Fortgeschrittene Funktionale Programmierung

13. Vorlesung

Janis Voigtländer

Universität Bonn

Wintersemester 2015/16

Erweiterung um Datentypen

Typen: $\tau := \quad \cdots \mid \mathsf{Bool} \mid [\tau]$

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

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

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

$$\frac{\Gamma \vdash t : \mathsf{Bool} \qquad \Gamma \vdash u_1 : \tau \qquad \Gamma \vdash u_2 : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{\mathsf{False} \to u_1 \,; \mathsf{True} \to u_2\}) : \tau}$$

$$\frac{\Gamma \vdash t : [\tau'] \qquad \Gamma \vdash u_1 : \tau \qquad \Gamma, x_1 : \tau', x_2 : [\tau'] \vdash u_2 : \tau}{\Gamma \vdash (\textbf{case } t \textbf{ of } \{[\,] \to u_1 \,; x_1 : x_2 \to u_2\}) : \tau}$$

Damit können wir jetzt zum Beispiel die bekannte Funktion `filter` $:: (\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha]$ schreiben, oder?

Rekursion in getyptem Lambda-Kalkül

Terme: $t := \;\cdots \mid \textbf{rec}\ t$

$$\frac{\Gamma \vdash t : \tau \to \tau}{\Gamma \vdash (\textbf{rec}\ t) : \tau}$$

Aus:

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

wird:

$$\begin{array}{l}
\textbf{rec}\ (\lambda f : (\alpha \to \text{Bool}) \to [\alpha] \to [\alpha]. \\
\qquad \lambda p : (\alpha \to \text{Bool}).\lambda l : [\alpha]. \\
\qquad \textbf{case}\ l\ \textbf{of}\ \{[\,] \qquad \to [\,]_\alpha\,; \\
\qquad\qquad\qquad\quad a : as \to \textbf{case}\ p\ a\ \textbf{of} \\
\qquad\qquad\qquad\qquad\qquad \{\text{False} \to f\ \ p\ as\,; \\
\qquad\qquad\qquad\qquad\qquad\ \ \text{True}\ \to a : (f\ \ p\ as)\}\})
\end{array}$$

Idee: $\texttt{f}\ (:: \tau) = \ldots\ \texttt{f}\ \ \ldots \quad \longrightarrow \quad \texttt{f} = \textbf{rec}\ (\lambda f : \tau.\ \ \ldots\ \ f\ \ \ldots)$

Weitere Erweiterungen unseres Lambda-Kalküls

Schließlich noch hinzu (weil semantisch interessant):

Terme: $t := \ \cdots \mid \textbf{seq}\ t\ t$

$$\frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash u : \tau_2}{\Gamma \vdash (\textbf{seq}\ t\ u) : \tau_2}$$

und (weil für explizite Polymorphie interessant):

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

Terme: $t := \ \cdots \mid \Lambda\alpha.t \mid t\ \tau$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda\alpha.t) : \forall\alpha.\tau}$$

$$\frac{\Gamma \vdash t : \forall\alpha.\tau}{\Gamma \vdash (t\ \tau') : \tau[\tau'/\alpha]}$$

Explizite Polymorphie: Beispiel

`filter`:

$$
\begin{array}{l}
\textbf{rec}\ (\lambda f : \forall\alpha.(\alpha \to \mathsf{Bool}) \to [\alpha] \to [\alpha].\\
\qquad \Lambda\alpha.\lambda p : (\alpha \to \mathsf{Bool}).\lambda l : [\alpha].\\
\qquad \textbf{case}\ l\ \textbf{of}\ \{[\,] \qquad \to [\,]_\alpha\,;\\
\qquad\qquad\qquad\quad a : as \to \textbf{case}\ p\ a\ \textbf{of}\\
\qquad\qquad\qquad\qquad\qquad \{\mathsf{False} \to f\ \alpha\ p\ as\,;\\
\qquad\qquad\qquad\qquad\qquad\ \ \mathsf{True}\ \to a : (f\ \alpha\ p\ as)\}\})
\end{array}
$$

GHC-Core

`filter`:

```
%rec
{f :: %forall alpha . (alpha -> Bool) ->
                      [] alpha ->
                      [] alpha =
   \ @ alpha
     (p :: alpha -> Bool)
     (l :: [] alpha) ->
       %case ([] alpha) l
       %of (_ :: [] alpha)
         {[] -> [] @ alpha;
          (:) (a :: alpha) (as :: [] alpha) ->
            %case ([] alpha) (p a)
            %of (_ :: Bool)
              {False -> f @ alpha p as;
               True -> (:) @ alpha a (f @ alpha p as)}}};
```

Operationelle Semantik

Zur Erinnerung: eine DP-Klausuraufgabe

Gegeben seien die folgenden Funktionsdefinitionen:

$\texttt{f} :: [\mathsf{Int}] \to [\mathsf{Int}]$
$\texttt{f}\ [\,] \qquad = [\,]$
$\texttt{f}\ (x : xs) = (\texttt{g}\ x) : (\texttt{f}\ xs)$

$\texttt{g} :: \mathsf{Int} \to \mathsf{Int}$
$\texttt{g}\ 3 = \texttt{g}\ 4$
$\texttt{g}\ n = n + 1$

sowie die vordefinierten Funktionen `head` und `tail`.
Notieren Sie die einzelnen Haskell-Auswertungsschritte für folgenden Ausdruck (bis zum Endergebnis, und unter genauer Beachtung von Haskells Auswertungsstrategie!):

$$\begin{aligned}\texttt{head}\ (\texttt{tail}\ (\texttt{f}\ [3, 3+1])) \quad &= \quad \underline{\hspace{6cm}} \\ &= \quad \underline{\hspace{6cm}} \\ &= \quad \underline{\hspace{6cm}} \\ &\ \vdots\end{aligned}$$

Eine ausgewählte Teilmenge des Kalküls:

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

Terme: $t := x \mid \lambda x : \tau.t \mid t\ t \mid \Lambda\alpha.t \mid t\ \tau \mid []_\tau \mid$
$t : t \mid \textbf{case}\ t\ \textbf{of}\ \{\cdots\} \mid \textbf{rec}\ t \mid \textbf{seq}\ t\ t$

$$\Gamma, x : \tau \vdash x : \tau \qquad\qquad \Gamma \vdash []_\tau : [\tau]$$

$$\frac{\Gamma, x : \tau_1 \vdash t : \tau_2}{\Gamma \vdash (\lambda x : \tau_1.t) : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash t : \tau_1 \to \tau_2 \qquad \Gamma \vdash u : \tau_1}{\Gamma \vdash (t\ u) : \tau_2}$$

$$\frac{\alpha, \Gamma \vdash t : \tau}{\Gamma \vdash (\Lambda\alpha.t) : \forall\alpha.\tau} \qquad \frac{\Gamma \vdash t : \forall\alpha.\tau}{\Gamma \vdash (t\ \tau') : \tau[\tau'/\alpha]} \qquad \frac{\Gamma \vdash t : \tau \to \tau}{\Gamma \vdash (\textbf{rec}\ t) : \tau}$$

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash u : [\tau]}{\Gamma \vdash (t : u) : [\tau]} \qquad \frac{\Gamma \vdash t : \tau_1 \qquad \Gamma \vdash u : \tau_2}{\Gamma \vdash (\textbf{seq}\ t\ u) : \tau_2}$$

$$\frac{\Gamma \vdash t : [\tau'] \qquad \Gamma \vdash u_1 : \tau \qquad \Gamma, x_1 : \tau', x_2 : [\tau'] \vdash u_2 : \tau}{\Gamma \vdash (\textbf{case}\ t\ \textbf{of}\ \{[] \to u_1 ; x_1 : x_2 \to u_2\}) : \tau}$$

Small-Step Semantics – Werte/Reduktionsschritte

$$\text{Werte: } v := \lambda x : \tau . t \mid \Lambda\alpha . t \mid [\,]_\tau \mid t : t \qquad \text{(spezielle Terme)}$$

Reduktionen:

$$(\lambda x : \tau . t)\ u \rightsquigarrow t[u/x] \qquad (\Lambda\alpha . t)\ \tau \rightsquigarrow t[\tau/\alpha]$$

$$\mathbf{case}\ [\,]_\tau\ \mathbf{of}\ \{\cdots\} \rightsquigarrow u_1 \qquad \mathbf{case}\ t_1 : t_2\ \mathbf{of}\ \{\cdots\} \rightsquigarrow u_2[t_i/x_i]$$

$$\mathbf{rec}\ t \rightsquigarrow t\ (\mathbf{rec}\ t) \qquad \mathbf{seq}\ v\ u \rightsquigarrow u$$

einfaches Beispiel:

$$\begin{aligned}
& (\Lambda\alpha . \mathbf{rec}\ (\lambda x : \alpha . x))\ \tau \\
\rightsquigarrow\ & \mathbf{rec}\ (\lambda x : \tau . x) \\
\rightsquigarrow\ & (\lambda x : \tau . x)\ (\mathbf{rec}\ (\lambda x : \tau . x)) \\
\rightsquigarrow\ & \mathbf{rec}\ (\lambda x : \tau . x) \\
\rightsquigarrow\ & (\lambda x : \tau . x)\ (\mathbf{rec}\ (\lambda x : \tau . x)) \\
\rightsquigarrow\ & \cdots
\end{aligned}$$

Small-Step Semantics – Reduktion in Kontext

Kontexte: $E := (-\ t) \mid (-\ \tau) \mid (\textbf{case}\ -\ \textbf{of}\ \{\cdots\}) \mid (\textbf{seq}\ -\ t)$
$S := \mathit{Id} \mid S \circ E$

Transitionen:

$$
\begin{array}{rcll}
(S, t) & \rightarrowtail & (S, t') & \text{wenn } t \leadsto t' \\
(S, E\{t\}) & \rightarrowtail & (S \circ E, t) & \text{wenn } t \text{ kein Wert} \\
(S \circ E, v) & \rightarrowtail & (S, E\{v\}) &
\end{array}
$$

Auswertung: $t \Downarrow v$ iff $(\mathit{Id}, t) \rightarrowtail^* (\mathit{Id}, v)$

einfaches Beispiel:

$$
\begin{array}{rl}
& (\mathit{Id}, (\textbf{seq}\ ((\lambda x : [\tau].x)\ [\,]_\tau)\ (\lambda y : [\tau].y))\ [\,]_\tau) \\
\rightarrowtail & (\mathit{Id} \circ (-\ [\,]_\tau), \textbf{seq}\ ((\lambda x : [\tau].x)\ [\,]_\tau)\ (\lambda y : [\tau].y)) \\
\rightarrowtail & (\mathit{Id} \circ (-\ [\,]_\tau) \circ (\textbf{seq}\ -\ (\lambda y : [\tau].y)), (\lambda x : [\tau].x)\ [\,]_\tau) \\
\rightarrowtail & (\mathit{Id} \circ (-\ [\,]_\tau) \circ (\textbf{seq}\ -\ (\lambda y : [\tau].y)), [\,]_\tau) \\
\rightarrowtail & (\mathit{Id} \circ (-\ [\,]_\tau), \textbf{seq}\ [\,]_\tau\ (\lambda y : [\tau].y)) \\
\rightarrowtail & (\mathit{Id} \circ (-\ [\,]_\tau), \lambda y : [\tau].y) \\
\rightarrowtail & (\mathit{Id}, (\lambda y : [\tau].y)\ [\,]_\tau) \\
\rightarrowtail & (\mathit{Id}, [\,]_\tau)
\end{array}
$$

Small-Step Semantics – komplexeres Beispiel

$$
\begin{array}{rl}
 & (Id, (\textbf{rec}\ t)\ g\ []_\tau) \\
\rightarrowtail & (Id \circ (-\ []_\tau), (\textbf{rec}\ t)\ g) \\
\rightarrowtail & (Id \circ (-\ []_\tau) \circ (-\ g), \textbf{rec}\ t) \\
\rightarrowtail & (Id \circ (-\ []_\tau) \circ (-\ g), t\ (\textbf{rec}\ t)) \\
\rightarrowtail & (Id \circ (-\ []_\tau) \circ (-\ g), \lambda p : (\tau \to \mathrm{Bool}).\lambda l : [\tau]. \\
 & \qquad\qquad\qquad\qquad\qquad \textbf{case}\ l\ \textbf{of}\ \{[] \quad\ \ \to []_\tau\,; \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad a : as \to \textbf{case}\ p\ a\ \textbf{of}\ \{\cdots\}\})
\end{array}
$$

$$
\begin{array}{l}
\text{wobei } t = \lambda f : (\tau \to \mathrm{Bool}) \to [\tau] \to [\tau]. \\
\qquad\qquad \lambda p : (\tau \to \mathrm{Bool}).\lambda l : [\tau]. \\
\qquad\qquad \textbf{case}\ l\ \textbf{of}\ \{[] \quad\ \ \to []_\tau\,; \\
\qquad\qquad\qquad\qquad\quad a : as \to \textbf{case}\ p\ a\ \textbf{of} \\
\qquad\qquad\qquad\qquad\qquad\qquad \{\mathrm{False} \to f\ \ p\ as\,; \\
\qquad\qquad\qquad\qquad\qquad\qquad\ \ \mathrm{True}\ \ \to a : (f\ \ p\ as)\}\}
\end{array}
$$

Small-Step Semantics – komplexeres Beispiel

$$
\begin{aligned}
&\rightarrowtail (Id \circ (-\ [\,]_\tau) \circ (-\ g), \lambda p : (\tau \to \mathrm{Bool}).\lambda l : [\tau].\\
&\qquad\qquad\qquad \textbf{case } l \textbf{ of } \{[\,] \quad\ \ \to [\,]_\tau\,;\\
&\qquad\qquad\qquad\qquad\quad a : as \to \textbf{case } p\ a \textbf{ of } \{\cdots\}\})\\
&\rightarrowtail (Id \circ (-\ [\,]_\tau), (\lambda p : (\tau \to \mathrm{Bool}).\lambda l : [\tau].\\
&\qquad\qquad \textbf{case } l \textbf{ of } \{[\,] \quad\ \ \to [\,]_\tau\,;\\
&\qquad\qquad\qquad a : as \to \textbf{case } p\ a \textbf{ of } \{\cdots\}\})\ g)\\
&\rightarrowtail (Id \circ (-\ [\,]_\tau), \lambda l : [\tau].\textbf{case } l \textbf{ of } \{[\,] \quad\ \ \to [\,]_\tau\,;\\
&\qquad\qquad\qquad\qquad\qquad a : as \to \textbf{case } g\ a \textbf{ of } \{\cdots\}\})
\end{aligned}
$$

$$
\begin{aligned}
\text{wobei } t = {} & \lambda f : (\tau \to \mathrm{Bool}) \to [\tau] \to [\tau].\\
& \lambda p : (\tau \to \mathrm{Bool}).\lambda l : [\tau].\\
& \textbf{case } l \textbf{ of } \{[\,] \quad\ \ \to [\,]_\tau\,;\\
& \qquad\quad a : as \to \textbf{case } p\ a \textbf{ of}\\
& \qquad\qquad\qquad\quad \{\mathrm{False} \to f\ \ p\ as\,;\\
& \qquad\qquad\qquad\quad\ \ \mathrm{True}\ \ \to a : (f\ \ p\ as)\}\}
\end{aligned}
$$

Small-Step Semantics – komplexeres Beispiel

$$
\begin{aligned}
&\rightarrowtail (\mathit{Id} \circ (-\ []_\tau), \lambda l : [\tau].\textbf{case}\ l\ \textbf{of}\ \{[] \quad\ \to []_\tau\,; \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad a : as \to \textbf{case}\ g\ a\ \textbf{of}\ \{\cdots\}\}) \\
&\rightarrowtail (\mathit{Id}, (\lambda l : [\tau].\textbf{case}\ l\ \textbf{of}\ \{[] \quad\ \to []_\tau\,; \\
&\qquad\qquad\qquad\qquad\qquad\qquad a : as \to \textbf{case}\ g\ a\ \textbf{of}\ \{\cdots\}\})\ []_\tau) \\
&\rightarrowtail (\mathit{Id}, \textbf{case}\ []_\tau\ \textbf{of}\ \{[] \quad\ \to []_\tau\,; \\
&\qquad\qquad\qquad\qquad a : as \to \textbf{case}\ g\ a\ \textbf{of}\ \{\cdots\}\}) \\
&\rightarrowtail (\mathit{Id}, []_\tau)
\end{aligned}
$$

$$
\begin{aligned}
\text{wobei } t = {} & \lambda f : (\tau \to \text{Bool}) \to [\tau] \to [\tau]. \\
& \lambda p : (\tau \to \text{Bool}).\lambda l : [\tau]. \\
& \textbf{case}\ l\ \textbf{of}\ \{[] \quad\ \to []_\tau\,; \\
& \qquad\qquad a : as \to \textbf{case}\ p\ a\ \textbf{of} \\
& \qquad\qquad\qquad\qquad \{\text{False} \to f\ p\ as\,; \\
& \qquad\qquad\qquad\qquad\ \ \text{True} \ \to a : (f\ p\ as)\}\}
\end{aligned}
$$

Small-Step Semantics – Zusammenfassung

Werte: $v := \lambda x : \tau.t \mid \Lambda\alpha.t \mid [\,]_\tau \mid t : t$

Reduktionen:

$$(\lambda x : \tau.t)\ u \rightsquigarrow t[u/x] \qquad (\Lambda\alpha.t)\ \tau \rightsquigarrow t[\tau/\alpha]$$

$$\textbf{case}\ [\,]_\tau\ \textbf{of}\ \{\cdots\} \rightsquigarrow u_1 \qquad \textbf{case}\ t_1 : t_2\ \textbf{of}\ \{\cdots\} \rightsquigarrow u_2[t_i/x_i]$$

$$\textbf{rec}\ t \rightsquigarrow t\ (\textbf{rec}\ t) \qquad \textbf{seq}\ v\ u \rightsquigarrow u$$

Kontexte:
$$E := (-\ t) \mid (-\ \tau) \mid (\textbf{case}\ -\ \textbf{of}\ \{\cdots\}) \mid (\textbf{seq}\ -\ t)$$
$$S := Id \mid S \circ E$$

Transitionen:

$$\begin{array}{rcll} (S, t) & \rightarrowtail & (S, t') & \text{wenn } t \rightsquigarrow t' \\ (S, E\{t\}) & \rightarrowtail & (S \circ E, t) & \text{wenn } t \text{ kein Wert} \\ (S \circ E, v) & \rightarrowtail & (S, E\{v\}) & \end{array}$$

Auswertung: $t \Downarrow v$ iff $(Id, t) \rightarrowtail^* (Id, v)$

Divergenz: $t{\Uparrow}$ iff $\neg\exists v.\, t \Downarrow v$

Big-Step Semantics (direkte Definition von $\Downarrow$)

Werte: $v := \lambda x : \tau . t \mid \Lambda \alpha . t \mid [\,]_\tau \mid t : t$ (wie zuvor)

Auswertung:

$$v \Downarrow v$$

$$\frac{t \Downarrow (\lambda x : \tau . t') \qquad t'[u/x] \Downarrow v}{(t\ u) \Downarrow v} \qquad \frac{t \Downarrow (\Lambda \alpha . t') \qquad t'[\tau/\alpha] \Downarrow v}{(t\ \tau) \Downarrow v}$$

$$\frac{t \Downarrow [\,]_\tau \qquad u_1 \Downarrow v}{(\textbf{case}\ t\ \textbf{of}\ \{[\,] \to u_1\,;\, x_1 : x_2 \to u_2\}) \Downarrow v}$$

$$\frac{t \Downarrow (t_1 : t_2) \qquad u_2[t_i/x_i] \Downarrow v}{(\textbf{case}\ t\ \textbf{of}\ \{[\,] \to u_1\,;\, x_1 : x_2 \to u_2\}) \Downarrow v}$$

$$\frac{(t\ (\textbf{rec}\ t)) \Downarrow v}{(\textbf{rec}\ t) \Downarrow v} \qquad \frac{t \Downarrow v' \qquad u \Downarrow v}{(\textbf{seq}\ t\ u) \Downarrow v}$$

Big-Step Semantics – einfaches Beispiel

$$\dfrac{\dfrac{\dfrac{(\lambda x : [\tau].x) \Downarrow (\lambda x : [\tau].x) \qquad x[[\,]_\tau/x] \Downarrow [\,]_\tau}{((\lambda x : [\tau].x)\ [\,]_\tau) \Downarrow [\,]_\tau} \qquad (\lambda y : [\tau].y) \Downarrow (\lambda y : [\tau].y)}{(\mathbf{seq}\ ((\lambda x : [\tau].x)\ [\,]_\tau)\ (\lambda y : [\tau].y)) \Downarrow (\lambda y : [\tau].y)} \qquad y[[\,]_\tau/y] \Downarrow [\,]_\tau}{((\mathbf{seq}\ ((\lambda x : [\tau].x)\ [\,]_\tau)\ (\lambda y : [\tau].y))\ [\,]_\tau) \Downarrow [\,]_\tau}$$

Vergleiche mit:

$$\begin{array}{ll}
 & (Id, (\mathbf{seq}\ ((\lambda x : [\tau].x)\ [\,]_\tau)\ (\lambda y : [\tau].y))\ [\,]_\tau) \\
\rightarrowtail & (Id \circ (-\ [\,]_\tau), \mathbf{seq}\ ((\lambda x : [\tau].x)\ [\,]_\tau)\ (\lambda y : [\tau].y)) \\
\rightarrowtail & (Id \circ (-\ [\,]_\tau) \circ (\mathbf{seq}\ -\ (\lambda y : [\tau].y)), (\lambda x : [\tau].x)\ [\,]_\tau) \\
\rightarrowtail & (Id \circ (-\ [\,]_\tau) \circ (\mathbf{seq}\ -\ (\lambda y : [\tau].y)), [\,]_\tau) \\
\rightarrowtail & (Id \circ (-\ [\,]_\tau), \mathbf{seq}\ [\,]_\tau\ (\lambda y : [\tau].y)) \\
\rightarrowtail & (Id \circ (-\ [\,]_\tau), \lambda y : [\tau].y) \\
\rightarrowtail & (Id, (\lambda y : [\tau].y)\ [\,]_\tau) \\
\rightarrowtail & (Id, [\,]_\tau)
\end{array}$$

Allgemeine Verbindung Small-Step/Big-Step

Beide Arten der Definition von $\Downarrow$ sind äquivalent.

Ein struktureller Zusammenhang, für alle Sprachkonstrukte außer für **rec**:

$$\frac{t \Downarrow v' \qquad t' \Downarrow v}{E\{t\} \Downarrow v}\,(E\{v'\} \leadsto t')$$

Vorteile/Nachteile von Small-Step/Big-Step?

Observational Equivalence

Frage: Wann sollten zwei Terme denn nun allgemein als semantisch äquivalent angesehen werden?

Vorschlag? $t \equiv t'$ gdw. für jeden Wert v, $t \Downarrow v \Leftrightarrow t' \Downarrow v$

Problem: Verschiedene Terme gleichem „extensionalen" Verhalten würden nicht immer als äquivalent angesehen, z.B. `heapsort` $\not\equiv$ `mergesort`.

Lösung:

- Erlaube (nur) bestimmte Beobachtungen, zum Beispiel Auswertung/Termination auf ausgewählten Typen.
- Aber verlange, dass äquivalente Terme in jedem möglichen Kontext zu gleichen Beobachtungen führen.
- Also, wähle als $\equiv$ die größte Kongruenzrelation, die bezüglich der erlaubten Beobachtungen „adäquat" ist (noch zu definieren).

Kongruenz: Kompatibilität und Substitutivität

$$x \equiv x \qquad\qquad [\,]_\tau \equiv [\,]_\tau$$

$$\frac{t \equiv t'}{(\lambda x : \tau_1 . t) \equiv (\lambda x : \tau_1 . t')} \qquad\qquad \frac{t \equiv t' \qquad u \equiv u'}{(t\ u) \equiv (t'\ u')}$$

$$\frac{t \equiv t'}{(\Lambda\alpha . t) \equiv (\Lambda\alpha . t')} \qquad \frac{t \equiv t'}{(t\ \tau) \equiv (t'\ \tau)} \qquad \frac{t \equiv t'}{(\textbf{rec}\ t) \equiv (\textbf{rec}\ t')}$$

$$\frac{t \equiv t' \qquad u \equiv u'}{(t : u) \equiv (t' : u')} \qquad\qquad \frac{t \equiv t' \qquad u \equiv u'}{(\textbf{seq}\ t\ u) \equiv (\textbf{seq}\ t'\ u')}$$

$$\frac{t \equiv t' \qquad u_1 \equiv u_1' \qquad u_2 \equiv u_2'}{(\textbf{case}\ t\ \textbf{of}\ \{\cdots\}) \equiv (\textbf{case}\ t'\ \textbf{of}\ \{\cdots\})}$$

$$\frac{t \equiv t' \qquad u \equiv u'}{t[u/x] \equiv t'[u'/x]} \qquad\qquad \frac{t \equiv t'}{t[\tau/\alpha] \equiv t'[\tau/\alpha]}$$

Adäquatheit

Sei τ ein Typ und seien t, t' Terme mit $\vdash t : \tau$ und $\vdash t' : \tau$.
Es gibt verschiedene mögliche Ersatzalternativen für
„$t \equiv t'$ gdw. (oder: impliziert) für jeden Wert v, $t \Downarrow v \Leftrightarrow t' \Downarrow v$“:

- $t \equiv t'$ impliziert $t\Downarrow \Leftrightarrow t'\Downarrow$
- $t \equiv t'$ impliziert $t\Downarrow \Leftrightarrow t'\Downarrow$, gefordert lediglich für $\tau = [\tau']$
- $t \equiv t'$ impliziert $t \Downarrow [\,]_{\tau'} \Leftrightarrow t' \Downarrow [\,]_{\tau'}$, gefordert lediglich für $\tau = [\tau']$

Sie induzieren alle die gleiche größte Kongruenzrelation!
„Observational Equivalence“

Überlegungen:

- Was wäre wenn man die kleinste adäquate Kongruenzrelation betrachten würde?
- Was wäre wenn man einfach die größte Kongruenzrelation, ohne Adäquatheitsforderung, betrachten würde?

Einige Resultate zu Observational Equivalence

- Wenn $t \equiv t'$, dann $t\Downarrow \Leftrightarrow t'\Downarrow$.
- Wenn $t\Uparrow$, $t'\Uparrow$, und t, t' haben selben Typ, dann $t \equiv t'$.
- Wenn $t \Downarrow v$, dann $t \equiv v$.
- Wenn $t \leadsto t'$, dann $t \equiv t'$.
- Für Terme t, t' des selben Funktionstyps:

$$t \equiv t' \;\text{ gdw. }\; (\forall u.\, (t\ u) \equiv (t'\ u)) \wedge (t\Downarrow \Leftrightarrow t'\Downarrow)$$

- Für Terme t, t' des selben polymorphen Typs:

$$t \equiv t' \;\text{ gdw. }\; (\forall \tau.\, (t\ \tau) \equiv (t'\ \tau)) \wedge (t\Downarrow \Leftrightarrow t'\Downarrow)$$

Leider ist $\equiv$ jedoch für viele Beweise zu konkreten Programmen schwer handhabbar, insbesondere da nicht „direkt“ und kompositionell definiert.