollen solche Ausdrücke typen können

Mögliche Kodierung: $\mathbf{let}\ x = t_1\ \mathbf{in}\ t_2$

- als syntaktischen Zucker für $(\lambda x.\ t_2)\ t_1$
- Aber so: $P = (\lambda\mathtt{f}.\ \mathtt{f}\,(\mathtt{f}\ \mathtt{true}))\ (\lambda\mathtt{x}.\ \mathtt{2})$ nicht typisierbar, denn in

$$\textsc{Abs}\ \frac{\mathtt{f} : \tau_{\mathtt{f}} \vdash \mathtt{f}\,(\mathtt{f}\ \mathtt{true}) : \ \ldots}{\vdash \lambda\mathtt{f}.\ \mathtt{f}\,(\mathtt{f}\ \mathtt{true}) : \ \ldots}$$

müsste $\tau_{\mathtt{f}} = \text{bool} \to \text{int}$ und $\tau_{\mathtt{f}} = \text{int} \to \text{int}$ in Typkontext eingetragen werden $\Rightarrow$ nicht möglich

Stattdessen: $\mathbf{let}\ x = t_1\ \mathbf{in}\ t_2$

- als neues Konstrukt im λ-Kalkül, neue Typregeln mit Typschemata

Typschemata

Typschema

Ein Typ der Gestalt $\forall\alpha_1.\,\forall\alpha_2.\;\ldots\;\forall\alpha_n.\,\tau$ heißt Typschema.
Es bindet freie Typvariablen $\alpha_1, \ldots, \alpha_n$ in τ

Beispiel: Typschema $\forall\alpha.\,\alpha \to \alpha$ steht für unendliche viele Typen, z.B.:

- $\mathsf{int} \to \mathsf{int}$, $\mathsf{bool} \to \mathsf{bool}$, ...
- $(\mathsf{bool} \to \mathsf{bool}) \to (\mathsf{bool} \to \mathsf{bool})$, $(\mathsf{int} \to \mathsf{bool}) \to (\mathsf{int} \to \mathsf{bool})$, ...

Instanziierung eines Typschemas

Für Nicht-Schema-Typen τ_2 ist der Typ $\tau\,[\alpha \mapsto \tau_2]$ eine Instanziierung vom Typschema $\forall\alpha.\,\tau$ Schreibweise: $(\forall\alpha.\,\tau) \succeq \tau\,[\alpha \mapsto \tau_2]$

Zum Beispiel:

- $\forall\alpha.\,\alpha \to \alpha \succeq \mathsf{int} \to \mathsf{int}$
- $\forall\alpha.\,\alpha \to \alpha \succeq (\mathsf{int} \to \mathsf{int}) \to (\mathsf{int} \to \mathsf{int})$
- $\mathsf{int} \succeq \mathsf{int}$

Aber:

- $\alpha \to \alpha \not\succeq \mathsf{int} \to \mathsf{int}$
- $\alpha \not\succeq \mathsf{bool}$
- $\forall\alpha.\,\alpha \to \alpha \not\succeq \mathsf{bool}$

Angepasste Regeln für Typschemata

Angepasste Regeln:

$$\textsc{Var}: \frac{\Gamma(x) = \tau' \qquad \tau' \succeq \tau}{\Gamma \vdash x : \tau}$$

$$\textsc{Abs}: \frac{\Gamma, x : \tau_1 \vdash t : \tau_2 \qquad \tau_1 \text{ kein Typschema}}{\Gamma \vdash \lambda x.\, t : \tau_1 \to \tau_2}$$

Const- und App-Regeln bleiben gleich.

λ-Gebundene Bezeichner niemals polymorph, vgl:

$$(\lambda \texttt{f}.\ \texttt{f}\ (\texttt{f}\ \texttt{true}))\ (\lambda \texttt{x}.\ \texttt{2})$$

Grund:

- Während Typisierung von $(\lambda \texttt{f}.\ t)$ unbekannt:
  Welcher Term t_f wird später übergeben: $(\lambda \texttt{f}.\ t)\ t_f$
- Anders bei Typisierung von $\texttt{let}\ f = t_f\ \texttt{in}\ t$

`let`-Polymorphismus mit Typschemata

Idee: `let`-gebundene Variablen getypt mit Typschema

Typabstraktion

Das Typschema $ta(\tau, \Gamma) = \forall\alpha_1.\ \forall\alpha_2.\ \ldots\ \forall\alpha_n.\ \tau$ heißt Typabstraktion von τ relativ zu Γ, wobei $\alpha_i \in FV(\tau) \setminus FV(\Gamma)$

Alle freien Typvariablen von τ quantifiziert, die nicht frei in Typannahmen Γ
$\Rightarrow$ Verhindere Abstraktion von globalen Typvariablen im Schema

Let-Typregel

$$\textsc{Let}: \frac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : ta(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : \tau_2}$$

- Effizient (meist „quasilinear“)
- Jedoch exponentieller Worst Case!

`let`-Polymorphismus: Beispiel

Kann $\texttt{let}\ \mathtt{f} = \lambda \mathtt{x}.\ 2\ \texttt{in}\ \mathtt{f}\ (\mathtt{f}\ \mathtt{true})$ getypt werden?

$$\dfrac{\dfrac{\dfrac{2 \in \mathit{Const}}{\mathtt{x} : \alpha \vdash 2 : \mathrm{int}}}{\vdash \lambda \mathtt{x}.\ 2 : \alpha \to \mathrm{int}} \qquad \ldots}{\vdash \texttt{let}\ \mathtt{f} = \lambda \mathtt{x}.\ 2\ \texttt{in}\ \mathtt{f}\ (\mathtt{f}\ \mathtt{true}) : ?}$$

$$\mathit{ta}(\alpha \to \mathrm{int},) = \forall \alpha.\ \alpha \to \mathrm{int} \qquad \begin{array}{l} (\forall \alpha.\ \alpha \to \mathrm{int}) \succeq \mathrm{bool} \to \mathrm{int} \\ (\forall \alpha.\ \alpha \to \mathrm{int}) \succeq \mathrm{int} \to \mathrm{int} \end{array}$$

$$\textsc{Let}: \dfrac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : \mathit{ta}(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : \tau_2}$$

let-Polymorphismus: Beispiel

Kann $\texttt{let f} = \lambda\texttt{x. 2 in f (f true)}$ getypt werden?

$$\dfrac{\dfrac{\dfrac{\Gamma(\texttt{f}) = \forall\alpha.\,\alpha \to \text{int} \quad \forall\alpha.\,\alpha \to \text{int} \succeq \text{int} \to \text{int}}{\Gamma \vdash \texttt{f} : \text{int} \to \text{int}} \qquad \ldots}{\underbrace{\texttt{f} : \forall\alpha.\,\alpha \to \text{int}}_{\Gamma} \vdash \texttt{f (f true)} : ?} \qquad \ldots}{\vdash \textbf{let}\ \texttt{f} = \lambda\texttt{x. 2}\ \textbf{in}\ \texttt{f (f true)} : ?}$$

$$ta(\alpha \to \text{int},) = \forall\alpha.\,\alpha \to \text{int} \qquad (\forall\alpha.\,\alpha \to \text{int}) \succeq \text{bool} \to \text{int}$$
$$(\forall\alpha.\,\alpha \to \text{int}) \succeq \text{int} \to \text{int}$$

$$\text{LET: } \dfrac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : ta(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \textbf{let}\ x = t_1\ \textbf{in}\ t_2 : \tau_2}$$

`let`-Polymorphismus: Beispiel

Kann $\mathbf{let}\ \mathtt{f} = \lambda\mathtt{x}.\ 2\ \mathbf{in}\ \mathtt{f}\ (\mathtt{f}\ \mathtt{true})$ getypt werden?

$$
\dfrac{
\dfrac{\Gamma(\mathtt{f}) = \forall\alpha.\,\alpha \to \mathrm{int} \quad \forall\alpha.\,\alpha \to \mathrm{int} \succeq \mathrm{int} \to \mathrm{int}}{\Gamma \vdash \mathtt{f} : \mathrm{int} \to \mathrm{int}}
\qquad
\dfrac{
\dfrac{\Gamma(\mathtt{f}) = \forall\alpha.\,\alpha \to \mathrm{int} \quad \forall\alpha.\,\alpha \to \mathrm{int} \succeq \mathrm{bool} \to \mathrm{int}}{\Gamma \vdash \mathtt{f} : \mathrm{bool} \to \mathrm{int}}
\qquad
\dfrac{\mathtt{true} \in \mathit{Const}}{\Gamma \vdash \mathtt{true} : \mathrm{bool}}
}{\Gamma \vdash \mathtt{f}\ \mathtt{true} : \mathrm{int}}
}{\underbrace{\mathtt{f} : \forall\alpha.\,\alpha \to \mathrm{int}}_{\Gamma} \vdash \mathtt{f}\ (\mathtt{f}\ \mathtt{true}) : \mathrm{int}}
$$

$$
\dfrac{\ldots \qquad \underbrace{\mathtt{f} : \forall\alpha.\,\alpha \to \mathrm{int}}_{\Gamma} \vdash \mathtt{f}\ (\mathtt{f}\ \mathtt{true}) : \mathrm{int}}{\vdash \mathbf{let}\ \mathtt{f} = \lambda\mathtt{x}.\ 2\ \mathbf{in}\ \mathtt{f}\ (\mathtt{f}\ \mathtt{true}) : \mathrm{int}}
$$

$$\mathit{ta}(\alpha \to \mathrm{int},) = \forall\alpha.\,\alpha \to \mathrm{int} \qquad (\forall\alpha.\,\alpha \to \mathrm{int}) \succeq \mathrm{bool} \to \mathrm{int}$$

$$(\forall\alpha.\,\alpha \to \mathrm{int}) \succeq \mathrm{int} \to \mathrm{int}$$

$$\textsc{Let}: \dfrac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : \mathit{ta}(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \mathbf{let}\ x = t_1\ \mathbf{in}\ t_2 : \tau_2}$$

$\texttt{let}$-Polymorphismus: Beispiel

Kann $\textbf{let}\ \texttt{f} = \lambda\texttt{x}.\ 2\ \textbf{in}\ \texttt{f}\ (\texttt{f}\ \texttt{true})$ getypt werden? Ja!

$$
\dfrac{
\dfrac{\dfrac{2 \in \mathit{Const}}{\texttt{x} : \alpha \vdash 2 : \text{int}}}{\vdash \lambda\texttt{x}.\ 2 : \alpha \to \text{int}}
\qquad
\dfrac{
\dfrac{\begin{array}{c}\Gamma(\texttt{f}) = \forall\alpha.\ \alpha \to \text{int} \\ \forall\alpha.\ \alpha \to \text{int} \succeq \text{int} \to \text{int}\end{array}}{\Gamma \vdash \texttt{f} : \text{int} \to \text{int}}
\quad
\dfrac{
\dfrac{\begin{array}{c}\Gamma(\texttt{f}) = \forall\alpha.\ \alpha \to \text{int} \\ \forall\alpha.\ \alpha \to \text{int} \succeq \text{bool} \to \text{int}\end{array}}{\Gamma \vdash \texttt{f} : \text{bool} \to \text{int}}
\quad
\dfrac{\texttt{true} \in \mathit{Const}}{\Gamma \vdash \texttt{true} : \text{bool}}
}{\Gamma \vdash \texttt{f}\ \texttt{true} : \text{int}}
}{\underbrace{\texttt{f} : \forall\alpha.\ \alpha \to \text{int}}_{\Gamma} \vdash \texttt{f}\ (\texttt{f}\ \texttt{true}) : \text{int}}
}{\vdash \textbf{let}\ \texttt{f} = \lambda\texttt{x}.\ 2\ \textbf{in}\ \texttt{f}\ (\texttt{f}\ \texttt{true}) : \text{int}}
$$

$$
\mathit{ta}(\alpha \to \text{int},) = \forall\alpha.\ \alpha \to \text{int} \qquad
\begin{array}{l}
(\forall\alpha.\ \alpha \to \text{int}) \succeq \text{bool} \to \text{int} \\
(\forall\alpha.\ \alpha \to \text{int}) \succeq \text{int} \to \text{int}
\end{array}
$$

$$
\text{LET}: \dfrac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : \mathit{ta}(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \textbf{let}\ x = t_1\ \textbf{in}\ t_2 : \tau_2}
$$

Typabstraktion/`let`-Typisierung

Beispiel: $\lambda \texttt{g}.\ \textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5$

Allgemeinster Typ: $(\text{int} \to \beta) \to \beta$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \texttt{y} : \text{int} \vdash \texttt{g} : \text{int} \to \beta \qquad \Gamma, \texttt{y} : \text{int} \vdash \texttt{y} : \text{int}}{\Gamma, \texttt{y} : \text{int} \vdash \texttt{g}\ \texttt{y} : \beta}}{\Gamma \vdash (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y}) : \text{int} \to \beta} \qquad \dfrac{\dots}{\Gamma, \texttt{f} : \text{int} \to \beta \vdash \texttt{f}\ 5 : \beta}}{\Gamma \vdash (\textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5) : \beta}}{\vdash (\lambda \texttt{g}.\ \textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5) : (\text{int} \to \beta) \to \beta}$$

$$\Gamma = \texttt{g} : \text{int} \to \beta \qquad\qquad ta(\text{int} \to \beta, \Gamma) = \text{int} \to \beta$$

- Keine Abstraktion $\forall \beta.$ in $\text{int} \to \beta$, da $\beta \in FV(\Gamma)$
- Ohne diese Einschränkung: falsche Typen möglich!

$$\textsc{Let}: \dfrac{\Gamma \vdash t_1 : \tau_1 \qquad \Gamma, x : ta(\tau_1, \Gamma) \vdash t_2 : \tau_2}{\Gamma \vdash \textbf{let}\ x = t_1\ \textbf{in}\ t_2 : \tau_2}$$

Typabstraktion/`let`-Typisierung

Beispiel: $\lambda \texttt{g}.\ \textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5$

Was wenn man in *ta* die Einschränkung $\alpha_i \notin FV(\Gamma)$ weglässt?

- Dann Falscher Typ $(\alpha \to \beta) \to \beta$ herleitbar:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \texttt{y} : \alpha \vdash \texttt{g} : \alpha \to \beta \qquad \Gamma, \texttt{y} : \alpha \vdash \texttt{y} : \alpha}{\Gamma, \texttt{y} : \alpha \vdash \texttt{g}\ \texttt{y} : \beta}}{\Gamma \vdash (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y}) : \alpha \to \beta} \qquad \dfrac{\dfrac{\dfrac{\Gamma, \texttt{f} : \forall \alpha.\ \alpha \to \beta \vdash \texttt{f} : \forall \alpha.\ \alpha \to \beta}{\Gamma, \texttt{f} : \forall \alpha.\ \alpha \to \beta \vdash \texttt{f} : \text{int} \to \beta}}{\dots}}{\Gamma, \texttt{f} : \forall \alpha.\ \alpha \to \beta \vdash \texttt{f}\ 5 : \beta}}{\Gamma \vdash (\textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5) : \beta}}{\vdash (\lambda \texttt{g}.\ \textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5) : (\alpha \to \beta) \to \beta}$$

$$\Gamma = \texttt{g} : \alpha \to \beta \qquad\qquad ta(\alpha \to \beta, \Gamma) \neq \forall \alpha.\ \alpha \to \beta$$

Dies führt später zu Laufzeitfehler, z.B. in

$$(\lambda \texttt{g}.\ \textbf{let}\ \texttt{f} = (\lambda \texttt{y}.\ \texttt{g}\ \texttt{y})\ \textbf{in}\ \texttt{f}\ 5)\ (\lambda \texttt{x}.\ \texttt{x} \| \texttt{true})$$

- erlaubt laut Herleitung ($\alpha = \beta =$ bool), aber Absturz bei $5 \| \texttt{true}$

Also: Abstraktion $\forall \alpha.\ \alpha \to \beta$ unzulässig, da α in Γ frei vorkommt

- Intuition: $f \overset{\eta}{=} g \ \Rightarrow$ Typen gekoppelt
  Anwendung $(\texttt{f}\ 5)$ erzwingt Instanziierung $\forall \alpha.\ \alpha \to \beta \succeq \text{int} \to \beta$
  Aber: falsche Abstraktion $\forall \alpha.\ \alpha \to \beta$ entkoppelt Typen von $\texttt{f}$, $\texttt{g}$

Typinferenz für `let`

Beispiel: $\texttt{\textbf{let}}\ \texttt{f} = \lambda\texttt{x}.\ \texttt{g y y}\ \texttt{\textbf{in}}\ \texttt{f}\ 3$

Mit:

$\Gamma \quad = \texttt{y} : \beta,\ \texttt{g} : \beta \to \beta \to \beta$

$$
\begin{aligned}
C &= C_0 \cup C_{\texttt{let}} \\
&= \{\alpha_1 = \alpha_9\} \cup \{ \\
&\alpha_2 = \alpha_3 \to \alpha_4 \\
&\alpha_5 = \alpha_6 \to \alpha_4 \\
&\alpha_7 = \alpha_8 \to \alpha_5 \\
&\alpha_7 = \beta \to \beta \to \beta \\
&\alpha_8 = \beta \\
&\}
\end{aligned}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \texttt{x} : \alpha_3 \vdash \texttt{g} : \alpha_7 \qquad \Gamma, \texttt{x} : \alpha_3 \vdash \texttt{y} : \alpha_8}{\Gamma, \texttt{x} : \alpha_3 \vdash \texttt{g y} : \alpha_5} \qquad \Gamma, \texttt{x} : \alpha_3 \vdash \texttt{y} : \alpha_6}{\Gamma, \texttt{x} : \alpha_3 \vdash \texttt{g y y} : \alpha_4}}{\Gamma \vdash \lambda\texttt{x}.\ \texttt{g y y} : \alpha_2} \qquad \Gamma, \texttt{f} : \mathit{ta}(\alpha_2, \Gamma) \vdash \texttt{f}\ 3 : \alpha_9}{\Gamma \vdash \texttt{\textbf{let}}\ \texttt{f} = \lambda\texttt{x}.\ \texttt{g y y}\ \texttt{\textbf{in}}\ \texttt{f}\ 3 : \alpha_1}
$$

- Über α_2 abstrahieren sinnlos: $\alpha_2 = \alpha_3 \to \alpha_4$
- Über α_3 oder α_4? Falls wir `g y y` durch `g x y` ersetzen?

Typinferenz für `let`

$$C = C_0 \cup C_{\texttt{let}}$$
$$= \{\alpha_1 = \alpha_9\} \cup \{$$
$$\alpha_2 = \alpha_3 \to \alpha_4$$
$$\alpha_5 = \alpha_6 \to \alpha_4$$
$$\alpha_7 = \alpha_8 \to \alpha_5$$
$$\alpha_7 = \beta \to \beta \to \beta$$
$$\alpha_8 = \beta$$
$$\}$$

Beispiel: $\textbf{let}\ \texttt{f} = \lambda \texttt{x}.\ \texttt{g y y}\ \textbf{in}\ \texttt{f}\ 3$

Mit:

$$\Gamma = \texttt{y} : \beta,\ \texttt{g} : \beta \to \beta \to \beta$$
$$\Gamma' = \texttt{y} : \beta,\ \texttt{g} : \beta \to \beta \to \beta,\ \texttt{f} : \forall \alpha_3.\alpha_3 \to \beta$$
$$\sigma_{let} = [\beta \Leftarrow \beta, \alpha_2 \Leftarrow \alpha_3 \to \beta, \ldots]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, \texttt{x} : \alpha_3 \vdash \texttt{g} : \alpha_7 \qquad \Gamma, \texttt{x} : \alpha_3 \vdash \texttt{y} : \alpha_8}{\Gamma, \texttt{x} : \alpha_3 \vdash \texttt{g y} : \alpha_5} \qquad \Gamma, \texttt{x} : \alpha_3 \vdash \texttt{y} : \alpha_6}{\Gamma, \texttt{x} : \alpha_3 \vdash \texttt{g y y} : \alpha_4}}{\Gamma \vdash \lambda \texttt{x}.\ \texttt{g y y} : \alpha_2} \qquad \Gamma' \vdash \texttt{f}\ 3 : \alpha_9}{\Gamma \vdash \textbf{let}\ \texttt{f} = \lambda \texttt{x}.\ \texttt{g y y}\ \textbf{in}\ \texttt{f}\ 3 : \alpha_1}$$

$\Rightarrow$ Unifikator $\sigma_{\texttt{let}}$ für $C_{\texttt{let}}$ aus linkem Teilbaum bestimmen

- Weiter im rechten Teilbaum, mit Annahmen $\Gamma' = \sigma_{\texttt{let}}(\Gamma), \texttt{f} : ta(\sigma_{\texttt{let}}(\alpha_2), \sigma_{\texttt{let}}(\Gamma))$ sowie $\sigma_{\texttt{let}}(C_0)$

Typinferenz für `let`

Beispiel: **let** f = λx. g y y **in** f 3

Mit:

$$\Gamma \quad = \mathtt{y} : \beta,\ \mathtt{g} : \beta \to \beta \to \beta$$
$$\Gamma' \quad = \mathtt{y} : \beta,\ \mathtt{g} : \beta \to \beta \to \beta,\ \mathtt{f} : \forall \alpha_3.\alpha_3 \to \beta$$

$$\sigma \quad = [\alpha_1 \Rightarrow \beta, \alpha_{12} \Rightarrow \text{int}, \ldots]$$

$$C = \{$$
$$\alpha_1 = \alpha_9$$
$$\alpha_{10} = \alpha_{11} \to \alpha_9$$
$$\alpha_{10} = \alpha_{12} \to \beta$$
$$\alpha_{11} = \text{int}$$
$$\}$$

$$\dfrac{\ldots \qquad \dfrac{\dfrac{\Gamma'(\mathtt{f}) = \forall \alpha_3.\alpha_3 \to \beta \qquad \forall \alpha_3.\alpha_3 \to \beta \succeq \alpha_{12} \to \beta}{\Gamma' \vdash \mathtt{f} : \alpha_{10}} \qquad \dfrac{3 \in \mathit{Const}}{\Gamma' \vdash 3 : \alpha_{11}}}{\Gamma' \vdash \mathtt{f}\ 3 : \alpha_9}}{\Gamma \vdash \mathbf{let}\ \mathtt{f} = \lambda \mathtt{x}.\ \mathtt{g}\ \mathtt{y}\ \mathtt{y}\ \mathbf{in}\ \mathtt{f}\ 3 : \alpha_1}$$

- Typschema in Annahme: Constraint mit neuen α_i für alle gebundenen Typvariablen

Zusammenfassung

Polymorphe Funktionen:

- Flexibel einsetzbar, trotzdem bleibt Typsicherheit garantiert

Polymorphe Typinferenz:

- Berechnung des allgemeinsten Typs mittels allgemeinsten Unifikator

$\Rightarrow$ Deklaration des Typs von Funktionen nicht unbedingt notwendig

`let`-Polymorphismus:

- **`let`**-definierte Funktionen können polymorph sein, da

$$\textbf{let } f = t_f \textbf{ in } t \quad \simeq \quad (\lambda \texttt{f}.\ t)\ t_f$$

  und bekannt ist, dass `f` in t an t_f gebunden ist

  $\Rightarrow$ Grad des Polymorphismus kann genau inferiert werden

- Rein λ-gebundenen Variablen `f` in $(\lambda \texttt{f}.\ t)$ können nicht polymorph sein, da Bindung von `f` unbekannt

$\Rightarrow$ Typabstraktion für f_t im **`let`** darf nicht über freie Typvariablen in Typannahme für f_t abstrahieren, da diese durch äußere λ entstehen:

$$\mathit{ta}(\tau, \Gamma) \text{ abstrahiert nur über } \alpha \in \mathit{FV}(\tau) \setminus \mathit{FV}(\Gamma)$$

Fazit

Fazit

Polymorphe Typinferenz nach Milner ist ein mächtiges und elegantes Instrument, das in konventionellen Sprachen kein Gegenstück hat.

Robin Milner