Freie Theoreme — Was und Wie

(Haskell in Leipzig 3)

Janis Voigtländer

Technische Universität Dresden

18. April 2008

Ein Beispiel:

$$
\begin{aligned}
&\mathit{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
&\mathit{filter}\ p\ [\,] \qquad\quad = [\,] \\
&\mathit{filter}\ p\ (x : xs) = \textbf{if}\ p\ x\ \textbf{then}\ x : \mathit{filter}\ p\ xs\ \textbf{else}\ \mathit{filter}\ p\ xs
\end{aligned}
$$

Ein Beispiel:

$$
\begin{aligned}
&\mathit{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
&\mathit{filter}\ p\ [\,] \qquad\quad = [\,] \\
&\mathit{filter}\ p\ (x : xs) = \textbf{if}\ p\ x\ \textbf{then}\ x : \mathit{filter}\ p\ xs\ \textbf{else}\ \mathit{filter}\ p\ xs
\end{aligned}
$$

Behauptung:

$$\mathit{filter}\ p\ (\mathit{map}\ h\ l) = \mathit{map}\ h\ (\mathit{filter}\ (p \circ h)\ l)$$

Kann per Induktion über l bewiesen werden.

Ein Beispiel:

$$
\begin{aligned}
&\mathit{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
&\mathit{filter}\ p\ [\,] \qquad\quad = [\,] \\
&\mathit{filter}\ p\ (x : xs) = \textbf{if}\ p\ x\ \textbf{then}\ x : \mathit{filter}\ p\ xs\ \textbf{else}\ \mathit{filter}\ p\ xs
\end{aligned}
$$

Behauptung:

$$\mathit{filter}\ p\ (\mathit{map}\ h\ l) = \mathit{map}\ h\ (\mathit{filter}\ (p \circ h)\ l)$$

Kann per Induktion über l bewiesen werden.

Ähnlich gilt:

$$\mathit{takeWhile}\ p\ (\mathit{map}\ h\ l) = \mathit{map}\ h\ (\mathit{takeWhile}\ (p \circ h)\ l)$$

Ein Beispiel:

Behauptung:

$$filter\ p\ (map\ h\ l) = map\ h\ (filter\ (p \circ h)\ l)$$

Kann per Induktion über l bewiesen werden.

Ähnlich gilt:

$$takeWhile\ p\ (map\ h\ l) = map\ h\ (takeWhile\ (p \circ h)\ l)$$

Tatsächlich gilt für jedes $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$f\ p\ (map\ h\ l) = map\ h\ (f\ (p \circ h)\ l)$$

Ein Beispiel:

Behauptung:

$$filter\ p\ (map\ h\ l) = map\ h\ (filter\ (p \circ h)\ l)$$

Kann per Induktion über l bewiesen werden.

Ähnlich gilt:

$$takeWhile\ p\ (map\ h\ l) = map\ h\ (takeWhile\ (p \circ h)\ l)$$

Tatsächlich gilt für jedes $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$f\ p\ (map\ h\ l) = map\ h\ (f\ (p \circ h)\ l)$$

Zauberei? Keine Induktion mehr nötig?

Ein Beispiel:

Behauptung:

$$filter\ p\ (map\ h\ l) = map\ h\ (filter\ (p \circ h)\ l)$$

Kann per Induktion über l bewiesen werden.

Ähnlich gilt:

$$takeWhile\ p\ (map\ h\ l) = map\ h\ (takeWhile\ (p \circ h)\ l)$$

Tatsächlich gilt für jedes $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$,

$$f\ p\ (map\ h\ l) = map\ h\ (f\ (p \circ h)\ l)$$

Zauberei? Keine Induktion mehr nötig?

Freie Theoreme! [Wadler 1989]

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ muss für jede mögliche Instanziierung von α einheitlich arbeiten.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ muss für jede mögliche Instanziierung von α einheitlich arbeiten.
- Die Ausgabeliste kann nur Elemente der Eingabe / enthalten.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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 und dem Eingabeprädikat p abhängen.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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*.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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*.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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 *f* mit *p* stets „die selben“ Elemente aus (*map h l*) wie es *f* mit $(p \circ h)$ aus *l* tut,

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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 *f* mit *p* stets „die selben“ Elemente aus (*map h l*) wie es *f* mit $(p \circ h)$ aus *l* tut, außer dass im ersten Fall die entsprechenden Abbilder unter *h* ausgegeben werden.

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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 *f* mit *p* stets „die selben“ Elemente aus (*map h l*) wie es *f* mit $(p \circ h)$ aus *l* tut, außer dass im ersten Fall die entsprechenden Abbilder unter *h* ausgegeben werden.
- Also ist (*f p* (*map h l*)) gleich (*map h* (*f* $(p \circ h)$ *l*)).

Warum? (intuitiv)

- $f :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$ 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* 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 *f* mit *p* stets „die selben“ Elemente aus (*map h l*) wie es *f* mit $(p \circ h)$ aus *l* tut, außer dass im ersten Fall die entsprechenden Abbilder unter *h* ausgegeben werden.
- Also ist (*f p* (*map h l*)) gleich (*map h* (*f* $(p \circ h)$ *l*)).
- Genau das wollten wir beweisen!

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\}
\end{array}
$$

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\}
\end{array}
$$

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\}
\end{array}
$$

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

- $g \in [\![\forall\alpha.\ \tau]\!]$ müsste eine ganze „Familie“ von Werten sein: für jeden Typ τ', eine Instanz mit Typ $\tau[\tau'/\alpha]$.

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

- $g \in [\![\forall\alpha.\ \tau]\!]$ müsste eine ganze „Familie“ von Werten sein: für jeden Typ τ', eine Instanz mit Typ $\tau[\tau'/\alpha]$.
- $[\![\forall\alpha.\tau]\!] = \{g \mid \forall\tau'.\ g_{\tau'} \in [\![\tau[\tau'/\alpha]]\!]\}$?

Warum? (formaler, aber etwas naiv)

Frage: Welche f haben Typ $\forall\alpha.\ (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$?

Ansatz: Denotationen von Typen als Mengen angeben.

$$
\begin{array}{lcl}
[\![\mathsf{Bool}]\!] & = & \{\mathsf{True}, \mathsf{False}\} \\
[\![\mathsf{Int}]\!] & = & \{\ldots, -2, -1, 0, 1, 2, \ldots\} \\
[\![(\tau_1, \tau_2)]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] \\
[\![[\tau]]\!] & = & \{[x_1, \ldots, x_n] \mid n \geq 0, x_i \in [\![\tau]\!]\} \\
[\![\tau_1 \to \tau_2]\!] & = & \{f : [\![\tau_1]\!] \to [\![\tau_2]\!]\} \\
[\![\forall\alpha.\tau]\!] & = & ?
\end{array}
$$

- $g \in [\![\forall\alpha.\ \tau]\!]$ müsste eine ganze „Familie“ von Werten sein: für jeden Typ τ', eine Instanz mit Typ $\tau[\tau'/\alpha]$.
- $[\![\forall\alpha.\tau]\!] = \{g \mid \forall\tau'.\ g_{\tau'} \in [\![\tau[\tau'/\alpha]]\!]\}$?
- Aber das schließt „ad-hoc-polymorphe“ Funktionen ein!

Unerwünschte Ad-Hoc-Polymorphie am Beispiel

- Mit der vorgeschlagenen Definition,
  $[\![\forall\alpha.\ (\alpha, \alpha) \to \alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]\}$.

Unerwünschte Ad-Hoc-Polymorphie am Beispiel

- Mit der vorgeschlagenen Definition,
  $[\![\forall\alpha.\ (\alpha,\alpha) \to \alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]\}$.
- Aber das erlaubt auch

$$\begin{aligned} g_{\mathsf{Bool}}\ (x, y) &= \mathit{not}\ x \\ g_{\mathsf{Int}}\ \ (x, y) &= y + 1\,, \end{aligned}$$

  was in Haskell beim Typ $\forall\alpha.\ (\alpha,\alpha) \to \alpha$ unmöglich ist.

Unerwünschte Ad-Hoc-Polymorphie am Beispiel

- Mit der vorgeschlagenen Definition,
  $[\![\forall\alpha.\ (\alpha,\alpha)\to\alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!]\times[\![\tau]\!]\to[\![\tau]\!]\}$.
- Aber das erlaubt auch

$$\begin{aligned} g_{\mathsf{Bool}}\ (x,y) &= \mathit{not}\ x \\ g_{\mathsf{Int}}\ \ (x,y) &= y+1\,, \end{aligned}$$

  was in Haskell beim Typ $\forall\alpha.\ (\alpha,\alpha)\to\alpha$ unmöglich ist.
- Um dies zu vermeiden, müssen wir

$$\begin{aligned} g_{\mathsf{Bool}} &: [\![\mathsf{Bool}]\!]\times[\![\mathsf{Bool}]\!]\to[\![\mathsf{Bool}]\!] \quad \text{und} \\ g_{\mathsf{Int}} &: [\![\mathsf{Int}]\!]\times[\![\mathsf{Int}]\!]\to[\![\mathsf{Int}]\!] \end{aligned}$$

  vergleichen und gewährleisten, dass sie sich „identisch verhalten“.

Unerwünschte Ad-Hoc-Polymorphie am Beispiel

- Mit der vorgeschlagenen Definition,
  $[\![\forall\alpha.\ (\alpha,\alpha) \to \alpha]\!] = \{g \mid \forall\tau.\ g_\tau : [\![\tau]\!] \times [\![\tau]\!] \to [\![\tau]\!]\}$.
- Aber das erlaubt auch

$$\begin{aligned} g_{\mathsf{Bool}}\ (x, y) &= \mathit{not}\ x \\ g_{\mathsf{Int}}\ \ (x, y) &= y + 1\,, \end{aligned}$$

  was in Haskell beim Typ $\forall\alpha.\ (\alpha,\alpha) \to \alpha$ unmöglich ist.
- Um dies zu vermeiden, müssen wir

$$\begin{aligned} g_{\mathsf{Bool}} &: [\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!] \to [\![\mathsf{Bool}]\!] \quad \text{und} \\ g_{\mathsf{Int}} &: [\![\mathsf{Int}]\!] \times [\![\mathsf{Int}]\!] \to [\![\mathsf{Int}]\!] \end{aligned}$$

  vergleichen und gewährleisten, dass sie sich „identisch verhalten“.
  Aber wie?

Idee [Reynolds 1983]

Beliebige Relationen benutzen, um Instanzen zu verknüpfen!

Idee [Reynolds 1983]

Beliebige Relationen benutzen, um Instanzen zu verknüpfen!

Im Beispiel ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Man wähle eine Relation $\mathcal{R} \subseteq \mathsf{Bool} \times \mathsf{Int}$.

Idee [Reynolds 1983]

Beliebige Relationen benutzen, um Instanzen zu verknüpfen!

Im Beispiel ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Man wähle eine Relation $\mathcal{R} \subseteq \mathsf{Bool} \times \mathsf{Int}$.
- Man nenne $(x_1, x_2) \in \mathsf{Bool} \times \mathsf{Bool}$ und $(y_1, y_2) \in \mathsf{Int} \times \mathsf{Int}$ verwandt, wenn $(x_1, y_1) \in \mathcal{R}$ und $(x_2, y_2) \in \mathcal{R}$.

Idee [Reynolds 1983]

Beliebige Relationen benutzen, um Instanzen zu verknüpfen!

Im Beispiel ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Man wähle eine Relation $\mathcal{R} \subseteq \mathsf{Bool} \times \mathsf{Int}$.
- Man nenne $(x_1, x_2) \in \mathsf{Bool} \times \mathsf{Bool}$ und $(y_1, y_2) \in \mathsf{Int} \times \mathsf{Int}$ verwandt, wenn $(x_1, y_1) \in \mathcal{R}$ und $(x_2, y_2) \in \mathcal{R}$.
- Man nenne $f_1 : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$ und $f_2 : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ verwandt, wenn verwandte Eingaben zu verwandten Ausgaben führen.

Idee [Reynolds 1983]

Beliebige Relationen benutzen, um Instanzen zu verknüpfen!

Im Beispiel ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Man wähle eine Relation $\mathcal{R} \subseteq \mathsf{Bool} \times \mathsf{Int}$.
- Man nenne $(x_1, x_2) \in \mathsf{Bool} \times \mathsf{Bool}$ und $(y_1, y_2) \in \mathsf{Int} \times \mathsf{Int}$ verwandt, wenn $(x_1, y_1) \in \mathcal{R}$ und $(x_2, y_2) \in \mathcal{R}$.
- Man nenne $f_1 : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$ und $f_2 : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ verwandt, wenn verwandte Eingaben zu verwandten Ausgaben führen.
- Dann sind g_{Bool} und g_{Int} mit

$$\begin{aligned} g_{\mathsf{Bool}}\ (x, y) &= \mathit{not}\ x \quad \text{und} \\ g_{\mathsf{Int}}\ \ (x, y) &= y + 1 \end{aligned}$$

nicht verwandt bei, zum Beispiel, Wahl von $\mathcal{R} = \{(\mathsf{True}, 1)\}$.

Idee [Reynolds 1983]

Beliebige Relationen benutzen, um Instanzen zu verknüpfen!

Im Beispiel ($g :: \forall\alpha.\ (\alpha, \alpha) \to \alpha$):

- Man wähle eine Relation $\mathcal{R} \subseteq \mathsf{Bool} \times \mathsf{Int}$.
- Man nenne $(x_1, x_2) \in \mathsf{Bool} \times \mathsf{Bool}$ und $(y_1, y_2) \in \mathsf{Int} \times \mathsf{Int}$ verwandt, wenn $(x_1, y_1) \in \mathcal{R}$ und $(x_2, y_2) \in \mathcal{R}$.
- Man nenne $f_1 : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$ und $f_2 : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$ verwandt, wenn verwandte Eingaben zu verwandten Ausgaben führen.
- Dann sind g_{Bool} und g_{Int} mit

$$\begin{aligned} g_{\mathsf{Bool}}\ (x, y) &= \mathit{not}\ x \quad \text{und} \\ g_{\mathsf{Int}}\ \ (x, y) &= y + 1 \end{aligned}$$

  nicht verwandt bei, zum Beispiel, Wahl von $\mathcal{R} = \{(\mathsf{True}, 1)\}$.

Reynolds: $g \in [\![\forall\alpha.\tau]\!]$ genau dann wenn für alle τ_1, τ_2 und $\mathcal{R} \subseteq [\![\tau_1]\!] \times [\![\tau_2]\!]$ gilt, dass g_{τ_1} und g_{τ_2} verwandt per „Fortsetzung“ von $\mathcal{R}$ bezüglich τ

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.
2. Ersetze Teiltypen ohne jeglichen Polymorphismus durch Identitätsrelationen.

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.
2. Ersetze Teiltypen ohne jeglichen Polymorphismus durch Identitätsrelationen.
3. Verwende folgende Regeln:

$$(\mathcal{R}, \mathcal{S}) \quad = \quad \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in \mathcal{R}, (x_2, y_2) \in \mathcal{S}\}$$

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.
2. Ersetze Teiltypen ohne jeglichen Polymorphismus durch Identitätsrelationen.
3. Verwende folgende Regeln:

$$
\begin{aligned}
(\mathcal{R}, \mathcal{S}) &= \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in \mathcal{R}, (x_2, y_2) \in \mathcal{S}\} \\
[\mathcal{R}] &= \{([x_1, \ldots, x_n], [y_1, \ldots, y_n]) \mid n \geq 0, (x_i, y_i) \in \mathcal{R}\}
\end{aligned}
$$

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.
2. Ersetze Teiltypen ohne jeglichen Polymorphismus durch Identitätsrelationen.
3. Verwende folgende Regeln:

$$
\begin{aligned}
(\mathcal{R}, \mathcal{S}) &= \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in \mathcal{R}, (x_2, y_2) \in \mathcal{S}\} \\
[\mathcal{R}] &= \{([x_1, \ldots, x_n], [y_1, \ldots, y_n]) \mid n \geq 0, (x_i, y_i) \in \mathcal{R}\} \\
\mathcal{R} \to \mathcal{S} &= \{(f_1, f_2) \mid \forall (a_1, a_2) \in \mathcal{R}.\ (f_1\ a_1, f_2\ a_2) \in \mathcal{S}\}
\end{aligned}
$$

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.
2. Ersetze Teiltypen ohne jeglichen Polymorphismus durch Identitätsrelationen.
3. Verwende folgende Regeln:

$$
\begin{aligned}
(\mathcal{R}, \mathcal{S}) &= \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in \mathcal{R}, (x_2, y_2) \in \mathcal{S}\} \\
[\mathcal{R}] &= \{([x_1, \ldots, x_n], [y_1, \ldots, y_n]) \mid n \geq 0, (x_i, y_i) \in \mathcal{R}\} \\
\mathcal{R} \to \mathcal{S} &= \{(f_1, f_2) \mid \forall (a_1, a_2) \in \mathcal{R}.\ (f_1\ a_1, f_2\ a_2) \in \mathcal{S}\} \\
\forall \mathcal{R}.\ \mathcal{F}(\mathcal{R}) &= \{(u, v) \mid \forall \tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (u_{\tau_1}, v_{\tau_2}) \in \mathcal{F}(\mathcal{R})\}
\end{aligned}
$$

Etwas genauer

Zur Interpretation von Typen als Relationen:

1. Ersetze (Quantifizierung über) Typvariablen durch (Quantifizierung über) Relationsvariablen.
2. Ersetze Teiltypen ohne jeglichen Polymorphismus durch Identitätsrelationen.
3. Verwende folgende Regeln:

$$
\begin{aligned}
(\mathcal{R}, \mathcal{S}) &= \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in \mathcal{R}, (x_2, y_2) \in \mathcal{S}\} \\
[\mathcal{R}] &= \{([x_1, \ldots, x_n], [y_1, \ldots, y_n]) \mid n \geq 0, (x_i, y_i) \in \mathcal{R}\} \\
\mathcal{R} \to \mathcal{S} &= \{(f_1, f_2) \mid \forall (a_1, a_2) \in \mathcal{R}.\ (f_1\ a_1, f_2\ a_2) \in \mathcal{S}\} \\
\forall \mathcal{R}.\ \mathcal{F}(\mathcal{R}) &= \{(u, v) \mid \forall \tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (u_{\tau_1}, v_{\tau_2}) \in \mathcal{F}(\mathcal{R})\}
\end{aligned}
$$

Dann gilt für jedes $f :: \tau$, dass das Paar (f, f) in der Interpretation von τ als Relation enthalten ist.

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$(f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to \mathit{id}_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$(f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$\Leftrightarrow \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$
wegen Definition von $\forall\mathcal{R}.\ \mathcal{F}(\mathcal{R})$

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$(f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ (f_{\tau_1}\ a_1, f_{\tau_2}\ a_2) \in ([\mathcal{R}] \to [\mathcal{R}])$$

wegen Definition von $\mathcal{R} \to \mathcal{S}$

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$(f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ (f_{\tau_1}\ a_1, f_{\tau_2}\ a_2) \in ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ \forall(l_1, l_2) \in [\mathcal{R}].\ (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in [\mathcal{R}]$$

wegen Definition von $\mathcal{R} \to \mathcal{S}$

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$(f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ (f_{\tau_1}\ a_1, f_{\tau_2}\ a_2) \in ([\mathcal{R}] \to [\mathcal{R}])$$

$$\Leftrightarrow \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ \forall(l_1, l_2) \in [\mathcal{R}].\ (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in [\mathcal{R}]$$

$$\Rightarrow \forall(a_1, a_2) \in (h \to id_{\mathsf{Bool}}).\ \forall(l_1, l_2) \in (map\ h).\ (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in (map\ h)$$

durch Spezialisierung zu $\mathcal{R} = h$, wobei dann $[\mathcal{R}] = (map\ h)$

für jede Funktion $h :: \tau_1 \to \tau_2$

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \text{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$
\begin{aligned}
& (f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\text{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\text{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\text{Bool}}).\ (f_{\tau_1}\ a_1, f_{\tau_2}\ a_2) \in ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\text{Bool}}).\ \forall(l_1, l_2) \in [\mathcal{R}]. \\
& \qquad\qquad (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in [\mathcal{R}] \\
\Rightarrow\ & \forall(a_1, a_2) \in (h \to id_{\text{Bool}}).\ \forall(l_1, l_2) \in (map\ h). \\
& \qquad\qquad (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in (map\ h) \\
\Rightarrow\ & \forall(l_1, l_2) \in (map\ h).\ (f_{\tau_1}\ (p \circ h)\ l_1, f_{\tau_2}\ p\ l_2) \in (map\ h) \\
& \text{durch Wahl von } (a_1, a_2) = (p \circ h, p) \in (h \to id_{\text{Bool}})
\end{aligned}
$$

für jede Funktion $h :: \tau_1 \to \tau_2$ und jedes $p :: \tau_2 \to \text{Bool}$.

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \mathsf{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$
\begin{aligned}
& (f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\mathsf{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ (f_{\tau_1}\ a_1, f_{\tau_2}\ a_2) \in ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\mathsf{Bool}}).\ \forall(l_1, l_2) \in [\mathcal{R}]. \\
& \qquad\qquad (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in [\mathcal{R}] \\
\Rightarrow\ & \forall(a_1, a_2) \in (h \to id_{\mathsf{Bool}}).\ \forall(l_1, l_2) \in (map\ h). \\
& \qquad\qquad (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in (map\ h) \\
\Rightarrow\ & \forall(l_1, l_2) \in (map\ h).\ (f_{\tau_1}\ (p \circ h)\ l_1, f_{\tau_2}\ p\ l_2) \in (map\ h) \\
\Leftrightarrow\ & \forall l_1 :: [\tau_1].\ map\ h\ (f_{\tau_1}\ (p \circ h)\ l_1) = f_{\tau_2}\ p\ (map\ h\ l_1)
\end{aligned}
$$

durch einfache Umformung

für jede Funktion $h :: \tau_1 \to \tau_2$ und jedes $p :: \tau_2 \to \mathsf{Bool}$.

Nun formal am Beispiel

Gegeben sei $f :: \forall\alpha.\ (\alpha \to \text{Bool}) \to ([\alpha] \to [\alpha])$.

Dann:

$$
\begin{aligned}
& (f, f) \in \forall\mathcal{R}.\ (\mathcal{R} \to id_{\text{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2.\ (f_{\tau_1}, f_{\tau_2}) \in (\mathcal{R} \to id_{\text{Bool}}) \to ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\text{Bool}}).\ (f_{\tau_1}\ a_1, f_{\tau_2}\ a_2) \in ([\mathcal{R}] \to [\mathcal{R}]) \\
\Leftrightarrow\ & \forall\mathcal{R}.\ \forall(a_1, a_2) \in (\mathcal{R} \to id_{\text{Bool}}).\ \forall(l_1, l_2) \in [\mathcal{R}]. \\
& \qquad\qquad (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in [\mathcal{R}] \\
\Rightarrow\ & \forall(a_1, a_2) \in (h \to id_{\text{Bool}}).\ \forall(l_1, l_2) \in (map\ h). \\
& \qquad\qquad (f_{\tau_1}\ a_1\ l_1, f_{\tau_2}\ a_2\ l_2) \in (map\ h) \\
\Rightarrow\ & \forall(l_1, l_2) \in (map\ h).\ (f_{\tau_1}\ (p \circ h)\ l_1, f_{\tau_2}\ p\ l_2) \in (map\ h) \\
\Leftrightarrow\ & \forall l_1 :: [\tau_1].\ map\ h\ (f_{\tau_1}\ (p \circ h)\ l_1) = f_{\tau_2}\ p\ (map\ h\ l_1)
\end{aligned}
$$

für jede Funktion $h :: \tau_1 \to \tau_2$ und jedes $p :: \tau_2 \to \text{Bool}$.

Das entspricht genau der ursprünglichen Behauptung!

Short Cut Fusion [Gill et al. 1993]

Beispiel:

$$
\begin{aligned}
&\mathit{fromTo}\ n\ m = \mathit{go}\ n \\
&\qquad \textbf{where}\ \mathit{go}\ i = \textbf{if}\ i > m\ \textbf{then}\ [\,] \\
&\qquad\qquad\qquad\qquad\qquad\quad \textbf{else}\ \ i : \mathit{go}\ (\mathit{succ}\ i) \\
&\\
&\mathit{sum}\ [\,] \qquad\quad = 0 \\
&\mathit{sum}\ (x : \mathit{xs}) = x + \mathit{sum}\ \mathit{xs}
\end{aligned}
$$

Short Cut Fusion [Gill et al. 1993]

Beispiel:

$$
\begin{array}{l}
\mathit{fromTo}\ n\ m = \mathit{go}\ n \\
\qquad \textbf{where}\ \mathit{go}\ i = \textbf{if}\ i > m\ \textbf{then}\ [\,] \\
\qquad\qquad\qquad\qquad\qquad\quad \textbf{else}\ \ i : \mathit{go}\ (\mathit{succ}\ i) \\
\\
\mathit{sum}\ [\,] \qquad\quad = 0 \\
\mathit{sum}\ (x : \mathit{xs}) = x + \mathit{sum}\ \mathit{xs}
\end{array}
$$

Problem: Ausdrücke wie

$$\mathit{sum}\ (\mathit{fromTo}\ 1\ 10)$$

führen zur Erzeugung eines Zwischenergebnisses.

Short Cut Fusion [Gill et al. 1993]

Beispiel:

$$
\begin{aligned}
&\mathit{fromTo}\ n\ m = \mathit{go}\ n \\
&\qquad \textbf{where}\ \mathit{go}\ i = \textbf{if}\ i > m\ \textbf{then}\ [\,] \\
&\qquad\qquad\qquad\qquad\qquad\quad \textbf{else}\ \ i : \mathit{go}\ (\mathit{succ}\ i) \\
&\\
&\mathit{sum}\ [\,] \qquad\quad = 0 \\
&\mathit{sum}\ (x : \mathit{xs}) = x + \mathit{sum}\ \mathit{xs}
\end{aligned}
$$

Problem: Ausdrücke wie

$$\mathit{sum}\ (\mathit{fromTo}\ 1\ 10)$$

führen zur Erzeugung eines Zwischenergebnisses.

Lösung:

1. Schreibe *fromTo* mittels *build*.
2. Schreibe *sum* mittels *foldr*.
3. Benutze folgende Regel:

$$
\begin{aligned}
&\{-\#\ \textbf{RULES}\ \text{“foldr/build”} \\
&\quad \forall (g :: \forall \beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta)\ c\ n. \\
&\quad\ \mathit{foldr}\ c\ n\ (\mathit{build}\ g) = g\ c\ n \qquad\qquad \#-\}
\end{aligned}
$$

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{aligned} &\mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ &\mathit{build}\ g = g\ (:)\ [] \end{aligned}$$

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{aligned} &\mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ &\mathit{build}\ g = g\ (:)\ [] \end{aligned}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$.

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{array}{l} \mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ \mathit{build}\ g = g\ (:)\ [] \end{array}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$. Dann:

$$(g, g) \in \forall\mathcal{R}.\ (\mathit{id}_\tau \to \mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R})$$

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{aligned} &\mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ &\mathit{build}\ g = g\ (:)\ [] \end{aligned}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$. Dann:

$$\begin{aligned} & (g, g) \in \forall\mathcal{R}.\ (id_\tau \to \mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R}) \\ \Leftrightarrow\ & \forall\mathcal{R}.\ \forall(c_1, c_2) \in (id_\tau \to \mathcal{R} \to \mathcal{R}).\ \forall(n_1, n_2) \in \mathcal{R}. \\ & \qquad\qquad (g_{\tau_1}\ c_1\ n_1, g_{\tau_2}\ c_2\ n_2) \in \mathcal{R} \end{aligned}$$

wegen Definition von $\forall\mathcal{R}.\ \mathcal{F}(\mathcal{R})$ und $\mathcal{R} \to \mathcal{S}$

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{array}{l} \mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ \mathit{build}\ g = g\ (:)\ [\,] \end{array}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$. Dann:

$$\begin{array}{ll} & (g, g) \in \forall\mathcal{R}.\ (\mathit{id}_\tau \to \mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R}) \\ \Leftrightarrow & \forall\mathcal{R}.\ \forall(c_1, c_2) \in (\mathit{id}_\tau \to \mathcal{R} \to \mathcal{R}).\ \forall(n_1, n_2) \in \mathcal{R}. \\ & \qquad\qquad (g_{\tau_1}\ c_1\ n_1, g_{\tau_2}\ c_2\ n_2) \in \mathcal{R} \\ \Rightarrow & \quad ((:), c_2) \in (\mathit{id}_\tau \to (\mathit{foldr}\ c_2\ n_2) \to (\mathit{foldr}\ c_2\ n_2)) \\ & \wedge\ ([\,], n_2) \in (\mathit{foldr}\ c_2\ n_2) \\ & \Rightarrow (g_{[\tau]}\ (:)\ [\,], g_{\tau'}\ c_2\ n_2) \in (\mathit{foldr}\ c_2\ n_2) \end{array}$$

durch Spezialisierung zu $\mathcal{R} = \mathit{foldr}\ c_2\ n_2$, $c_1 = (:)$ und $n_1 = [\,]$

für jede mögliche Wahl von $c_2 :: \tau \to \tau' \to \tau'$ und $n_2 :: \tau'$.

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{array}{l} \mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ \mathit{build}\ g = g\ (:)\ [\,] \end{array}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$. Dann:

$$\begin{array}{ll} & (g, g) \in \forall\mathcal{R}.\ (id_\tau \to \mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R}) \\ \Leftrightarrow & \forall\mathcal{R}.\ \forall(c_1, c_2) \in (id_\tau \to \mathcal{R} \to \mathcal{R}).\ \forall(n_1, n_2) \in \mathcal{R}. \\ & \qquad\qquad (g_{\tau_1}\ c_1\ n_1, g_{\tau_2}\ c_2\ n_2) \in \mathcal{R} \\ \Rightarrow & \quad ((:), c_2) \in (id_\tau \to (\mathit{foldr}\ c_2\ n_2) \to (\mathit{foldr}\ c_2\ n_2)) \\ & \wedge\ ([\,], n_2) \in (\mathit{foldr}\ c_2\ n_2) \\ & \Rightarrow (g_{[\tau]}\ (:)\ [\,], g_{\tau'}\ c_2\ n_2) \in (\mathit{foldr}\ c_2\ n_2) \\ \Leftrightarrow & \mathit{foldr}\ c_2\ n_2\ (g_{[\tau]}\ (:)\ [\,]) = g_{\tau'}\ c_2\ n_2 \\ & \text{durch Vereinfachung} \end{array}$$

für jede mögliche Wahl von $c_2 :: \tau \to \tau' \to \tau'$ und $n_2 :: \tau'$.

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{aligned}&\mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha]\\&\mathit{build}\ g = g\ (:)\ [\,]\end{aligned}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$. Dann:

$$\begin{aligned}&(g, g) \in \forall\mathcal{R}.\ (id_\tau \to \mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R})\\
\Leftrightarrow\ &\forall\mathcal{R}.\ \forall(c_1, c_2) \in (id_\tau \to \mathcal{R} \to \mathcal{R}).\ \forall(n_1, n_2) \in \mathcal{R}.\\
&\qquad\qquad (g_{\tau_1}\ c_1\ n_1, g_{\tau_2}\ c_2\ n_2) \in \mathcal{R}\\
\Rightarrow\ &\quad ((:), c_2) \in (id_\tau \to (\mathit{foldr}\ c_2\ n_2) \to (\mathit{foldr}\ c_2\ n_2))\\
&\wedge\ ([\,], n_2) \in (\mathit{foldr}\ c_2\ n_2)\\
&\Rightarrow (g_{[\tau]}\ (:)\ [\,], g_{\tau'}\ c_2\ n_2) \in (\mathit{foldr}\ c_2\ n_2)\\
\Leftrightarrow\ &\mathit{foldr}\ c_2\ n_2\ (g_{[\tau]}\ (:)\ [\,]) = g_{\tau'}\ c_2\ n_2\\
\Leftrightarrow\ &\mathit{foldr}\ c_2\ n_2\ (\mathit{build}\ g) = g_{\tau'}\ c_2\ n_2\\
&\text{per Definition von } \mathit{build}\end{aligned}$$

für jede mögliche Wahl von $c_2 :: \tau \to \tau' \to \tau'$ und $n_2 :: \tau'$.

Korrektheitsbeweis

Zur Erinnerung:

$$\begin{array}{l} \mathit{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha] \\ \mathit{build}\ g = g\ (:)\ [\,] \end{array}$$

Gegeben sei $g :: \forall\beta.\ (\tau \to \beta \to \beta) \to \beta \to \beta$. Dann:

$$\begin{array}{ll} & (g, g) \in \forall\mathcal{R}.\ (id_\tau \to \mathcal{R} \to \mathcal{R}) \to (\mathcal{R} \to \mathcal{R}) \\ \Leftrightarrow & \forall\mathcal{R}.\ \forall(c_1, c_2) \in (id_\tau \to \mathcal{R} \to \mathcal{R}).\ \forall(n_1, n_2) \in \mathcal{R}. \\ & \qquad\qquad (g_{\tau_1}\ c_1\ n_1, g_{\tau_2}\ c_2\ n_2) \in \mathcal{R} \\ \Rightarrow & \quad ((:), c_2) \in (id_\tau \to (\mathit{foldr}\ c_2\ n_2) \to (\mathit{foldr}\ c_2\ n_2)) \\ & \wedge\ ([\,], n_2) \in (\mathit{foldr}\ c_2\ n_2) \\ & \Rightarrow (g_{[\tau]}\ (:)\ [\,], g_{\tau'}\ c_2\ n_2) \in (\mathit{foldr}\ c_2\ n_2) \\ \Leftrightarrow & \mathit{foldr}\ c_2\ n_2\ (g_{[\tau]}\ (:)\ [\,]) = g_{\tau'}\ c_2\ n_2 \\ \Leftrightarrow & \mathit{foldr}\ c_2\ n_2\ (\mathit{build}\ g) = g_{\tau'}\ c_2\ n_2 \end{array}$$

für jede mögliche Wahl von $c_2 :: \tau \to \tau' \to \tau'$ und $n_2 :: \tau'$.

Außerdem

- weitere Fusion-Regeln:
  - *destroy*/*unfoldr* [Svenningsson 2002]
  - circular fusion [Fernandes et al. 2007]
  - stream fusion [Coutts et al. 2007]
  - ...

Außerdem

- weitere Fusion-Regeln:
  - *destroy*/*unfoldr* [Svenningsson 2002]
  - circular fusion [Fernandes et al. 2007]
  - stream fusion [Coutts et al. 2007]
  - ...
- Einbeziehung von Typ- und Typkonstruktorklassen

Außerdem

- weitere Fusion-Regeln:
  - *destroy*/*unfoldr* [Svenningsson 2002]
  - circular fusion [Fernandes et al. 2007]
  - stream fusion [Coutts et al. 2007]
  - ...
- Einbeziehung von Typ- und Typkonstruktorklassen
- 0-1-k Prinzipien [Day et al. 1999, V. 2008]
- ...?

Literatur I


D. Coutts, R. Leshchinskiy, and D. Stewart.
Stream fusion: From lists to streams to nothing at all.
In *International Conference on Functional Programming, Proceedings*, pages 315–326. ACM Press, 2007.


N.A. Day, J. Launchbury, and J. Lewis.
Logical abstractions in Haskell.
In *Haskell Workshop, Proceedings*. Technical Report UU-CS-1999-28, Utrecht University, 1999.


J.P. Fernandes, A. Pardo, and J. Saraiva.
A shortcut fusion rule for circular program calculation.
In *Haskell Workshop, Proceedings*, pages 95–106. ACM Press, 2007.

Literatur II


A. Gill, J. Launchbury, and S.L. Peyton Jones.
A short cut to deforestation.
In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 223–232. ACM Press, 1993.


J.C. Reynolds.
Types, abstraction and parametric polymorphism.
In *Information Processing, Proceedings*, pages 513–523. Elsevier Science Publishers B.V., 1983.


J. Svenningsson.
Shortcut fusion for accumulating parameters & zip-like functions.
In *International Conference on Functional Programming, Proceedings*, pages 124–132. ACM Press, 2002.

Literatur III

J. Voigtländer.
Much ado about two: A pearl on parallel prefix computation.
In *Principles of Programming Languages, Proceedings*, pages 29–35. ACM Press, 2008.

P. Wadler.
Theorems for free!
In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.