Polymorpher Lambda-Kalkül [Girard 1972, Reynolds 1974]

Typen: $\tau := \alpha \mid \tau \to \tau \mid \forall\alpha.\tau$

Terme: $t := x \mid \lambda x : \tau.t \mid t\ t \mid \Lambda\alpha.t \mid t\ \tau$

$$\Gamma, x : \tau \vdash x : \tau \qquad [\![x]\!]_{\theta,\sigma} = \sigma(x)$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1.t) : \tau_1 \to \tau_2} \qquad [\![\lambda x : \tau_1.t]\!]_{\theta,\sigma}\ a = [\![t]\!]_{\theta,\sigma[x \mapsto a]}$$

$$\frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t\ u) : \tau_2} \qquad [\![t\ u]\!]_{\theta,\sigma} = [\![t]\!]_{\theta,\sigma}\ [\![u]\!]_{\theta,\sigma}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda\alpha.t) : \forall\alpha.\tau} \qquad [\![\Lambda\alpha.t]\!]_{\theta,\sigma}\ S = [\![t]\!]_{\theta[\alpha \mapsto S],\sigma}$$

$$\frac{\Gamma \vdash t : \forall\alpha.\tau}{\Gamma \vdash (t\ \tau') : \tau[\tau'/\alpha]} \qquad [\![t\ \tau']\!]_{\theta,\sigma} = [\![t]\!]_{\theta,\sigma}\ [\![\tau']\!]_{\theta}$$

Das Parametrizitäts-Theorem [Reynolds 1983, Wadler 1989]

Gegeben τ und Environments θ_1, θ_2, ρ mit $\rho(\alpha) \subseteq \theta_1(\alpha) \times \theta_2(\alpha)$, definiere $\Delta_{\tau,\rho} \subseteq [\![\tau]\!]_{\theta_1} \times [\![\tau]\!]_{\theta_2}$ wie folgt:

$$
\begin{array}{lcl}
\Delta_{\alpha,\rho} & = & \rho(\alpha) \\
\Delta_{\tau_1 \to \tau_2,\rho} & = & \{(f_1, f_2) \mid \forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (f_1\ a_1, f_2\ a_2) \in \Delta_{\tau_2,\rho}\} \\
\Delta_{\forall\alpha.\tau,\rho} & = & \{(g_1, g_2) \mid \forall \mathcal{R} \subseteq S_1 \times S_2.\ (g_1\ S_1, g_2\ S_2) \in \Delta_{\tau,\rho[\alpha \mapsto \mathcal{R}]}\}
\end{array}
$$

Dann gilt für jeden geschlossenen Term t geschlossenen Typs τ:

$$([\![t]\!]_{\emptyset,\emptyset}, [\![t]\!]_{\emptyset,\emptyset}) \in \Delta_{\tau,\emptyset}.$$

Beweis-Skizze

Beweise folgende allgemeinere Aussage:

$\Gamma \vdash t : \tau$ impliziert $(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau,\rho}$,
vorausgesetzt $(\sigma_1(x), \sigma_2(x)) \in \Delta_{\tau',\rho}$ für jedes $x : \tau'$ in Γ

per Induktion über die Struktur von Typableitungen.

Der Basis-Fall is einfach. In den anderen Fällen:

$$\frac{\forall (a_1, a_2) \in \Delta_{\tau_1,\rho}.\ (\llbracket t \rrbracket_{\theta_1,\sigma_1[x \mapsto a_1]}, \llbracket t \rrbracket_{\theta_2,\sigma_2[x \mapsto a_2]}) \in \Delta_{\tau_2,\rho}}{(\llbracket \lambda x : \tau_1.t \rrbracket_{\theta_1,\sigma_1}, \llbracket \lambda x : \tau_1.t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1 \to \tau_2,\rho} \qquad (\llbracket u \rrbracket_{\theta_1,\sigma_1}, \llbracket u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_1,\rho}}{(\llbracket t\ u \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ u \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau_2,\rho}}$$

$$\frac{\forall \mathcal{R} \subseteq S_1 \times S_2.\ (\llbracket t \rrbracket_{\theta_1[\alpha \mapsto S_1],\sigma_1}, \llbracket t \rrbracket_{\theta_2[\alpha \mapsto S_2],\sigma_2}) \in \Delta_{\tau,\rho[\alpha \mapsto \mathcal{R}]}}{(\llbracket \Lambda\alpha.t \rrbracket_{\theta_1,\sigma_1}, \llbracket \Lambda\alpha.t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}$$

$$\frac{(\llbracket t \rrbracket_{\theta_1,\sigma_1}, \llbracket t \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\forall\alpha.\tau,\rho}}{(\llbracket t\ \tau' \rrbracket_{\theta_1,\sigma_1}, \llbracket t\ \tau' \rrbracket_{\theta_2,\sigma_2}) \in \Delta_{\tau[\tau'/\alpha],\rho}}$$

Hinzufügen von Datentypen

Typen: $\tau := \cdots \mid \mathsf{Bool} \mid [\tau]$
Terme: $t := \cdots \mid \mathsf{True} \mid \mathsf{False} \mid [\,]_\tau \mid t : t \mid \textbf{case } t \textbf{ of } \{\cdots\}$

$$\Gamma \vdash \mathsf{True} : \mathsf{Bool} \quad , \quad \Gamma \vdash \mathsf{False} : \mathsf{Bool} \quad , \quad \Gamma \vdash [\,]_\tau : [\tau]$$

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : [\tau]}{\Gamma \vdash (t : u) : [\tau]}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash u : \tau \qquad \Gamma \vdash v : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{\mathsf{True} \to u\,; \mathsf{False} \to v\}) : \tau}$$

$$\frac{\Gamma \vdash t : [\tau'] \qquad \Gamma \vdash u : \tau \qquad \Gamma, x_1 : \tau', x_2 : [\tau'] \vdash v : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{[\,] \to u\,; (x_1 : x_2) \to v\}) : \tau}$$

Mit „offensichtlichen“ Erweiterungen der Semantik und mit

$$\begin{aligned} \Delta_{\mathsf{Bool},\rho} &= \{(\mathsf{True}, \mathsf{True}), (\mathsf{False}, \mathsf{False})\} \\ \Delta_{[\tau],\rho} &= \{([x_1, \ldots, x_n], [y_1, \ldots, y_n]) \mid n \geq 0, (x_i, y_i) \in \Delta_{\tau,\rho}\}\,, \end{aligned}$$

ist das Parametrizitäts-Theorem immer noch erfüllt.

Typklassen

Für jedes

$$\texttt{g} :: [\alpha] \to [\alpha]$$

hatten wir

$$\texttt{g}\ (\texttt{map}\ f\ l) \quad = \quad \texttt{map}\ f\ (\texttt{g}\ l)$$

für beliebige f und l.

Was ist mit

$$\texttt{g} :: \mathsf{Eq}\ \alpha \Rightarrow [\alpha] \to [\alpha] \quad ?$$

Obiges freies Theorem schlägt fehl!

Gegenbeispiel: $\texttt{g} = \texttt{nub}$, $f = \texttt{const}\ 1$ und $l = [1, 2]$.

Warum `g (map f l) = map f (g l)`, intuitiv gesehen

- `g` :: $[\alpha] \to [\alpha]$ muss für jede mögliche Instanziierung von α einheitlich arbeiten.
- Die Ausgabeliste kann nur Elemente der Eingabe *l* enthalten.
- Welche, und in welcher Reihenfolge/Vielfachheit, kann lediglich von *l* abhängen.
- Die einzig mögliche Grundlage zur Entscheidung ist die Länge von *l*.
- Die Listen (`map f l`) und *l* haben stets die selbe Länge.
- Also wählt `g` stets „die selben“ Elemente aus (`map f l`) wie es dies aus *l* tut, außer dass im ersten Fall die entsprechenden Abbilder unter *f* ausgegeben werden.
- Also ist (`g (map f l)`) gleich (`map f (g l)`).
- Argumentation gelungen!

Warum g (map f l) = map f (g l), intuitiv gesehen

- g :: Eq $\alpha \Rightarrow [\alpha] \to [\alpha]$ muss für jede mögliche Instanziierung von α einheitlich arbeiten.
- Die Ausgabeliste kann nur Elemente der Eingabe l enthalten.
- Welche, und in welcher Reihenfolge/Vielfachheit, kann lediglich von l abhängen.
- Die einzig mögliche Grundlage zur Entscheidung ist die Länge von l.

↯ Falsch! Auch möglich: Elemente von l auf Gleichheit testen.

- Die Listen (map f l) und l haben stets die selbe Länge.

Aber Gleichheitstests innerhalb beider Listen liefern nicht notwendigerweise immer das selbe Ergebnis!
Sie tun es, wenn f „injektiv“ ist.

- Also wählt g stets „die selben“ Elemente aus (map f l) wie es dies aus l tut, außer dass im ersten Fall die entsprechenden Abbilder unter f ausgegeben werden.
- Also ist (g (map f l)) gleich (map f (g l)).
- Dies liefert ein verfeinertes freies Theorem!

Formaler: Dictionary Translation

Jedes

$$\mathtt{g} :: \mathsf{Eq}\ \alpha \Rightarrow [\alpha] \to [\alpha]$$

kann als ein

$$\mathtt{g}' :: (\alpha \to \alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

angesehen werden, wobei für jeden Instanztyp τ von Eq,

$$\mathtt{g}_\tau \quad = \quad \mathtt{g}'_\tau\ (==)_\tau$$

Das freie Theorem für $\mathtt{g}'$ besagt, dass

$$\mathtt{g}'\ p\ (\mathtt{map}\ f\ l) \quad = \quad \mathtt{map}\ f\ (\mathtt{g}'\ q\ l)\,,$$

vorausgesetzt, dass für alle x und y, $q\ x\ y = p\ (f\ x)\ (f\ y)$.

Dies bedeutet, dass

$$\mathtt{g}\ (\mathtt{map}\ f\ l) \quad = \quad \mathtt{map}\ f\ (\mathtt{g}\ l)\,,$$

vorausgesetzt, dass für alle x und y, $x == y$ gdw. $(f\ x) == (f\ y)$.

Weiteres wichtiges Konstrukt: Allgemeine Rekursion

Wir hatten für jedes

$$\texttt{g} :: (a \to \text{Bool}) \to [a] \to [a]$$

behauptet, dass

$$\texttt{g}\ p\ (\texttt{map}\ h\ l) \quad = \quad \texttt{map}\ h\ (\texttt{g}\ (p \circ h)\ l)$$

für beliebige p, h und l.

Was ist mit

$$\texttt{g} :: (a \to \text{Bool}) \to [a] \to [a]$$
$$\texttt{g}\ p\ l = [\texttt{head}\ (\texttt{g}\ p\ l)] \qquad ?$$

Obiges freies Theorem schlägt fehl!

Gegenbeispiel: $p = \texttt{id}$, $h = \texttt{const}$ True und $l = [\,]$.

Versuch einer Argumentation

- `g` :: $(a \to \text{Bool}) \to [a] \to [a]$ muss für jede mögliche Instanziierung von a einheitlich arbeiten.
- Die Ausgabeliste kann nur Elemente der Eingabe l enthalten.
- Welche, und in welcher Reihenfolge/Vielfachheit, kann lediglich von l und dem Eingabeprädikat p abhängen.
- Die einzig möglichen Grundlagen zur Entscheidung sind die Länge von l und die Ergebnisse von p auf Elementen von l.
- Aber, die Listen (`map` h l) und l haben stets die selbe Länge.
- Und Anwendung von p auf ein Element von (`map` h l) hat stets das selbe Ergebnis wie Anwendung von $(p \circ h)$ auf das entsprechende Element von l.
- Also wählt `g` mit p stets „die selben“ Elemente aus (`map` h l) wie es `g` mit $(p \circ h)$ aus l tut, außer dass im ersten Fall die entsprechenden Abbilder unter h ausgegeben werden.
- Also ist (`g` p (`map` h l)) gleich (`map` h (`g` $(p \circ h)$ l)).
- Genau das wollten wir beweisen!

Versuch einer Argumentation

- `g` :: $(a \to \text{Bool}) \to [a] \to [a]$ muss für jede mögliche Instanziierung von a einheitlich arbeiten.
- Die Ausgabeliste kann nur Elemente der Eingabe l enthalten.

↯ Falsch! Auch möglich: $\bot$.

- Welche, und in welcher Reihenfolge/Vielfachheit, kann lediglich von l und dem Eingabeprädikat p abhängen.
- Die einzig möglichen Grundlagen zur Entscheidung sind die Länge von l und die Ergebnisse von p auf Elementen von l.

↯ Falsch! Weitere mögliche Grundlage: Ergebnis von p auf $\bot$.

- Aber, die Listen (`map` h l) und l haben stets die selbe Länge.
- Und Anwendung von p auf ein Element von (`map` h l) hat stets das selbe Ergebnis wie Anwendung von $(p \circ h)$ auf das entsprechende Element von l.

Und Anwendung von p auf $\bot$ hat das selbe Ergebnis wie Anwendung von $(p \circ h)$ auf $\bot$, vorausgesetzt h ist strikt ($h\ \bot = \bot$).

Versuch einer Argumentation

- Die einzig möglichen Grundlagen zur Entscheidung sind die Länge von l und die Ergebnisse von p auf Elementen von l.

↯ Falsch! Weitere mögliche Grundlage: Ergebnis von p auf $\bot$.

- Aber, die Listen (`map` h l) und l haben stets die selbe Länge.
- Und Anwendung von p auf ein Element von (`map` h l) hat stets das selbe Ergebnis wie Anwendung von $(p \circ h)$ auf das entsprechende Element von l.

Und Anwendung von p auf $\bot$ hat das selbe Ergebnis wie Anwendung von $(p \circ h)$ auf $\bot$, vorausgesetzt h ist strikt ($h\ \bot = \bot$).

- Also wählt `g` mit p stets „die selben“ Elemente aus (`map` h l) wie es `g` mit $(p \circ h)$ aus l tut, außer dass im ersten Fall die entsprechenden Abbilder unter h ausgegeben werden.

Aber sie könnten auch, an gleichen Positionen, $\bot$ ausgeben.

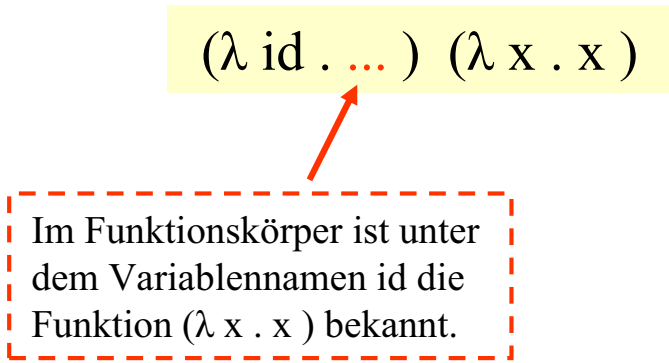
- Also ist (`g` p (`map` h l)) gleich (`map` h (`g` $(p \circ h)$ l)), wenn h strikt ist.

Rekursion im λ-Kalkül (1)

- Die Erweiterung des Kalküls um vordefinierte Funktionen stellte keine Veränderung der Ausdrucksmächtigkeit dar, da diese Funktionen auch simuliert werden konnten.

- Um die Äquivalenz des Kalküls zur Turing-Maschine zu zeigen, fehlt aber noch die Möglichkeit Rekursion auszudrücken.

- Problem: Die Funktionen im λ-Kalkül haben eigtl. keinen Namen.



Der Bezeichnung der Funktion (λ x . x) mit *id* ist aber nur lokal und somit in anderen Funktionen nicht bekannt.

Rekursion im λ-Kalkül (2)

- Wegen der fehlenden globalen Funktionsbezeichner kann z.B. die rekursive Fakultäts-funktion nicht direkt als λ-Ausdruck definiert werden:

$$\text{fak } n := \text{if } (n = 0) \text{ then } 1 \text{ else } n \times \text{fak } (n\text{-}1)$$

- Statt der obigen Definition betrachten wir nun eine leicht modifizierte nicht-rekursive Variante fak‘:

$$\text{fak‘ } f\ n := \text{if } (n = 0) \text{ then } 1 \text{ else } n \times f\ (n\text{-}1)$$

es gilt:

$$\text{fak’ } (\text{fak}) = \text{fak}$$

$$\text{fak’ } f\ n \ldots \equiv \lambda\ f\ n\ .\ \ldots$$

Da fak‘ nicht-rekursiv ist,
gibt es einen dazugehörigen
λ-Ausdruck.

Rekursion im λ-Kalkül (3)

- Die Transformation von fak zu fak‘ war systematisch – zu jeder rekursiven Funktion H kann ein entsprechendes (nicht-rekursives) Funktional H‘ gefunden werden.

- Wir suchen jetzt eine (nicht-rekursive) Funktion **Y**, für die gilt:

$$Y\,(H') = H$$

(H rekursiv, Y und H‘ nicht-rekursiv)

- Da generell H‘(H) = H gilt, ist H ein Fixpunkt der Funktion H‘.

- Weil für die gesuchte Funktion Y(H‘) → H gilt, wird sie auch als Fixpunktkombinator bezeichnet:

$$H = Y\,(H') = H'\,(H) = H'\,(Y\,H')$$

insbes. gilt:

$$Y\,(H') = H'\,(Y\,H')$$

Rekursion im λ-Kalkül (4)

- Die Funktion **Y** wird definiert als:

$$\lambda\, h\, x\, .\, h\, (x\, x)\;\; (\lambda\, x\, .\, h\, (x\, x))$$

- Dass diese Funktion das gesuchte Y ist, wird durch folgenden Zusammenhang belegt:

$$\begin{aligned} \mathbf{Y}\;\; \mathbf{H'} \;\; &= (\lambda\, h\, x\, .\, h\, (x\, x)\; (\lambda\, x\, .\, h\, (x\, x))\;)\;\; H' \\ &\rightarrow (\lambda\, x\, .\, H'\, (x\, x))\; (\lambda\, x\, .\, H'\, (x\, x)) \\ &\rightarrow H'\; ((\lambda\, x\, .\, H'\, (x\, x))\; (\lambda\, x\, .\, H'\, (x\, x))) \\ &= \mathbf{H'\; (\; Y\; H')} \end{aligned}$$

Ein vollständiger Beweis ist das aber nicht – dieser kann z.B. bei Meyer oder Stoy nachgelesen werden.

Rekursion im λ-Kalkül (5)

Zusammenfasung:

- Um eine rekursive Funktion H zu definieren, betrachten wir stattdessen die nicht-rekursive Variante H‘.
- Die Funktionalität von H liefert dann der λ-Ausdruck **Y(H‘)**.

⇒ Zahlen, Arithmetik, Logik, Rekursion im λ-Kalkül ausdrückbar!
- Turing vollständig -

Mit dem typisierten λ-Kalkül wird jetzt eine echte Steigerung der Ausdrucksmächtigkeit vorgestellt….