Typbasierte Programmtransformation

Janis Voigtländer

Technische Universität Dresden

14. Juli 2009

Funktionale Programme in Haskell

Ein Beispiel:

$$
\begin{aligned}
\texttt{map}\ f\ [\,] &= [\,] \\
\texttt{map}\ f\ (a : as) &= (f\ \ a) : (\texttt{map}\ f\ \ as)
\end{aligned}
$$

Funktionale Programme in Haskell

Ein Beispiel:

$$\begin{aligned} \texttt{map}\ f\ [\,] &= [\,] \\ \texttt{map}\ f\ (a : as) &= (f\ \ a) : (\texttt{map}\ f\ \ as) \end{aligned}$$

Einige Aufrufe:

$\texttt{map}\ \texttt{succ}\ [1, 2, 3] \qquad = [2, 3, 4]$

Funktionale Programme in Haskell

Ein Beispiel:

$$\begin{aligned}
\texttt{map}\ f\ [\,] &= [\,] \\
\texttt{map}\ f\ (a : as) &= (f\ \ a) : (\texttt{map}\ f\ \ as)
\end{aligned}$$

Einige Aufrufe:

$$\begin{aligned}
\texttt{map succ}\ [1, 2, 3] &= [2, 3, 4] \\
\texttt{map not}\ \ [\mathsf{True}, \mathsf{False}] &= [\mathsf{False}, \mathsf{True}]
\end{aligned}$$

Funktionale Programme in Haskell

Ein Beispiel:

$$\begin{aligned} \texttt{map}\ f\ [\,] &= [\,] \\ \texttt{map}\ f\ (a : as) &= (f\ \ a) : (\texttt{map}\ f\ \ as) \end{aligned}$$

Einige Aufrufe:

$$\begin{aligned} \texttt{map succ}\ [1,2,3] &= [2,3,4] \\ \texttt{map not}\ \ [\text{True},\text{False}] &= [\text{False},\text{True}] \\ \texttt{map even}\ [1,2,3] &= [\text{False},\text{True},\text{False}] \end{aligned}$$

Funktionale Programme in Haskell

Ein Beispiel:

$$\begin{aligned} \texttt{map}\ f\ [\,] &= [\,] \\ \texttt{map}\ f\ (a : as) &= (f\ \ a) : (\texttt{map}\ f\ \ as) \end{aligned}$$

Einige Aufrufe:

$$\begin{aligned} &\texttt{map succ}\ [1, 2, 3] && = [2, 3, 4] \\ &\texttt{map not}\ \ [\mathsf{True}, \mathsf{False}] && = [\mathsf{False}, \mathsf{True}] \\ &\texttt{map even}\ [1, 2, 3] && = [\mathsf{False}, \mathsf{True}, \mathsf{False}] \\ &\texttt{map not}\ \ [1, 2, 3] && \end{aligned}$$

Funktionale Programme in Haskell

Ein Beispiel:

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ \ a) : (\texttt{map}\ f\ \ as)
\end{aligned}$$

Einige Aufrufe:

$\texttt{map succ}\ [1, 2, 3] = [2, 3, 4]$

$\texttt{map not}\ [\mathsf{True}, \mathsf{False}] = [\mathsf{False}, \mathsf{True}]$

$\texttt{map even}\ [1, 2, 3] = [\mathsf{False}, \mathsf{True}, \mathsf{False}]$

$\texttt{map not}\ [1, 2, 3]$

Funktionale Programme in Haskell

Ein Beispiel:

$$
\begin{aligned}
&\texttt{map} :: (\alpha \rightarrow \beta) \rightarrow [\alpha] \rightarrow [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}
$$

Einige Aufrufe:

$\texttt{map succ}$ $[1, 2, 3]$	$= [2, 3, 4]$		— $\alpha, \beta \mapsto \text{Int}, \text{Int}$
$\texttt{map not}$ $[\text{True}, \text{False}]$	$= [\text{False}, \text{True}]$		— $\alpha, \beta \mapsto \text{Bool}, \text{Bool}$
$\texttt{map even}$ $[1, 2, 3]$	$= [\text{False}, \text{True}, \text{False}]$		— $\alpha, \beta \mapsto \text{Int}, \text{Bool}$
$\texttt{map not}$ $[1, 2, 3]$			

Funktionale Programme in Haskell

Ein Beispiel:

$$\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta]$$
$$\texttt{map}\ f\ [\,] = [\,]$$
$$\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)$$

Einige Aufrufe:

$\texttt{map succ}\ [1,2,3] = [2,3,4]$		— $\alpha, \beta \mapsto \text{Int}, \text{Int}$
$\texttt{map not}\ [\text{True}, \text{False}] = [\text{False}, \text{True}]$		— $\alpha, \beta \mapsto \text{Bool}, \text{Bool}$
$\texttt{map even}\ [1,2,3] = [\text{False}, \text{True}, \text{False}]$		— $\alpha, \beta \mapsto \text{Int}, \text{Bool}$
$\texttt{map not}\ [1,2,3]$	↯ zur Compile-Zeit zurückgewiesen	

Funktionale Programme in Haskell

Ein Beispiel:

$$\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta]$$

Einige Aufrufe:

$\texttt{map succ}\ [1,2,3]$	$= [2,3,4]$	— $\alpha, \beta \mapsto$ Int, Int
$\texttt{map not}\ [\text{True}, \text{False}]$	$= [\text{False}, \text{True}]$	— $\alpha, \beta \mapsto$ Bool, Bool
$\texttt{map even}\ [1,2,3]$	$= [\text{False}, \text{True}, \text{False}]$	— $\alpha, \beta \mapsto$ Int, Bool
$\texttt{map not}\ [1,2,3]$	$\lightning$ zur Compile-Zeit zurückgewiesen	

Komposition von Funktionen

Ein weiteres Beispiel:

$$
\begin{array}{l}
\texttt{filter} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{filter}\ p\ [\,] = [\,] \\
\texttt{filter}\ p\ (a : as)\ \mid\ p\ a = a : (\texttt{filter}\ p\ as) \\
\qquad\qquad\qquad\quad \mid\ \texttt{otherwise} = \texttt{filter}\ p\ as
\end{array}
$$

Komposition von Funktionen

Ein weiteres Beispiel:

$$
\begin{array}{ll}
\texttt{filter} :: (\alpha \rightarrow \text{Bool}) \rightarrow [\alpha] \rightarrow [\alpha] & \\
\texttt{filter}\ p\ [\,] & = [\,] \\
\texttt{filter}\ p\ (a : as) & \mid\ p\ a \qquad\qquad = a : (\texttt{filter}\ p\ as) \\
& \mid\ \texttt{otherwise} = \texttt{filter}\ p\ as
\end{array}
$$

Problem: Ausdrücke wie $\texttt{map}\ f\ (\texttt{filter}\ p\ l)$ erfordern Konstruktion von Zwischenergebnissen.

Komposition von Funktionen

Ein weiteres Beispiel:

$$
\begin{array}{l}
\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{filter}\ p\ [\,] \qquad = [\,] \\
\texttt{filter}\ p\ (a : as) \mid p\ a \qquad = a : (\texttt{filter}\ p\ as) \\
\qquad\qquad\qquad\quad \mid \texttt{otherwise} = \texttt{filter}\ p\ as
\end{array}
$$

Problem: Ausdrücke wie $\texttt{map}\ f\ (\texttt{filter}\ p\ l)^*$ erfordern Konstruktion von Zwischenergebnissen.

---

* $\texttt{sum}\ [f\ x \mid x \leftarrow [1..n], p\ x] \quad \leadsto \quad \texttt{sum}\ (\texttt{map}\ f\ (\texttt{filter}\ p\ (\texttt{enumFromTo}\ 1\ n)))$

Komposition von Funktionen

Ein weiteres Beispiel:

$$
\begin{aligned}
&\texttt{filter} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha] \\
&\texttt{filter}\ p\ [\,] \qquad\qquad = [\,] \\
&\texttt{filter}\ p\ (a : as)\ \mid\ p\ a \qquad\qquad = a : (\texttt{filter}\ p\ as) \\
&\qquad\qquad\qquad\quad\ \mid\ \texttt{otherwise} = \texttt{filter}\ p\ as
\end{aligned}
$$

Problem: Ausdrücke wie $\texttt{map}\ f\ (\texttt{filter}\ p\ l)$ * erfordern Konstruktion von Zwischenergebnissen.

Lösung?: Explizite Regeln

$$
\begin{aligned}
\texttt{map}\ f\ (\texttt{filter}\ p\ l) &\leadsto \cdots \\
\texttt{filter}\ p\ (\texttt{map}\ f\ l) &\leadsto \cdots \\
\texttt{map}\ f_1\ (\texttt{map}\ f_2\ l) &\leadsto \cdots \\
\texttt{filter}\ p_1\ (\texttt{filter}\ p_2\ l) &\leadsto \cdots
\end{aligned}
$$

---

* $\texttt{sum}\ [f\ x \mid x \leftarrow [1..n], p\ x] \leadsto \texttt{sum}\ (\texttt{map}\ f\ (\texttt{filter}\ p\ (\texttt{enumFromTo}\ 1\ n)))$

Komposition von Funktionen

Ein weiteres Beispiel:

$$
\begin{array}{l}
\texttt{filter} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{filter}\ p\ [\,] \qquad = [\,] \\
\texttt{filter}\ p\ (a : as)\ \mid\ p\ a \qquad\qquad = a : (\texttt{filter}\ p\ as) \\
\qquad\qquad\qquad\quad\ \mid\ \texttt{otherwise} = \texttt{filter}\ p\ as
\end{array}
$$

Für jede Wahl von p, f und l gilt:

$$\texttt{filter}\ p\ (\texttt{map}\ f\ \ l) \ \ = \ \ \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l)$$

Beweisbar per Induktion.

Komposition von Funktionen

Ein weiteres Beispiel:

$$\begin{array}{l}
\texttt{filter} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha] \\
\texttt{filter}\ p\ [\,] \qquad = [\,] \\
\texttt{filter}\ p\ (a : as) \mid p\ a \qquad\quad = a : (\texttt{filter}\ p\ as) \\
\qquad\qquad\qquad\quad \mid \texttt{otherwise} = \texttt{filter}\ p\ as
\end{array}$$

Für jede Wahl von p, f und l gilt:

$$\texttt{filter}\ p\ (\texttt{map}\ f\ l) \ = \ \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l)$$

Beweisbar per Induktion.

Oder als „freies Theorem“ [Wadler, FPCA'89].

Komposition von Funktionen

Ein weiteres Beispiel:

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

Für jede Wahl von p, f und l gilt:

$$\texttt{filter}\ p\ (\texttt{map}\ f\ l) \quad = \quad \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l)$$

Beweisbar per Induktion.

Oder als „freies Theorem“ [Wadler, FPCA’89].

Komposition von Funktionen

Ein weiteres Beispiel:

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

Für jede Wahl von p, f und l gilt:

$$\texttt{filter}\ p\ (\texttt{map}\ f\ l) \quad = \quad \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l)$$

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ l) \quad = \quad \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l)$$

Komposition von Funktionen

Ein weiteres Beispiel:

$$\texttt{filter} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

$$\texttt{takeWhile} :: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$$

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

Für jede Wahl von p, f und l gilt:

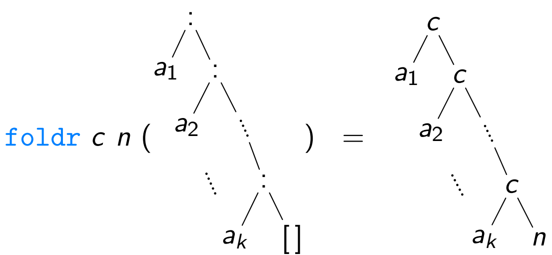
$$\texttt{filter}\ p\ (\texttt{map}\ f\ l) \;=\; \texttt{map}\ f\ (\texttt{filter}\ (p \circ f)\ l)$$

$$\texttt{takeWhile}\ p\ (\texttt{map}\ f\ l) \;=\; \texttt{map}\ f\ (\texttt{takeWhile}\ (p \circ f)\ l)$$

$$\texttt{g}\ p\ (\texttt{map}\ f\ l) \;=\; \texttt{map}\ f\ (\texttt{g}\ (p \circ f)\ l)$$

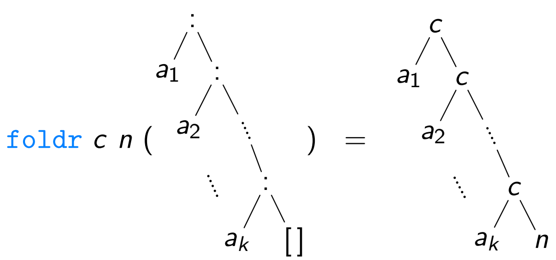
Short Cut Fusion [Gill et al., FPCA'93]

Schreibe Listenkonsumenten mittels `foldr`:



Short Cut Fusion [Gill et al., FPCA'93]

Schreibe Listenkonsumenten mittels `foldr`:

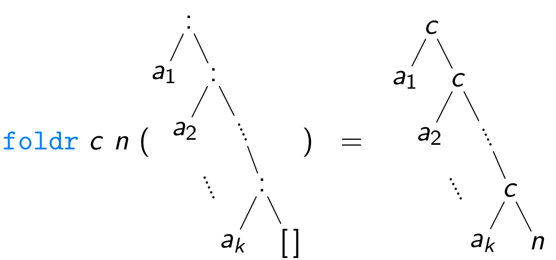


Schreibe Listenerzeuger mittels `build`:

$$\texttt{build} :: (\forall \beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha]$$
$$\texttt{build}\ \mathit{prod} = \mathit{prod}\ (:)\ [\,]$$

Short Cut Fusion [Gill et al., FPCA'93]

Schreibe Listenkonsumenten mittels `foldr`:

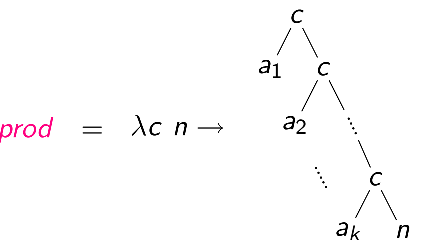


Schreibe Listenerzeuger mittels `build`:

$$\texttt{build} :: (\forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta) \to [\alpha]$$
$$\texttt{build}\ \mathit{prod} = \mathit{prod}\ (:)\ [\,]$$

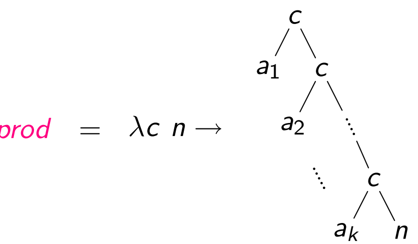
Short Cut Fusion [Gill et al., FPCA'93]

Jedes derart polymorphe *prod* muss einer Funktion der folgenden Form entsprechen, für feste $k \geq 0$ und $a_1, \ldots, a_k$:



Short Cut Fusion [Gill et al., FPCA'93]

Jedes derart polymorphe *prod* muss einer Funktion der folgenden Form entsprechen, für feste $k \geq 0$ und $a_1, \ldots, a_k$:



Zum Beispiel:

$\texttt{filter} :: (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]$

$\texttt{filter}\ p\ as = \texttt{build}\ (\lambda c\ n \to \textbf{let}\ c'\ a\ r \mid p\ a = c\ a\ r$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid \texttt{otherwise} = r$
$\qquad\qquad\qquad\qquad\quad \textbf{in}\ \texttt{foldr}\ c'\ n\ as)$

Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c\ n\ (\texttt{build}\ \mathit{prod}) \quad \rightsquigarrow \quad \mathit{prod}\ c\ n$$

Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ \mathit{prod}) \quad \rightsquigarrow \quad \mathit{prod}\ c'\ n'$$

Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ \mathit{prod}) \quad \rightsquigarrow \quad \mathit{prod}\ c'\ n'$$

Zur Rechtfertigung:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ \mathit{prod})$$

Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ \mathit{prod}) \quad \rightsquigarrow \quad \mathit{prod}\ c'\ n'$$

Zur Rechtfertigung:

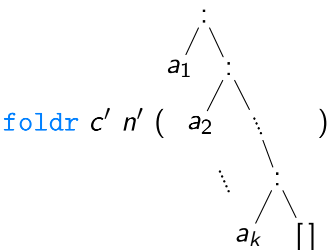
$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ (\lambda c\ n \rightarrow \quad c\ a_1\ (c\ a_2\ (\cdots (c\ a_k\ n)))\ ))$$

Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ \mathit{prod}) \quad \rightsquigarrow \quad \mathit{prod}\ c'\ n'$$

Zur Rechtfertigung:

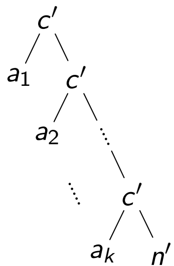


Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ prod) \quad \rightsquigarrow \quad prod\ c'\ n'$$

Zur Rechtfertigung:

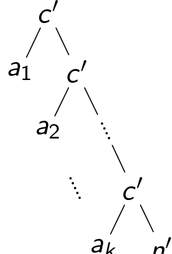


Short Cut Fusion [Gill et al., FPCA'93]

Benutze folgende Regel:

$$\texttt{foldr}\ c'\ n'\ (\texttt{build}\ \mathit{prod}) \quad \leadsto \quad \mathit{prod}\ c'\ n'$$

Zur Rechtfertigung:



Für den Compiler (GHC):

$$\{-\#\ \textbf{RULES}\ \text{“foldr/build”}$$
$$\forall(\mathit{prod} :: \forall\beta.\ (\alpha \to \beta \to \beta) \to \beta \to \beta)\ c\ n.$$
$$\texttt{foldr}\ c\ n\ (\texttt{build}\ \mathit{prod}) = \mathit{prod}\ c\ n \qquad \#-\}$$

Die `destroy`-Funktion [Svenningsson, ICFP'02]

Als Alternative zu `foldr`:

$$\texttt{destroy} :: (\forall \beta.\ (\beta \rightarrow \text{Maybe}\ (\alpha, \beta)) \rightarrow \beta \rightarrow \gamma) \rightarrow [\alpha] \rightarrow \gamma$$

$$\texttt{destroy}\ \mathit{cons}\ \mathit{as} = \mathit{cons}\ \texttt{match}\ \mathit{as}$$

wobei:

$$\textbf{data}\ \text{Maybe}\ \tau = \text{Nothing} \mid \text{Just}\ \tau$$

$$\begin{aligned}
&\texttt{match} :: [\alpha] \rightarrow \text{Maybe}\ (\alpha, [\alpha]) \\
&\texttt{match}\ [\,] \qquad\quad = \text{Nothing} \\
&\texttt{match}\ (a : \mathit{as}) = \text{Just}\ (a, \mathit{as})
\end{aligned}$$

Die destroy-Funktion [Svenningsson, ICFP'02]

Als Alternative zu foldr:

$\texttt{destroy} :: (\forall \beta.\ (\beta \rightarrow \text{Maybe}\ (\alpha, \beta)) \rightarrow \beta \rightarrow \gamma) \rightarrow [\alpha] \rightarrow \gamma$

$\texttt{destroy}\ \mathit{cons}\ \mathit{as} = \mathit{cons}\ \texttt{match}\ \mathit{as}$

wobei:

$\textbf{data}\ \text{Maybe}\ \tau = \text{Nothing} \mid \text{Just}\ \tau$

$\texttt{match} :: [\alpha] \rightarrow \text{Maybe}\ (\alpha, [\alpha])$

$\texttt{match}\ [\,] \qquad = \text{Nothing}$

$\texttt{match}\ (a : \mathit{as}) = \text{Just}\ (a, \mathit{as})$

Zum Beispiel:

$\texttt{zip} :: [\alpha] \rightarrow [\beta] \rightarrow [(\alpha, \beta)]$

$\texttt{zip}\ \mathit{as}\ \mathit{bs} = \texttt{destroy}\ (\lambda p\ x \rightarrow \texttt{destroy}\ (\lambda q\ y \rightarrow \texttt{zipD}\ p\ q\ x\ y)\ \mathit{bs})\ \mathit{as}$

$\quad \textbf{where}\ \texttt{zipD} = \lambda p\ q\ x\ y \rightarrow$

$\qquad \textbf{case}\ (p\ x, q\ y)\ \textbf{of}$

$\qquad\quad (\text{Nothing}\ \ , \text{Nothing}\ \ ) \rightarrow [\,]$

$\qquad\quad (\text{Just}\ (a, x'), \text{Just}\ (b, y')) \rightarrow (a, b) : (\texttt{zipD}\ p\ q\ x'\ y')$

Eine `destroy`/`build`-Regel [V., PEPM'08]

Laut Definitionen ist

$$\texttt{destroy}\ \mathit{cons}\ (\texttt{build}\ \mathit{prod})$$

...

Eine destroy/build-Regel [V., PEPM'08]

Laut Definitionen ist

$$\texttt{destroy}\ \mathit{cons}\ (\texttt{build}\ \mathit{prod})$$

das Gleiche wie

$$\mathit{cons}\ \texttt{match}\ (\mathit{prod}\ (:)\ [\,])\,,$$

wobei:

$$\begin{aligned} \texttt{match}\ [\,] &= \text{Nothing} \\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as) \end{aligned}$$

Eine destroy/build-Regel [V., PEPM'08]

Laut Definitionen ist

$$\texttt{destroy}\ \mathit{cons}\ (\texttt{build}\ \mathit{prod})$$

das Gleiche wie

$$\mathit{cons}\ \texttt{match}\ (\mathit{prod}\ (:)\ [\,])\,,$$

wobei:

$$\begin{aligned} \texttt{match}\ [\,] &= \text{Nothing} \\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as) \end{aligned}$$

Warum dann nicht einfach

$$\texttt{destroy}\ \mathit{cons}\ (\texttt{build}\ \mathit{prod})$$
$$\leadsto$$
$$\mathit{cons}\ \texttt{id}\ (\mathit{prod}\ (\lambda a\ as \rightarrow \text{Just}\ (a, as))\ \text{Nothing})\quad ?$$

Eine destroy/build-Regel [V., PEPM'08]

Laut Definitionen ist

$$\texttt{destroy}\ \mathit{cons}\ (\texttt{build}\ \mathit{prod})$$

das Gleiche wie

$$\mathit{cons}\ \texttt{match}\ (\mathit{prod}\ (:)\ [\,])\,,$$

wobei:

$$\begin{aligned}\texttt{match}\ [\,] &= \text{Nothing}\\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as)\end{aligned}$$

Warum dann nicht einfach

$$\texttt{destroy}\ \mathit{cons}\ (\texttt{build}\ \mathit{prod})$$
$$\rightsquigarrow$$
$$\mathit{cons}\ \texttt{id}\ (\mathit{prod}\ (\lambda a\ as \rightarrow \text{Just}\ (a, as))\ \text{Nothing})\quad ?$$

Erhält diese Regel die Semantik?

Beweis der Korrektheit

Alles, was wir über *cons* und *prod* wissen, sind ihre Typen:

$$\mathit{cons} :: \forall \beta.\ (\beta \to \mathsf{Maybe}\ (\mathsf{T}_1, \beta)) \to \beta \to \mathsf{T}_2$$

und

$$\mathit{prod} :: \forall \beta.\ (\mathsf{T}_1 \to \beta \to \beta) \to \beta \to \beta$$

Beweis der Korrektheit

Alles, was wir über *cons* und *prod* wissen, sind ihre Typen:

$$\textit{cons} :: \forall\beta.\ (\beta \to \mathsf{Maybe}\ (\mathsf{T}_1, \beta)) \to \beta \to \mathsf{T}_2$$

und

$$\textit{prod} :: \forall\beta.\ (\mathsf{T}_1 \to \beta \to \beta) \to \beta \to \beta$$

Aber dies könnte reichen, Dank freier Theoreme!

Im Folgenden, eine Beweisskizze.

Wo beginnen?

Das freie Theorem für

$$cons :: \forall\beta.\ (\beta \to \text{Maybe}\ (\text{T}_1, \beta)) \to \beta \to \text{T}_2$$

ist:

$$\begin{aligned}
&\forall \tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2, \mathcal{R}\ \text{strict, continuous, and bottom-reflecting.} \\
&\quad \forall p :: \tau_1 \to \text{Maybe}\ (\text{T}_1, \tau_1), q :: \tau_2 \to \text{Maybe}\ (\text{T}_1, \tau_2). \\
&\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\
&\qquad \wedge\ (\forall (x, y) \in \mathcal{R}.\ (p\ x, q\ y) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, \mathcal{R}))) \\
&\qquad \Rightarrow \forall (z, v) \in \mathcal{R}.\ cons\ p\ z = cons\ q\ v
\end{aligned}$$

Wo beginnen?

Das freie Theorem für

$$cons :: \forall\beta.\ (\beta \to \mathsf{Maybe}\ (\mathsf{T}_1, \beta)) \to \beta \to \mathsf{T}_2$$

ist:

$$\begin{array}{l}
\forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2, \mathcal{R} \text{ strict, continuous, and bottom-reflecting.} \\
\quad \forall p :: \tau_1 \to \mathsf{Maybe}\ (\mathsf{T}_1, \tau_1), q :: \tau_2 \to \mathsf{Maybe}\ (\mathsf{T}_1, \tau_2). \\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\
\qquad \wedge\ (\forall(x, y) \in \mathcal{R}.\ (p\ x, q\ y) \in \mathrm{lift}_{\mathsf{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, \mathcal{R}))) \\
\qquad \Rightarrow \forall(z, v) \in \mathcal{R}.\ cons\ p\ z = cons\ q\ v
\end{array}$$

Zur Erinnerung, wir wollen beweisen:

$$\begin{array}{c}
cons\ \texttt{match}\ (prod\ (:)\ [\,]) \\
= \\
cons\ \texttt{id}\ (prod\ (\lambda a\ as \to \mathsf{Just}\ (a, as))\ \mathsf{Nothing})
\end{array}$$

Wo beginnen?

Das freie Theorem für

$$cons :: \forall\beta.\ (\beta \to \text{Maybe}\ (\text{T}_1, \beta)) \to \beta \to \text{T}_2$$

ist: [Johann & V., POPL'04]

$$\begin{array}{l}
\forall\tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2, \mathcal{R}\ \text{strict, continuous, and bottom-reflecting}. \\
\quad \forall p :: \tau_1 \to \text{Maybe}\ (\text{T}_1, \tau_1), q :: \tau_2 \to \text{Maybe}\ (\text{T}_1, \tau_2). \\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\
\qquad \wedge\ (\forall (x, y) \in \mathcal{R}.\ (p\ x, q\ y) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, \mathcal{R}))) \\
\qquad \Rightarrow \forall (z, v) \in \mathcal{R}.\ cons\ p\ z = cons\ q\ v
\end{array}$$

Zur Erinnerung, wir wollen beweisen:

$$cons\ \texttt{match}\ (prod\ (:)\ [\,]) \\ = \\ cons\ \texttt{id}\ (prod\ (\lambda a\ as \to \text{Just}\ (a, as))\ \text{Nothing})$$

Wo beginnen?

Das freie Theorem für

$$cons :: \forall\beta.\ (\beta \rightarrow \mathsf{Maybe}\ (\mathsf{T}_1, \beta)) \rightarrow \beta \rightarrow \mathsf{T}_2$$

ist: [Johann & V., POPL'04]

$$\begin{array}{l}
\forall \tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2, \mathcal{R}\ \text{strict, continuous, and bottom-reflecting}.\\
\quad \forall p :: \tau_1 \rightarrow \mathsf{Maybe}\ (\mathsf{T}_1, \tau_1), q :: \tau_2 \rightarrow \mathsf{Maybe}\ (\mathsf{T}_1, \tau_2).\\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
\qquad \wedge\ (\forall (x, y) \in \mathcal{R}.\ (p\ x, q\ y) \in \mathsf{lift}_{\mathsf{Maybe}}(\mathsf{lift}_{(,)}(\mathsf{id}, \mathcal{R})))\\
\qquad \Rightarrow \forall (z, v) \in \mathcal{R}.\ cons\ p\ z = cons\ q\ v
\end{array}$$

Zur Erinnerung, wir wollen beweisen:

$$\begin{array}{c}
cons\ \texttt{match}\ (prod\ (:)\ [\,])\\
=\\
cons\ \texttt{id}\ (prod\ (\lambda a\ as \rightarrow \mathsf{Just}\ (a, as))\ \mathsf{Nothing})
\end{array}$$

Wo beginnen?

Das freie Theorem für

$$\mathit{cons} :: \forall\beta.\ (\beta \rightarrow \mathsf{Maybe}\ (\mathsf{T}_1, \beta)) \rightarrow \beta \rightarrow \mathsf{T}_2\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{aligned}
&\forall \tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total}.\\
&\quad \forall p :: \tau_1 \rightarrow \mathsf{Maybe}\ (\mathsf{T}_1, \tau_1), q :: \tau_2 \rightarrow \mathsf{Maybe}\ (\mathsf{T}_1, \tau_2).\\
&\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
&\qquad \wedge\ (\forall x :: \tau_1.\ (p\ x, q\ (f\ x)) \in \mathrm{lift}_{\mathsf{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f)))\\
&\qquad \Rightarrow \forall y :: \tau_1.\ \mathit{cons}\ p\ y = \mathit{cons}\ q\ (f\ y)
\end{aligned}$$

Zur Erinnerung, wir wollen beweisen:

$$\mathit{cons}\ \texttt{match}\ (\mathit{prod}\ (:)\ [\,])$$
$$=$$
$$\mathit{cons}\ \texttt{id}\ (\mathit{prod}\ (\lambda a\ \mathit{as} \rightarrow \mathsf{Just}\ (a, \mathit{as}))\ \mathsf{Nothing})$$

Wo beginnen?

Das freie Theorem für

$$\mathit{cons} :: \forall\beta.\ (\beta \to \text{Maybe}\ (\text{T}_1, \beta)) \to \beta \to \text{T}_2\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l}
\forall\tau_1, \tau_2, f :: \tau_1 \to \tau_2, f \text{ strict and total.} \\
\quad \forall p :: \tau_1 \to \text{Maybe}\ (\text{T}_1, \tau_1), q :: \tau_2 \to \text{Maybe}\ (\text{T}_1, \tau_2). \\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\
\qquad \wedge\ (\forall x :: \tau_1.\ (p\ x, q\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f))) \\
\qquad \Rightarrow \forall y :: \tau_1.\ \mathit{cons}\ p\ y = \mathit{cons}\ q\ (f\ y)
\end{array}$$

Zur Erinnerung, wir wollen beweisen:

$$\begin{array}{c}
\mathit{cons}\ \texttt{match}\ (\mathit{prod}\ (:)\ [\,]) \\
= \\
\mathit{cons}\ \texttt{id}\ (\mathit{prod}\ (\lambda a\ \mathit{as} \to \text{Just}\ (a, \mathit{as}))\ \text{Nothing})
\end{array}$$

Wo beginnen?

Das freie Theorem für

$$cons :: \forall \beta.\ (\beta \rightarrow \text{Maybe}\ (\mathsf{T}_1, \beta)) \rightarrow \beta \rightarrow \mathsf{T}_2\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l}
\forall \tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total.} \\
\quad \forall p :: \tau_1 \rightarrow \text{Maybe}\ (\mathsf{T}_1, \tau_1), q :: \tau_2 \rightarrow \text{Maybe}\ (\mathsf{T}_1, \tau_2). \\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\
\qquad \wedge\ (\forall x :: \tau_1.\ (p\ x, q\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f))) \\
\qquad \Rightarrow \forall y :: \tau_1.\ cons\ p\ y = cons\ q\ (f\ y)
\end{array}$$

Zur Erinnerung, wir wollen beweisen:

$$cons\ \texttt{match}\ (prod\ (:)\ [\,])$$
$$=$$
$$cons\ \texttt{id}\ (prod\ (\lambda a\ as \rightarrow \text{Just}\ (a, as))\ \text{Nothing})$$

Wo beginnen?

Das freie Theorem für

$$cons :: \forall\beta.\ (\beta \rightarrow \text{Maybe}\ (\text{T}_1, \beta)) \rightarrow \beta \rightarrow \text{T}_2\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l} \forall\tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total.} \\ \quad \forall p :: \tau_1 \rightarrow \text{Maybe}\ (\text{T}_1, \tau_1), q :: \tau_2 \rightarrow \text{Maybe}\ (\text{T}_1, \tau_2). \\ \qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\ \qquad \wedge\ (\forall x :: \tau_1.\ (p\ x, q\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f))) \\ \qquad \Rightarrow \forall y :: \tau_1.\ cons\ p\ y = cons\ q\ (f\ y) \end{array}$$

Zur Erinnerung, wir wollen beweisen:

$$\begin{array}{c} cons\ \texttt{match}\ (prod\ (:)\ [\,]) \\ = \\ cons\ \texttt{id}\ (prod\ (\lambda a\ as \rightarrow \text{Just}\ (a, as))\ \text{Nothing}) \end{array}$$

Wo beginnen?

Das freie Theorem für

$$cons :: \forall\beta.\ (\beta \to \text{Maybe}\ (\mathsf{T}_1, \beta)) \to \beta \to \mathsf{T}_2\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{aligned}
&\forall\tau_1, \tau_2, f :: \tau_1 \to \tau_2, f \text{ strict and total.}\\
&\quad \forall p :: \tau_1 \to \text{Maybe}\ (\mathsf{T}_1, \tau_1), q :: \tau_2 \to \text{Maybe}\ (\mathsf{T}_1, \tau_2).\\
&\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
&\qquad \wedge\ (\forall x :: \tau_1.\ (p\ x, q\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)))\\
&\qquad \Rightarrow \forall y :: \tau_1.\ cons\ p\ y = cons\ q\ (f\ y)
\end{aligned}$$

Zur Erinnerung, wir wollen beweisen:

$$cons\ \mathtt{match}\ (prod\ (:)\ [\,])$$
$$=$$
$$cons\ \mathtt{id}\ (prod\ (\lambda a\ as \to \text{Just}\ (a, as))\ \text{Nothing})$$

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathsf{lift}_{\mathsf{Maybe}}(\mathsf{lift}_{(,)}(\mathsf{id}, f))$
3. $f\ (\mathit{prod}\ (:)\ [\,]) = \mathit{prod}\ (\lambda a\ \mathit{as} \rightarrow \mathsf{Just}\ (a, \mathit{as}))\ \mathsf{Nothing}$

(Hinweis: $\texttt{match} \neq \bot \Leftrightarrow \texttt{id} \neq \bot$ ist trivial erfüllt.)

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $f\ (prod\ (:)\ [\,]) = prod\ (\lambda a\ as \rightarrow \mathrm{Just}\ (a, as))\ \mathrm{Nothing}$

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathsf{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $f\ (prod\ (:)\ [\,]) = prod\ (\lambda a\ as \rightarrow \mathrm{Just}\ (a, as))\ \mathrm{Nothing}$

Das freie Theorem für

$$prod :: \forall \beta.\ (\mathsf{T}_1 \rightarrow \beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta$$

ist:

$$\begin{array}{l}
\forall \tau_1, \tau_2, \mathcal{R} \subseteq \tau_1 \times \tau_2, \mathcal{R} \text{ strict, continuous, and bottom-reflecting.} \\
\quad \forall p :: \mathsf{T}_1 \rightarrow \tau_1 \rightarrow \tau_1, q :: \mathsf{T}_1 \rightarrow \tau_2 \rightarrow \tau_2. \\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot) \\
\qquad \wedge\ (\forall x :: \mathsf{T}_1.\ (p\ x \neq \bot \Leftrightarrow q\ x \neq \bot) \\
\qquad\qquad\qquad\quad \wedge\ \forall (y, z) \in \mathcal{R}.\ (p\ x\ y, q\ x\ z) \in \mathcal{R}) \\
\qquad \Rightarrow \forall (v, w) \in \mathcal{R}.\ (prod\ p\ v, prod\ q\ w) \in \mathcal{R}
\end{array}$$

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $f\ (prod\ (:)\ [\,]) = prod\ (\lambda a\ as \rightarrow \mathrm{Just}\ (a, as))\ \mathrm{Nothing}$

Das freie Theorem für

$$prod :: \forall \beta.\ (\mathsf{T}_1 \rightarrow \beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l}
\forall \tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total}.\\
\quad \forall p :: \mathsf{T}_1 \rightarrow \tau_1 \rightarrow \tau_1, q :: \mathsf{T}_1 \rightarrow \tau_2 \rightarrow \tau_2.\\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
\qquad \wedge\ (\forall x :: \mathsf{T}_1.\ (p\ x \neq \bot \Leftrightarrow q\ x \neq \bot)\\
\qquad\qquad\qquad\qquad \wedge\ \forall y :: \tau_1.\ f\ (p\ x\ y) = q\ x\ (f\ y))\\
\qquad \Rightarrow \forall z :: \tau_1.\ f\ (prod\ p\ z) = prod\ q\ (f\ z)
\end{array}$$

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $f\ (\mathit{prod}\ (:)\ [\,]) = \mathit{prod}\ (\lambda a\ as \rightarrow \mathrm{Just}\ (a, as))\ \mathrm{Nothing}$

Das freie Theorem für

$$\mathit{prod} :: \forall \beta.\ (\mathsf{T}_1 \rightarrow \beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l}
\forall \tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total}.\\
\quad \forall p :: \mathsf{T}_1 \rightarrow \tau_1 \rightarrow \tau_1, q :: \mathsf{T}_1 \rightarrow \tau_2 \rightarrow \tau_2.\\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
\qquad \wedge\ (\forall x :: \mathsf{T}_1.\ (p\ x \neq \bot \Leftrightarrow q\ x \neq \bot)\\
\qquad\qquad\qquad \wedge\ \forall y :: \tau_1.\ f\ (p\ x\ y) = q\ x\ (f\ y))\\
\qquad \Rightarrow \forall z :: \tau_1.\ f\ (\mathit{prod}\ p\ z) = \mathit{prod}\ q\ (f\ z)
\end{array}$$

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $f\ (\mathit{prod}\ (:)\ [\,]) = \mathit{prod}\ (\lambda a\ \mathit{as} \rightarrow \mathrm{Just}\ (a, \mathit{as}))\ \mathrm{Nothing}$

Das freie Theorem für

$$\mathit{prod} :: \forall \beta.\ (\mathsf{T}_1 \rightarrow \beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l}
\forall \tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total}.\\
\quad \forall p :: \mathsf{T}_1 \rightarrow \tau_1 \rightarrow \tau_1, q :: \mathsf{T}_1 \rightarrow \tau_2 \rightarrow \tau_2.\\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
\qquad \wedge\ (\forall x :: \mathsf{T}_1.\ (p\ x \neq \bot \Leftrightarrow q\ x \neq \bot)\\
\qquad\qquad\qquad\qquad \wedge\ \forall y :: \tau_1.\ f\ (p\ x\ y) = q\ x\ (f\ y))\\
\qquad \Rightarrow \forall z :: \tau_1.\ f\ (\mathit{prod}\ p\ z) = \mathit{prod}\ q\ (f\ z)
\end{array}$$

Wie weiter?

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\mathtt{match}\ x, \mathtt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $f\ (\mathit{prod}\ (:)\ [\,]) = \mathit{prod}\ (\lambda a\ \mathit{as} \rightarrow \mathrm{Just}\ (a, \mathit{as}))\ \mathrm{Nothing}$

Das freie Theorem für

$$\mathit{prod} :: \forall \beta.\ (\mathsf{T}_1 \rightarrow \beta \rightarrow \beta) \rightarrow \beta \rightarrow \beta\,,$$

spezialisiert auf die Ebene von Funktionen, ist:

$$\begin{array}{l}
\forall \tau_1, \tau_2, f :: \tau_1 \rightarrow \tau_2, f \text{ strict and total}.\\
\quad \forall p :: \mathsf{T}_1 \rightarrow \tau_1 \rightarrow \tau_1, q :: \mathsf{T}_1 \rightarrow \tau_2 \rightarrow \tau_2.\\
\qquad\quad (p \neq \bot \Leftrightarrow q \neq \bot)\\
\qquad \wedge\ (\forall x :: \mathsf{T}_1.\ (p\ x \neq \bot \Leftrightarrow q\ x \neq \bot)\\
\qquad\qquad\qquad\quad \wedge\ \forall y :: \tau_1.\ f\ (p\ x\ y) = q\ x\ (f\ y))\\
\qquad \Rightarrow \forall z :: \tau_1.\ f\ (\mathit{prod}\ p\ z) = \mathit{prod}\ q\ (f\ z)
\end{array}$$

Fast geschafft

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\mathtt{match}\ x, \mathtt{id}\ (f\ x)) \in \mathsf{lift}_{\mathsf{Maybe}}(\mathsf{lift}_{(,)}(\mathsf{id}, f))$
3. $\forall x :: \mathsf{T}_1, y :: [\mathsf{T}_1].\ f\ ((:)\ x\ y) = (\lambda a\ as \rightarrow \mathsf{Just}\ (a, as))\ x\ (f\ y)$
4. $f\ [\,] = \mathsf{Nothing}$

(Hinweis: die „$\neq \bot$“-Bedingungen sind wieder trivial erfüllt.)

Fast geschafft

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathsf{lift}_{\mathsf{Maybe}}(\mathsf{lift}_{(,)}(\mathsf{id}, f))$
3. $\forall x :: \mathsf{T}_1, y :: [\mathsf{T}_1].\ f\ ((:)\ x\ y) = (\lambda a\ as \rightarrow \mathsf{Just}\ (a, as))\ x\ (f\ y)$
4. $f\ [\,] = \mathsf{Nothing}$

Fast geschafft

Alles, was wir brauchen, ist eine Funktion f so dass:

1. f ist strict und total
2. $\forall x :: [\mathrm{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$
3. $\forall x :: \mathrm{T}_1, y :: [\mathrm{T}_1].\ f\ ((:)\ x\ y) = (\lambda a\ as \to \mathrm{Just}\ (a, as))\ x\ (f\ y)$
4. $f\ [\,] = \mathrm{Nothing}$

Die letzten beiden Bedingungen lassen keine andere Wahl als:

$$\begin{aligned} f\ [\,] &= \mathrm{Nothing} \\ f\ (x : y) &= \mathrm{Just}\ (x, f\ y) \end{aligned}$$

Fast geschafft

Alles, was wir brauchen, ist:

1. f ist strict und total
2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$

Die letzten beiden Bedingungen lassen keine andere Wahl als:

$$\begin{aligned} f\ [\,] &= \mathrm{Nothing} \\ f\ (x : y) &= \mathrm{Just}\ (x, f\ y) \end{aligned}$$

Fast geschafft

Alles, was wir brauchen, ist:

1. f ist strict und total
2. $\forall x :: [\mathrm{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f))$

$$
\begin{aligned}
f\ [\,] &= \mathrm{Nothing} \\
f\ (x : y) &= \mathrm{Just}\ (x, f\ y)
\end{aligned}
$$

Dieses f ist strict und total!

Fast geschafft

2. $\forall x :: [\mathsf{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathsf{lift}_{\mathsf{Maybe}}(\mathsf{lift}_{(,)}(\mathsf{id}, f))$?

$$
\begin{aligned}
f\ [\,] &= \mathsf{Nothing} \\
f\ (x : y) &= \mathsf{Just}\ (x, f\ y)
\end{aligned}
$$

Schließlich . . .

Wir haben:

$$\mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f)) = \{(\bot, \bot), (\mathrm{Nothing}, \mathrm{Nothing})\} \cup \{(\mathrm{Just}\ x_1, \mathrm{Just}\ y_1) \mid (x_1, y_1) \in \mathrm{lift}_{(,)}(\mathrm{id}, f)\}$$

$$\mathrm{lift}_{(,)}(\mathrm{id}, f) = \{(\bot, \bot)\} \ \cup \ \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Schließlich ...

Wir haben:

$$\text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)) = \{(\bot,\bot),(\text{Nothing},\text{Nothing})\} \cup \{(\text{Just } x_1, \text{Just } y_1) \mid (x_1, y_1) \in \text{lift}_{(,)}(\text{id}, f)\}$$

$$\text{lift}_{(,)}(\text{id}, f) = \{(\bot,\bot)\} \cup \{((x_1,x_2),(y_1,y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [T_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f))\,,$$

Schließlich ...

Wir haben:

$$\text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)) = \{(\bot,\bot), (\text{Nothing}, \text{Nothing})\} \cup \{(\text{Just } x_1, \text{Just } y_1) \mid (x_1, y_1) \in \text{lift}_{(,)}(\text{id}, f)\}$$

$$\text{lift}_{(,)}(\text{id}, f) = \{(\bot,\bot)\} \cup \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [\text{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)),$$

untersuchen wir alle Fälle für Eingaben von `match` und f, unter Verwendung der Definitionen:

$$\begin{aligned} \texttt{match}\ [\,] &= \text{Nothing} & \qquad f\ [\,] &= \text{Nothing} \\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as) & \qquad f\ (x : y) &= \text{Just}\ (x, f\ y) \end{aligned}$$

Schließlich ...

Wir haben:

$$\mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f)) = \{(\bot, \bot), (\mathrm{Nothing}, \mathrm{Nothing})\} \cup \{(\mathrm{Just}\ x_1, \mathrm{Just}\ y_1) \mid (x_1, y_1) \in \mathrm{lift}_{(,)}(\mathrm{id}, f)\}$$

$$\mathrm{lift}_{(,)}(\mathrm{id}, f) = \{(\bot, \bot)\} \ \cup \ \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [\mathrm{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \mathrm{lift}_{\mathrm{Maybe}}(\mathrm{lift}_{(,)}(\mathrm{id}, f)),$$

untersuchen wir alle Fälle für Eingaben von `match` und f, unter Verwendung der Definitionen:

$$\begin{aligned} \texttt{match}\ [\,] &= \mathrm{Nothing} & \qquad f\ [\,] &= \mathrm{Nothing} \\ \texttt{match}\ (a : as) &= \mathrm{Just}\ (a, as) & \qquad f\ (x : y) &= \mathrm{Just}\ (x, f\ y) \end{aligned}$$

Schließlich ...

Wir haben:

$$\text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)) = \{(\bot, \bot), (\text{Nothing}, \text{Nothing})\} \cup \{(\text{Just } x_1, \text{Just } y_1) \mid (x_1, y_1) \in \text{lift}_{(,)}(\text{id}, f)\}$$

$$\text{lift}_{(,)}(\text{id}, f) = \{(\bot, \bot)\} \cup \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [\mathrm{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)),$$

untersuchen wir alle Fälle für Eingaben von `match` und f, unter Verwendung der Definitionen:

$$\begin{aligned} \texttt{match}\ [\,] &= \text{Nothing} \\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as) \end{aligned} \qquad \begin{aligned} f\ [\,] &= \text{Nothing} \\ f\ (x : y) &= \text{Just}\ (x, f\ y) \end{aligned}$$

Schließlich ...

Wir haben:

$$\text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)) = \{(\bot, \bot), (\text{Nothing}, \text{Nothing})\} \cup \{(\text{Just } x_1, \text{Just } y_1) \mid (x_1, y_1) \in \text{lift}_{(,)}(\text{id}, f)\}$$

$$\text{lift}_{(,)}(\text{id}, f) = \{(\bot, \bot)\} \cup \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [T_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)),$$

untersuchen wir alle Fälle für Eingaben von `match` und f,
unter Verwendung der Definitionen:

$$\begin{aligned} \texttt{match}\ [\,] &= \text{Nothing} \\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as) \end{aligned} \qquad \begin{aligned} f\ [\,] &= \text{Nothing} \\ f\ (x : y) &= \text{Just}\ (x, f\ y) \end{aligned}$$

Schließlich ...

Wir haben:

$$\text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)) = \{(\bot, \bot), (\text{Nothing}, \text{Nothing})\} \cup \{(\text{Just } x_1, \text{Just } y_1) \mid (x_1, y_1) \in \text{lift}_{(,)}(\text{id}, f)\}$$

$$\text{lift}_{(,)}(\text{id}, f) = \{(\bot, \bot)\} \cup \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [\text{T}_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)),$$

untersuchen wir alle Fälle für Eingaben von `match` und f,
unter Verwendung der Definitionen:

$$\begin{aligned} \texttt{match}\ [\,] &= \text{Nothing} & \qquad f\ [\,] &= \text{Nothing} \\ \texttt{match}\ (a : as) &= \text{Just}\ (a, as) & \qquad f\ (x : y) &= \text{Just}\ (x, f\ y) \end{aligned}$$

Schließlich ...

Wir haben:

$$\text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)) = \{(\bot, \bot), (\text{Nothing}, \text{Nothing})\} \cup \{(\text{Just } x_1, \text{Just } y_1) \mid (x_1, y_1) \in \text{lift}_{(,)}(\text{id}, f)\}$$

$$\text{lift}_{(,)}(\text{id}, f) = \{(\bot, \bot)\} \cup \{((x_1, x_2), (y_1, y_2)) \mid x_1 = y_1 \wedge f\ x_2 = y_2\}$$

Um zu zeigen, dass

$$\forall x :: [T_1].\ (\texttt{match}\ x, \texttt{id}\ (f\ x)) \in \text{lift}_{\text{Maybe}}(\text{lift}_{(,)}(\text{id}, f)),$$

untersuchen wir alle Fälle für Eingaben von `match` und f,
unter Verwendung der Definitionen:

```
match []       = Nothing          f []      = Nothing
match (a : as) = Just (a, as)     f (x : y) = Just (x, f y)
```

Fertig!

Fazit

Typangaben:

- schränken das Verhalten von Programmen ein
- erlauben daher die Herleitung von Aussagen über Programme

Fazit

Typangaben:

- schränken das Verhalten von Programmen ein
- erlauben daher die Herleitung von Aussagen über Programme

Erhaltene Programmtransformationen:

- bauen insbesondere auf Polymorphismus und „Higher-Order“

Fazit

Typangaben:

- schränken das Verhalten von Programmen ein
- erlauben daher die Herleitung von Aussagen über Programme

Erhaltene Programmtransformationen:

- bauen insbesondere auf Polymorphismus und „Higher-Order“
- können nicht-triviale Nebenbedingungen haben

Fazit

Typangaben:

- schränken das Verhalten von Programmen ein
- erlauben daher die Herleitung von Aussagen über Programme

Erhaltene Programmtransformationen:

- bauen insbesondere auf Polymorphismus und „Higher-Order“
- können nicht-triviale Nebenbedingungen haben

Korrektheitsbeweise:

- sind zum Teil automatisierbar
- wo nicht, zumindest systematisch/zielgetrieben

Fazit

Typangaben:

- schränken das Verhalten von Programmen ein
- erlauben daher die Herleitung von Aussagen über Programme

Erhaltene Programmtransformationen:

- bauen insbesondere auf Polymorphismus und „Higher-Order“
- können nicht-triviale Nebenbedingungen haben

Korrektheitsbeweise:

- sind zum Teil automatisierbar
- wo nicht, zumindest systematisch/zielgetrieben
  (ähnlich für andere Transformationen [V., FLOPS'08])

Fazit

Typangaben:

- schränken das Verhalten von Programmen ein
- erlauben daher die Herleitung von Aussagen über Programme

Erhaltene Programmtransformationen:

- bauen insbesondere auf Polymorphismus und „Higher-Order"
- können nicht-triviale Nebenbedingungen haben

Korrektheitsbeweise:

- sind zum Teil automatisierbar
- wo nicht, zumindest systematisch/zielgetrieben
  (ähnlich für andere Transformationen [V., FLOPS'08])

Weiter von Interesse:

- größere Abdeckung von Sprachkonstrukten
- noch ausdrucksstärkere Typsysteme

Literatur I

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.

P. Johann and J. Voigtländer.
Free theorems in the presence of seq.
In *Principles of Programming Languages, Proceedings*, pages 99–110. ACM Press, 2004.

P. Johann and J. Voigtländer.
A family of syntactic logical relations for the semantics of Haskell-like languages.
Information and Computation, 207(2):341–368, 2009.

Literatur II

S.L. Peyton Jones, A. Tolmach, and C.A.R. Hoare.
Playing by the rules: Rewriting as a practical optimisation technique in GHC.
In *Haskell Workshop, Proceedings*, pages 203–233, 2001.

F. Stenger and J. Voigtländer.
Parametricity for Haskell with imprecise error semantics.
In *Typed Lambda Calculi and Applications, Proceedings*, volume 5608 of *LNCS*, pages 294–308. Springer-Verlag, 2009.

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.
Proving correctness via free theorems: The case of the destroy/build-rule.
In *Partial Evaluation and Program Manipulation, Proceedings*, pages 13–20. ACM Press, 2008.

J. Voigtländer.
Semantics and pragmatics of new shortcut fusion rules.
In *Functional and Logic Programming, Proceedings*, volume 4989 of *LNCS*, pages 163–179. Springer-Verlag, 2008.

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