Universität Siegen
Institut für Theoretische Informatik
Fachbereich Mathematik

SS 2007

Vorlesungsmitschrift

Theorie der Programmierung III

Benedikt Meurer

16.07.2007

Inhaltsverzeichnis

1 Denotationelle Semantik

Idee: Mathematisches Modell
Im Modell werden Funktionen benutzt, zum Beispiel:

- $\lambda x : \mathbf{int}.x \leadsto$ Identität auf $\mathbb{Z}$
- $\lambda x : \mathbf{int}.x + 1 \leadsto$ Nachfolgerfunktion
- $\mathbf{rec}\, \mathit{fact} : \mathbf{int} \to \mathbf{int}.\lambda x : \mathbf{int}.\mathbf{if}\, x = 0\, \mathbf{then}\, 1\, \mathbf{else}\, \ldots \leadsto$ Fakultätsfunktion (partiell, da nur auf $\mathbb{N}$ definiert)
- $\lambda f : \mathbf{int} \to \mathbf{int}.\lambda x : \mathbf{int}.f\,(f\,x) \leadsto$ die Funktion *twice* auf $(\mathbb{Z} \to \mathbb{Z})$, $\mathit{twice}\,(f) = f \circ f$

Problem: Ungetypte Programmiersprachen, zum Beispiel $\lambda x.x\,x$ lässt sich nicht so einfach als Funktion ansehen.

Vorteile denotationeller Semantik:

- Anschaulichkeit (λ-Abstraktion sollte man sich als Funktion vorstellen)
- Sie ist „kompositionell“, d.h. die Semantik eines zusammengesetzten Ausdrucks wird (induktiv) definiert über die Semantik seiner Teilausdrücke (d.h. auch nicht abgeschlossenen Ausdrücken muss man eine Semantik zuordnen)

Nachteile denotationeller Semantik:

- Kleine Änderungen in der Sprache können eine große Änderung im Modell notwendig machen

2 Denotationelle Semantik für $\mathcal{L}_1^t$

Zuerst: Denotationelle Semantik für $\mathcal{L}_1^t$ (einfach getypt, keine Rekursion)
$\rightsquigarrow$ Denotationelle Semantik besteht aus *totalen* Funktionen (weil alle Programme terminieren)
Genauer: Für jeden Typ τ wird ein *semantischer Bereich* $[\![\tau]\!]$ definiert durch Induktion über die Struktur von τ:

$$\begin{array}{rcl}
[\![\mathbf{int}]\!] & = & \mathbb{Z} \\
[\![\mathbf{bool}]\!] & = & Bool = \{true, false\} \\
[\![\mathbf{unit}]\!] & = & \{(\,)\} \\
[\![\tau \to \tau']\!] & = & ([\![\tau]\!] \xrightarrow{t} [\![\tau']\!]) \\
 & (= & \text{Menge der totalen Funktionen } f : [\![\tau]\!] \to [\![\tau']\!])
\end{array}$$

Ziel: Jedem abgeschlossenen Ausdruck $e :: \tau$ soll ein Element aus $[\![\tau]\!]$ zugeordnet werden.

Frage: Wie behandelt man nicht abgeschlossene Ausdrücke (z.B. $x + 1$)?
$\rightsquigarrow$ Man benötigt eine „Belegung“ oder „Umgebung“, die den frei vorkommenden Namen Elemente aus dem richtigen semantischen Bereich zuordnet.

Definition 2.1

(a) *Eine* Umgebung *η ist eine (totale) Funktion* $\eta : Id \to \bigcup_{\tau \in Type} [\![\tau]\!]$.

(b) *η passt zu Γ (Schreibweise: $\eta \models \Gamma$), wenn für alle $id \in dom\,(\Gamma)$ gilt:*
Wenn $\Gamma(id) = \tau$, dann $\eta\,(id) \in [\![\tau]\!]$, d.h. $\eta\,(id) \in [\![\Gamma(id)]\!]$.

Env sei die Menge aller Umgebungen η, und $Env_\Gamma = \{\eta \mid \eta \models \Gamma\}$.

Ziel: Jedem gültigen Typurteil $\Gamma \triangleright e :: \tau$ wird bei vorgegebenener Umgebung $\eta \in Env_\Gamma$ ein Element $[\![\Gamma \triangleright e :: \tau]\!]\eta \in [\![\tau]\!]$ zugeordnet, durch Induktion über die Struktur von e.

Definition 2.2 *Dazu definieren wir $[\![\Gamma \triangleright e :: \tau]\!]\eta$ induktiv wie folgt:*

- $[\![\Gamma \triangleright n :: \mathbf{int}]\!]\eta = n$
- $[\![\Gamma \triangleright b :: \mathbf{bool}]\!]\eta = b$
- $[\![\Gamma \triangleright (\,) :: \mathbf{unit}]\!]\eta = (\,)$
- $[\![\Gamma \triangleright op :: \mathbf{int} \to \mathbf{int} \to \mathbf{int}]\!]\eta = op^I_{curry} : \mathbb{Z} \to (\mathbb{Z} \xrightarrow{t} \mathbb{Z})$,
  wobei $op^I_{curry}\, n_1 = f : \mathbb{Z} \to \mathbb{Z}$ *mit* $f\, n_2 = op^I(n_1, n_2)$

  Bemerkung: Division wird weggelassen, damit keine exceptions auftreten.
- $[\![\Gamma \triangleright id :: \tau]\!]\eta = \eta(id) \in [\![\tau]\!]$, *da* $\eta \models \Gamma$ *und* $\Gamma(id) = \tau$
- $[\![\Gamma \triangleright e_1\, e_2 :: \tau]\!]\eta = \underbrace{([\![\Gamma \triangleright e_1 :: \tau' \to \tau]\!]\eta)}_{\in [\![\tau' \to \tau]\!] = ([\![\tau']\!] \to [\![\tau]\!])} \underbrace{([\![\Gamma \triangleright e_2 :: \tau']\!]\eta)}_{\in [\![\tau']\!]}$
- $[\![\Gamma \triangleright \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 :: \tau]\!]\eta = \begin{cases} [\![\Gamma \triangleright e_1 :: \tau]\!]\eta & \text{falls } [\![\Gamma \triangleright e_0 :: \mathbf{bool}]\!]\eta = true \\ [\![\Gamma \triangleright e_2 :: \tau]\!]\eta & \text{falls } [\![\Gamma \triangleright e_0 :: \mathbf{bool}]\!]\eta = false \end{cases}$

- $[\![\Gamma \triangleright \lambda id : \tau.e :: \tau \to \tau']\!]\eta = f : [\![\tau]\!] \to [\![\tau']\!], d \mapsto [\![\Gamma[^\tau/_{id}] \triangleright e :: \tau']\!]\eta[^d/_{id}]$

  <u>Man beachte:</u> Wenn $\eta \models \Gamma$, dann $\eta[^d/_{id}] \models \Gamma[^\tau/_{id}]$, da $d \in [\![\tau]\!]$. Also ist (nach Induktionsannahme) $[\![\Gamma[^\tau/_{id}] \triangleright e :: \tau']\!]\eta[^d/_{id}] \in [\![\tau']\!]$
  $\leadsto$ f ist tatsächlich eine Funktion von $[\![\tau]\!]$ nach $[\![\tau']\!]$, also $f \in [\![\tau \to \tau']\!]$.

Die Überlegungen zeigen: $[\![\Gamma \triangleright e :: \tau]\!]\eta$ ist wohldefiniert, falls $\eta \in Env_\Gamma$.

Satz 2.1 (Koinzidenzlemma) *Wenn $\Gamma =_{free\,(e)} \Gamma'$, $\eta \in Env_\Gamma$, $\eta' \in Env_{\Gamma'}$ und $\eta =_{free\,(e)} \eta'$, dann gilt:*

$$[\![\Gamma \triangleright e :: \tau]\!]\eta \quad = \quad [\![\Gamma' \triangleright e :: \tau]\!]\eta'$$

In Worten besagt der Satz: Die Semantik von e hängt nur von der Belegung der frei vorkommenden Namen ab.
<u>Speziell:</u> Wenn e abgeschlossen ist, kommt es auf (Γ und) η nicht an, d.h. $[\![e :: \tau]\!]\eta$ ist abhängig von η und man schreibt dan einfach $[\![e :: \tau]\!]$.

<u>Beispiel:</u> $[\![\lambda x : \textbf{int}.x + 1 :: \textbf{int} \to \textbf{int}]\!] = succ : \mathbb{Z} \to \mathbb{Z}$

Satz 2.2 (Substitutionslemma) *Wenn $\Gamma[^{\tau'}/_{id}] \triangleright e :: \tau$ und $\Gamma \triangleright e' :: \tau'$, dann gilt:*

(a) $\Gamma \triangleright e[^{e'}/_{id}] :: \tau$ (vgl. TP I)

(b) für alle $\eta \in Env_\Gamma$:

$$[\![\Gamma \triangleright e[^{e'}/_{id}] :: \tau]\!]\eta = [\![\Gamma[^{\tau'}/_{id}] \triangleright e :: \tau]\!]\eta[^{[\![\Gamma \triangleright e' :: \tau']\!]\eta}/_{id}]$$

In Worten: Die syntaktische Substitution $e[^{e'}/_{id}]$ entspricht einem „semantischen Einsetzen“: Das Element $[\![\Gamma \triangleright e' :: \tau']\!]\eta$ wird in η an id „gebunden“.

<u>Ziel:</u> Es soll gezeigt werden, dass denotationelle und operationelle Semantik in folgendem Sinn übereinstimmen: Wenn τ Basistyp, dann gilt:

$$e \xrightarrow{*} c \quad \Leftrightarrow \quad [\![e :: \tau]\!] = c$$

Satz 2.3 (Adäquatheit der denotationellen Semantik) *Sei β Basistyp. Dann gilt für jeden abgeschlossenen Ausdruck $e :: \beta$:*

$$e \xrightarrow{*} c \quad \Leftrightarrow \quad [\![e :: \beta]\!] = c$$

„$\Rightarrow$“ In Übung 1 bewiesen: Jeder small step ist semantikerhaltend, also folgt aus $e \xrightarrow{*} c$, dass $[\![e :: \beta]\!] = [\![c :: \beta]\!] = c$.

„$\Leftarrow$“ Man benötigt Induktion über die Grösse von e.
$\leadsto$ Passende Induktionsbehauptung formulieren, die sich auf (nicht unbedingt abgeschlossene) Ausdrücke beliebigen Typs τ bezieht.

Zunächst: Verallgemeinerung auf abgeschlossene Ausdrücke.

Definition 2.3 *$Exp^\tau = \{e \in Exp \mid e :: \tau\}$ Menge der abgeschlossenen Ausdrücke vom Typ τ. $Val^\tau = Exp^\tau \cap Val$. Wir definieren Relationen*

$$\begin{array}{lll} \leq_\tau & \subseteq & [\![\tau]\!] \times Exp^\tau \\ \leq_\tau^0 & \subseteq & [\![\tau]\!] \times Val^\tau \end{array}$$

durch Induktion über τ:

(i) $\leq_\beta^0$ *ist die Gleichheit auf* $[\![\beta]\!]$ $(= Val^\beta)$

(ii) $\leq_{\tau\to\tau'}^0 \subseteq [\![\tau \to \tau']\!] \times Val^{\tau\to\tau'}$ *ist definiert durch:*

$$f \leq_{\tau\to\tau'}^0 v \quad \Leftrightarrow \quad \begin{array}{l} \textit{für alle } d \in [\![\tau]\!] \textit{ und alle } v' \in Val^\tau \textit{ gilt:} \\ d \leq_\tau^0 v' \Rightarrow f\,d \leq_{\tau'} v\,v' \end{array}$$

(iii) $\leq_\tau \subseteq [\![\tau]\!] \times Exp^\tau$ *ist definiert durch:*

$$d \leq_\tau e \quad \Leftrightarrow \quad \textit{es ex. ein } v \in Val^\tau \textit{ mit } e \xrightarrow{*} v \textit{ und } d \leq_\tau^0 v$$

Satz 2.4 *Für jeden Ausdruck $e \in Exp^\tau$ gilt:*

$$[\![e :: \tau]\!] \quad \leq_\tau \quad e$$

Intuitiv: e verhält sich (operationell) so, wie es durch die denotationelle Semantik beschrieben wird.
Um Satz 2.4 durch vollständige Induktion über die Struktur von e zu beweisen, muss die Aussage auf nicht abgeschlossene Ausdrücke verallgemeinert werden:

Lemma 2.1 *Wenn $\underbrace{[id_1 : \tau_1, \ldots, id_n : \tau_n]}_{\Gamma} \rhd e :: \tau$ und $\eta \models \Gamma$, dann gilt für alle $e_i \in Exp^{\tau_i}$ mit $i = 1, \ldots, n$: Wenn $\eta\,(id_i) \leq_{\tau_i} e_i$ für $i = 1, \ldots, n$, dann gilt:*

$$[\![\Gamma \rhd e :: \tau]\!]\eta \quad \leq_\tau \quad e[{}^{e_i}/_{id_i}]_{i=1}^n$$

<u>Beachte:</u> Da $\mathit{free}\,(e) \subseteq \mathit{dom}\,(\Gamma) = \{id_1, \ldots, id_n\}$ folgt: $e[{}^{e_i}/_{id_i}]_{i=1}^n$ ist abgeschlossen.

Beweis (des Lemmas): Durch vollständige Induktion über die Struktur von e:

- $e :: \beta$

  Trivial.

- $e = op :: \mathbf{int} \to \mathbf{int} \to \beta$

  $[\![\Gamma \rhd op :: \mathbf{int} \to \mathbf{int} \to \beta]\!]\eta = op^I_{curry}$ und $op[{}^{e_i}/_{id_i}]_{i=1}^n = op$.

  Noch zu zeigen: $op^I_{curry} \leq_{\mathbf{int}\to\mathbf{int}\to\beta} op$

  Sei $n_1 \in \mathbb{Z}$, zu zeigen: $op^I_{curry}\,n_1 \leq_{\mathbf{int}\to\beta} op\,n_1$.

  Da $op\,n_1 \in Val$, bedeutet dies

  $$op^I_{curry}\,n_1 \quad \leq_{\mathbf{int}\to\beta}^0 \quad op\,n_1.$$

  Sei $n_2 \in \mathbb{Z}$, dann bleibt zu zeigen: $op^I_{curry}\,n_1\,n_2 \leq_\beta op\,n_1\,n_2$

  Ok, da $op\,n_1\,n_2 \xrightarrow{(\text{Op})} op^I(n_1, n_2) = op^I_{curry}\,n_1\,n_2$. Also gilt $\leq_\beta^0$.

- $e = id$

  Wegen $\Gamma \triangleright id :: \tau$ muss $id = id_i$ und $\tau = \tau_i$ (für ein i) sein.

$$\begin{array}{rcl} [\![\Gamma \triangleright id_i :: \tau_i]\!]\eta & = & \eta(id_i) \\ id_i[{}^{e_j}/{}_{id_j}]_{j=1}^n & = & e_i, \text{ und } \eta(id_i) \leq_{\tau_i} e_i \text{ gilt nach Voraussetzung.} \end{array}$$

- $e = e'\,e''$

  $[\![\Gamma \triangleright e'\,e'' :: \tau]\!]\eta = ([\![\Gamma \triangleright e' :: \tau' \to \tau]\!]\eta)\,([\![\Gamma \triangleright e'' :: \tau']\!]\eta)$

  Nach Induktionsvoraussetzung gilt:

  (1) $[\![\Gamma \triangleright e' :: \tau' \to \tau]\!]\eta \leq_{\tau' \to \tau} e'[{}^{e_i}/{}_{id_i}]_{i=1}^n$

  (2) $[\![\Gamma \triangleright e'' :: \tau']\!]\eta \leq_{\tau'} e''[{}^{e_i}/{}_{id_i}]_{i=1}^n$

  (1) bedeutet nach (iii) der Definition von $\leq_{\tau' \to \tau}$: Es ex. ein Wert v mit

  (1a) $e'[{}^{e_i}/{}_{id_i}]_{i=1}^n \xrightarrow{*} v$

  (1b) $[\![\Gamma \triangleright e' :: \tau' \to \tau]\!]\eta \leq^0_{\tau' \to \tau} v$

  (2) bedeutet nach Teil (iii) der Definition: Es ex. ein Wert v' mit

  (2a) $e''[{}^{e_i}/{}_{id_i}]_{i=1}^n \xrightarrow{*} v'$

  (2b) $[\![\Gamma \triangleright e'' :: \tau']\!]\eta \leq^0_{\tau'} v'$

  Aus (1a) und (2a) folgt:

$$e[{}^{e_i}/{}_{id_i}]_{i=1}^n = (e'[{}^{e_i}/{}_{id_i}]_{i=1}^n)\,(e''[{}^{e_i}/{}_{id_i}]_{i=1}^n) \xrightarrow{*} v\,(e''[{}^{e_i}/{}_{id_i}]_{i=1}^n) \xrightarrow{*} v\,v'$$

  Aus (1b) und (2b) ergibt sich nach Teil (ii) der Definition:

$$([\![\Gamma \triangleright e' :: \tau' \to \tau]\!])\,([\![\Gamma \triangleright e'' :: \tau']\!]) \leq_\tau v\,v'$$

  Nach Teil (iii) der Definition ex. dann ein v'' mit $v\,v' \xrightarrow{*} v''$

  (3) $[\![\Gamma \triangleright e :: \tau]\!]\eta \leq^0_\tau v''$

  und es gilt:

  (4) $e[{}^{e_i}/{}_{id_i}]_{i=1}^n \xrightarrow{*} v\,v' \xrightarrow{*} v''$

  Aus (3) und (4) folgt: $[\![\Gamma \triangleright e :: \tau]\!]\eta \leq_\tau e[{}^{e_i}/{}_{id_i}]_{i=1}^n$.

- $e = \lambda id : \tau'.e'$

  Wenn $\Gamma \triangleright \lambda id : \tau'.e' :: \tau' \to \tau''$ und $\eta \models \Gamma$, dann gilt:

$$\Gamma[{}^{\tau'}/{}_{id}] \triangleright e' :: \tau''$$

  Zu zeigen: $[\![\Gamma \triangleright \lambda id : \tau'.e' :: \tau' \to \tau'']\!]\eta \leq_{\tau' \to \tau''} \underbrace{(\lambda id : \tau'.e')[{}^{e_i}/{}_{id_i}]_{i=1}^n}_{\text{also } \leq^0_{\tau' \to \tau''} \text{ zu zeigen}}$

  Seien $d \in [\![\tau']\!]$ und $v \in Val^{\tau'}$ mit $d \leq^0_{\tau'} v$. Noch zu zeigen:

$$\begin{array}{rcl} ([\![\Gamma \triangleright \lambda id : \tau'.e' :: \tau' \to \tau'']\!]\eta)\,(d) & \leq_{\tau''} & ((\lambda id : \tau'.e')[{}^{e_i}/{}_{id_i}]_{i=1}^n)\,v \\ ([\![\Gamma \triangleright \lambda id : \tau'.e' :: \tau' \to \tau'']\!]\eta)\,(d) & = & [\![\Gamma[{}^{\tau'}/{}_{id}] \triangleright e' :: \tau'']\!]\,\underbrace{\eta[{}^{d}/{}_{id}]}_{\eta'} \end{array}$$

O.B.d.A: $id \notin \{id_1, \ldots, id_n\}$. Dann gilt:

$$((\lambda id : \tau'.e')[^{e_i}/_{id_i}]_{i=1}^n)\, v = (\lambda id : \tau'.e'[^{e_i}/_{id_i}]_{i=1}^n)\, v \xrightarrow{\text{(Beta-V)}} e'[^{e_i}/_{id_i}]_{i=1}^n[^v/_{id}]$$

wegen $\{id_1, \ldots, id_n, id\} = dom\,(\Gamma[^{\tau'}/_{id}])$ und $\eta'(id_i) = \eta(id_i) \leq_{\tau_i} e_i$ und $\eta'(id) = d \leq_{\tau'} v$ gilt nach Induktionsannahme:

$$[\![\Gamma[^{\tau'}/_{id}] \rhd e' :: \tau'']\!]\eta' \quad \leq_{\tau''} \quad e'[^{e_i}/_{id_i}]_{i=1}^n[^v/_{id}] \qquad \square$$

Bewiesen: Für jeden Ausdruck $e \in Exp^\tau$ gilt: $[\![e :: \tau]\!] \leq_\tau e$. Per Definition von $\leq_\tau$ bedeutet das: Es ex. ein $v \in Val^\tau$ mit $e \xrightarrow{*} v$ und $[\![e :: \tau]\!] \leq_\tau^0 v$. Also insbesondere:

Satz 2.5 *Jeder abgeschlossene wohlgetypte Ausdruck in $\mathcal{L}_1^t$ (ohne exceptions) terminiert mit einem Wert.*

Für Basistyp β ergibt sich insbesondere: Es ex. ein $v \in Val^\beta$ mit $e \xrightarrow{*} v$ und $[\![e :: \beta]\!] = v$. Damit ist Satz 2.3 (Adäquatheit) bewiesen:

Für jeden Ausdruck $e \in Exp^\beta$ gilt: $e \xrightarrow{*} c \;\Leftrightarrow\; [\![e :: \beta]\!] = c$

2.1 Denotationelle Äquivalenz

Idee: Zwei Ausdrücke heissen *denotationell äquivalent*, wenn sie die gleiche denotationelle Semantik haben.
Genauer: Seien e, e' Ausdrücke mit $\Gamma \rhd e :: \tau$ und $\Gamma \rhd e' :: \tau$. e und e' heissen *denotationell äquivalent* bezüglich Γ, wenn gilt:

$$[\![\Gamma \rhd e :: \tau]\!] \quad = \quad [\![\Gamma \rhd e' :: \tau]\!]$$

(d.h. $[\![\Gamma \rhd e :: \tau]\!]\eta = [\![\Gamma \rhd e' :: \tau]\!]\eta$ für alle $\eta \in Env_\Gamma$).

Frage: Kann denotationelle Äquivalenz von Γ abhängen?

Leider ja!

Beispiel

(1) Seien $x, y \in Id$ verschieden.

$[\![\Gamma \rhd x = y :: \mathbf{bool}]\!] = [\![\Gamma \rhd \mathit{true} :: \mathbf{bool}]\!]$ *gilt, wenn* $\Gamma(x) = \Gamma(y) = \mathbf{unit}$ *(denn es kann nur* $\eta(x) = \eta(y) = (\,)$ *sein).*

Aber nicht, wenn $\Gamma(x) = \Gamma(y) = \mathbf{int}$ *(weil man dann* $\eta(x) \neq \eta(y)$ *wählen kann).*

(2) $[\![\Gamma \rhd f\,x :: \mathbf{int}]\!] = [\![\Gamma \rhd f\,y :: \mathbf{int}]\!]$
gilt, wenn $\Gamma(x) = \Gamma(y) = \mathbf{unit}$ *und* $\Gamma(f) = \mathbf{unit} \to \mathbf{int}$,
aber nicht, wenn $\Gamma(x) = \Gamma(y) = \mathbf{int}$ *und* $\Gamma(f) = \mathbf{int} \to \mathbf{int}$.

2.2 Operationelle Äquivalenz

Auch als *Beobachtungskongruenz* (engl.: *operational congruence*) bezeichnet.

Idee: Zwei Ausdrücke heissen *operationell äquivalent*, wenn sie sich in jedem Programm gegeneinander austauschen lassen, ohne dass sich das „beobachtbare Verhalten" des Programms verändert.

Definition 2.4

(a) Ein Programm *in $\mathcal{L}_1^t$ ist ein abgeschlossener Ausdruck vom Basistyp.*

(b) Das beobachtbare Verhalten *eines Programms ist der Wert, mit dem es terminiert.*

Noch zu definieren: „Austauschbarkeit" von Programmstücken.

Definition 2.5

(a) Ein Kontext $\mathrm{C}[\,]$ *ist ein „Ausdruck mit einem Loch", z.B.* $\lambda x : \mathbf{int}.x + [\,]$.

(b) Mit $\mathrm{C}[e]$ *bezeichnet man den Ausdruck, der aus* $\mathrm{C}[\,]$ *entsteht, indem man* e wörtlich *in das Loch einsetzt (ohne Umbenennung!).*

Beispiel *Sei* $\mathrm{C}[\,] = \lambda x : \mathbf{int}.x + [\,]$. *Dann ist*

$$\mathrm{C}[x + 2] = \lambda x : \mathbf{int}.x + (x + 2),$$

d.h. durch die Einsetzung können auch neue Bindungen enstehen.

Definition 2.6

(a) Ein Kontext $\mathrm{C}[\,]$ *heisst* (Γ, τ)-Programmkontext, *wenn für alle Ausdrücke e gilt: Wenn $\Gamma \rhd e :: \tau$, dann ist* $\mathrm{C}[e]$ *ein Programm.*

(b) Seien e, e' Ausdrücke mit $\Gamma \rhd e :: \tau$ und $\Gamma \rhd e' :: \tau$. e und e' heissen operationell äquivalent *bezüglich Γ, wenn für jeden (Γ, τ)-Programmkontext* $\mathrm{C}[\,]$ *gilt:*

> $\mathrm{C}[e]$ *und* $\mathrm{C}[e']$ *haben das gleiche beobachtbare Verhalten (d.h. sie terminieren mit dem gleichen Wert).*

Satz 2.6 (Korrektheit der denotationelle Semantik) *Seien e, e' Ausdrücke mit $\Gamma \rhd e :: \tau$ und $\Gamma \rhd e' :: \tau$. Dann gilt:*

> *Wenn e und e' denotationell äquivalent sind bezüglich Γ, dann sind sie auch operationell äquivalent bezüglich Γ.*

Beweis: Sei $[\![\Gamma \rhd e :: \tau]\!] = [\![\Gamma \rhd e' :: \tau]\!]$, und sei weiter $\mathrm{C}[\,]$ ein (Γ, τ)-Programmkontext. Dann gilt wegen der „Kompositionalität" (d.h. der induktiven Definition) der denotationellen Semantik:

$$[\![\mathrm{C}[e] :: \beta]\!] \quad = \quad [\![\mathrm{C}[e'] :: \beta]\!]$$

Also stimmt nach Satz 2.3 (Adäquatheit) auch das beobachtbare Verhalten von $\mathrm{C}[e]$ und $\mathrm{C}[e']$ überein. Damit ist gezeigt, dass e und e' auch operationell äquivalent bezüglich Γ sind. □

Nachtrag: „Kompositionalität“ der denotationellen Semantik bedeutet: Die denotationelle Semantik eines Ausdrucks ergibt sich aus der denotationellen Semantik seiner (unmittelbaren) Teilausdrücke. Das folgt unmittelbar aus der induktiven Definition der denotationellen Semantik.
(Genauer: Die denotationelle Semantik eines Typurteils ergibt sich aus der denotationellen Semantik seiner Prämissen.)

Anwendung von Satz 2.6: „Programmtransformation“ zur Vereinfachung oder Optimierung von Programmen.

Beispiel *Wenn $id \notin free\,(e)$, dann gilt (Beweis: Übung):*

$$[\![\Gamma \rhd e :: \tau \to \tau']\!] \quad = \quad [\![\Gamma \rhd \lambda id : \tau.e\; id :: \tau \to \tau']\!]$$

Das beweisst, dass man in jedem Programm (von $\mathcal{L}_1^t$, ohne exceptions) einen Ausdruck der Form

$$\lambda id : \tau.e\; id$$

mit $id \notin free\,(e)$, vereinfachen kann zu e (sog. η-Regel). Konkret:

$$\textbf{let}\; succ\,(x : \textbf{int}) = 1 + x\;\textbf{in}\;\ldots$$

steht für

$$\textbf{let}\; succ = \lambda x : \textbf{int}.+\;1\;x\;\textbf{in}\;\ldots$$

lässt sich vereinfachen zu

$$\textbf{let}\; succ = +\,1\;\textbf{in}\;\ldots$$

Warnung: In Sprachen mit exceptions oder Rekursion gilt nur die sog. η-value-Regel:

$$[\![\Gamma \rhd v :: \tau \to \tau']\!] \quad = \quad [\![\Gamma \rhd \lambda id : \tau.v\; id :: \tau \to \tau']\!]$$

Frage: Gilt auch die Umkehrung von Satz 2.6? D.h. sind zwei operationell äquivalente Ausdrücke auch denotationell äquivalent?
Für viele denotationelle Semantiken (z.B. $\mathcal{L}_2^t$ mit der üblichen denotationelle Semantik) gilt diese Umkehrung *nicht*! Für $\mathcal{L}_1^t$ ist es nicht bekannt.

Definition 2.7 *Eine (adäquate) denotationelle Semantik heisst* voll abstrakt *(engl.:* fully abstract*), wenn alle operationell äquivalenten Ausdrücke auch denotationell äquivalent sind.*

Warum *fully abstract*? Eine denotationelle Semantik, die diese Eigenschaft besitzt, „identifiziert“ alle Programmstücke, die sich identifizieren lassen, ohne die Korrektheit der Semantik aufzugeben. Sie macht also keine unnötigen Unterscheidungen. In diesem Sinne ist sie „so abstrakt wie möglich“.

Klassisches Resultat über full abstraction (Plotkin 1977)
Wenn man $\mathcal{L}_2^t$ um ein „paralleles oder“ erweitert, so ist das übliche denotationelle Modell „fully abstract“.

3 Denotationelle Semantik für $\mathcal{L}_2^t$

<u>Bisher:</u> Denotationelles Modell für $\mathcal{L}_1^t$ (ohne exceptions). Bereiche $[\![\tau \to \tau']\!]$ bestanden aus (allen) totalen Funktionen $f : [\![\tau]\!] \to [\![\tau']\!]$.

<u>Jetzt:</u> Denotationelles Modell für $\mathcal{L}_2^t$ (ohne exceptions). Um nicht terminierende Programme zu beschreiben, braucht man (im Prinzip) partielle Funktionen.

<u>Frage:</u> Wie ordnet man einem rekursiven Ausdruck, z.B.

$$\textbf{rec}\, \mathit{fact} : \textbf{int} \to \textbf{int}.\lambda x : \textbf{int}.\textbf{if}\, x = 0\, \textbf{then}\, 1\, \textbf{else}\, x * \mathit{fact}\, (x - 1)$$

die „richtige" partielle Funktion zu?

<u>1. Idee:</u> Man sucht eine partielle Funktion $f : [\![\textbf{int}]\!] \rightharpoonup [\![\textbf{int}]\!]$, die „Lösung" der Rekursionsgleichung ist, also im Beispiel Lösung der Gleichung:

$$\mathit{fact} \quad = \quad \lambda x : \textbf{int}.\textbf{if}\, x = 0\, \textbf{then}\, 1\, \textbf{else}\, x * \mathit{fact}\, (x - 1)$$

Welche Lösung hat diese Gleichung?

1. Lösung:

$$f : [\![\textbf{int}]\!] \rightharpoonup [\![\textbf{int}]\!]$$
$$f(n) = \begin{cases} n! & \text{falls } n \geq 0 \\ \mathit{undef} & \text{sonst} \end{cases}$$

„richtige" Lösung

2. Lösung:

$$f' : [\![\textbf{int}]\!] \rightharpoonup [\![\textbf{int}]\!]$$
$$f'(n) = \begin{cases} n! & \text{falls } n \geq 0 \\ 0 & \text{sonst} \end{cases}$$

„unsinnige" Lösung, denn für $n < 0$ gilt:

$$f'(n) \quad = \quad \textbf{if}\, n = 0\, \textbf{then}\, 1\, \textbf{else}\, f'\, (n - 1)$$

<u>Weiteres Beispiel:</u> $g = \lambda x : \textbf{int}.g\, x$
Jede partielle Funktion ist Lösung dieser Funktionsgleichung. Die „richtige" Lösung ist die leere (d.h. überall undefinierte) Funktion.

Wie findet man unter mehreren Lösungen die „richtige", d.h. die die der operationellen Semantik des **rec**-Ausdrucks entspricht?

Die „richtige" Lösung ist die, die den kleinstmöglichen Definitionsbereich hat, d.h. die *kleinste* Funktion (bezüglich $\subseteq$), die eine Lösung ist (bei Funktionen höherer Stufe braucht man eine kompliziertere Ordnung als „$\subseteq$").

<u>2. Idee:</u> Man „approximiert" die (gewünschte) Semantik eines rekursiven Ausdrucks wie folgt:

0. Approximation = leere Funktion

1. Approximation = die Funktion, die nur die Resultate liefert, die sich *ohne* rekursiven Aufruf ergeben, im Beispiel:

$$\begin{array}{ll} \mathit{fact}: & 0 \mapsto 1 \\ & \mathit{undef} \quad \text{sonst} \end{array}$$

2. Approximation = die Funktion, die nur die Resultate mit Rekursionstiefe ≤ 1 liefert, im Beispiel:

$$\begin{array}{ll} \mathit{fact}: & 0 \mapsto 1 \\ & 1 \mapsto 1 \\ & \mathit{undef} \quad \text{sonst} \end{array}$$

n. Approximation = die Funktion, die nur die Resultate mit Rekursionstiefe $< n$ liefert.

Man beachte: Die $(n+1)$-te Approximation ergibt sich, indem man die n-te Approximation in die rechte Seite der Funktionsgleichung einsetzt. Die gewünschte Gesamtfunktion ergibt sich als Supremum all dieser Approximationen.

Für Funktionen 1. Stufe ist das Supremum einfach die Vereinigung $f = \bigcup\limits_{n \in \mathbb{N}} f_n$, wobei f_n die n-te Approximation ist.

Für Funktionen höherer Stufe müss das "Supremum" noch definiert werden.

Wir werden sehen: Unter geeigneten Voraussetzungen ist das Supremum der Approximationen identisch mit der kleinsten Lösung der Funktionsgleichung.

Wir benötigen: Auf jedem semantischen Bereich eine Ordnungsrelation, die gewisse „gute Eigenschaften" hat, nämlich:

- es müssen gewisse Suprema existieren
- es müssen gewisse Gleichungen lösbar sein

$\leadsto$ führt zur *Bereichstheorie* (engl.: *domain theory*)

3.1 Vollständige partielle Ordnungen

Definition 3.1 (Partielle Ordnungen) *Eine* partielle Ordnung *(engl:* partial order*, kurz:* po*) ist ein Paar $(D, \sqsubseteq)$, wobei D eine Menge ist und $\sqsubseteq$ eine zweistellige Relation auf D mit:*

- *Reflexivität: $d \sqsubseteq d$ für alle $d \in D$.*
- *Antisymmetrie: Wenn $d_1 \sqsubseteq d_2$ und $d_2 \sqsubseteq d_1$, dann $d_1 = d_2$.*
- *Transitivität: Wenn $d_1 \sqsubseteq d_2$ und $d_2 \sqsubseteq d_3$, dann $d_1 \sqsubseteq d_3$.*

Definition 3.2 *Sei $(D, \sqsubseteq)$ eine po und $s \subseteq D$.*

(a) $d \in S$ heisst kleinstes Element *von S, wenn $d \sqsubseteq s$ für alle $s \in S$.*
(analog: grösstes Element*)*

(b) $d \in D$ heisst obere Schranke *(engl.:* upper bound*, kurz:* ub*) von S, wenn $s \sqsubseteq d$ für alle $s \in S$.*
(analog: untere Schranke*)*

(c) *$d \in D$ heisst* kleinste obere Schranke *(engl.:* least upper bound*, kurz:* lub*) oder* Supremum *von S, wenn d kleinstes Element der Menge aller oberer Schranken von S ist, d.h. wenn gilt:*
   - *d ist obere Schranke von S (also $s \sqsubseteq d$ für alle $s \in S$)*
   - *$d \sqsubseteq d'$ für jede (andere) obere Schranke d' von S*

   (analog: grösste untere Schranke*)*

Bezeichnungen *Sei $(D, \sqsubseteq)$ eine po.*

- *Wenn D selbst ein kleinstes Element besitzt, dann bezeichnet man dieses Element mit $\bot$ oder $\bot_D$ (lies: „bottom“)*
  (Analog: $\top$ oder $\top_D$ (lies: „top“) für das gröste Element)
- *Wenn das Supremum einer Menge $S \subseteq D$ existiert, so bezeichnet man es mit $\bigsqcup S$ oder $\bigsqcup_D S$. Wenn $S = \{d_1, \ldots, d_n\}$ endlich ist, so schreibt man auch $d_1 \sqcup \ldots \sqcup d_n$ statt $\bigsqcup S$.*
  (Analog: $\bigsqcap S$ für das Infimum)

Bemerkung *Sei $(D, \sqsubseteq)$ eine po und $S \subseteq D$.*

- *S besitzt höchstens ein kleinstes (grösstes) Element.*
- *S kann beliebig viele obere (untere) Schranken besitzen: Eine obere (untere) Schranke muss nicht in S liegen. Wenn sie in S liegt, dann ist sie grösstes (kleinstes) Element von S.*
- *S besitzt höchstens eine kleinste obere (grösste untere) Schranke. Auch diese muss nicht in S liegen. Wenn sie in S liegt, ist sie grösstes (kleinstes) Element von S.*
- *Jedes $d \in D$ ist obere (untere) Schranke der leeren Menge:*

$$\begin{array}{lcll} \bigsqcup \emptyset & = & \bot_D & \textit{(falls es existiert)} \\ \bigsqcap \emptyset & = & \top_D & \textit{(falls es existiert)} \end{array}$$

Definition 3.3 *Sei D eine partielle Ordnung:*

(a) *Eine Menge $\Delta \subseteq D$ heisst* gerichtet *(engl.:* directed*), wenn $\Delta \neq \emptyset$ und zu je zwei Elementen $d_1, d_2 \in \Delta$ existiert ein $d \in \Delta$ mit $d_1 \sqsubseteq d$ und $d_2 \sqsubseteq d$ (d.h. d ist obere Schranke von d_1 und d_2).*

(b) *D heisst* gerichtet vollständig *(engl.:* directed complete, dcpo = directed complete partial order*), wenn jede gerichtete Teilmenge von D ein Supremum besitzt.*

(c) *D heisst* vollständiger Verband*, wenn (sogar) jede Teilmenge $S \subseteq D$ ein Supremum besitzt.*

Bemerkung

(a) *Jede total geordnete Teilmenge $S \subseteq D$ ist gerichtet (S heisst total geordnet, wenn für alle $s_1, s_2 \in S$ gilt: $s_1 \sqsubseteq s_2$ oder $s_2 \sqsubseteq s_1$).*

(b) *Jede aufsteigende Folge $d_0 \sqsubseteq d_1 \sqsubseteq \ldots$ heisst gerichtete Teilmenge von D.*

Beispiel

(a) $(\mathbb{N}, \leq)$ *ist* keine *dcpo, denn $\mathbb{N}$ selbst ist gerichtet und besitzt* keine *obere Schranke.*
Analog: $(\mathbb{Z}, \leq)$, $(\mathbb{Q}, \leq)$, $(\mathbb{R}, \leq)$ sind keine *dcpo's.*

(b) $(\underbrace{[0,1]}_{\subseteq\mathbb{R}}, \leq)$ *ist eine dcpo, da jede Teilmenge ein Supremum besitzt.*

(c) $(\mathcal{P}(A), \subseteq)$ *ist eine dcpo (sogar ein vollständiger Verband).*

(d) $((A \rightharpoonup B), \sqsubseteq)$ *ist eine dcpo.*

Beweis: Sei $\Delta \subseteq (A \rightharpoonup B)$. Zu zeigen: Das Supremum $\bigsqcup \Delta$ existiert.

Betrachte $G = \bigcup\limits_{f\in\Delta} graph(f)$.

Wir zeigen: G ist Graph einer partiellen Funktion $g : A \rightharpoonup B$.

Seien $(a, b_1), (a, b_2) \in G$. Es genügt zu zeigen, dass $b_1 = b_2$ gilt.

Seien $f_1, f_2 \in A$ mit $(a, b_1) \in graph(f_1)$ und $(a, b_2) \in graph(f_2)$ (beide existieren nach Definition von G).

Da Δ gerichtet ist, existiert ein $f \in \Delta$ mit $f_1 \sqsubseteq f$ und $f_2 \sqsubseteq f$, also insbesondere $(a, b_1) \in graph(f)$ und $(a, b_2) \in graph(f)$, und damit $b_1 = b_2$.

Also ist G tatsächlich Graph einer Funktion g, und wegen $G = \bigcup\limits_{f\in\Delta} graph(f)$ gilt offensichtlich $g = \bigsqcup \Delta$. □

Satz 3.1 *Sei $(D, \sqsubseteq)$ eine partielle Ordnung, in der keine unendlichen aufsteigenden Folgen $d_1 \sqsubset d_2 \sqsubset \ldots$ existieren. Dann ist D eine dcpo, in der jede gerichtete Menge $\Delta \subseteq D$ (sogar) ein grösstes Element besitzt.*

Beweis: Angenommen, $\Delta \subseteq D$ ist gerichtet und besitzt *kein* grösstes Element, d.h. zu jedem Element $d \in \Delta$ existiert ein $d' \in \Delta$ mit $d' \not\sqsubseteq d$. Da Δ gerichtet ist, ex. ein $e \in \Delta$ mit $d \sqsubset e$. Damit ist gezeigt: Zu jedem $d \in \Delta$ ex. ein $e \in \Delta$ mit $d \sqsubset e$.
$\rightsquigarrow$ Durch Induktion erhält man, eine echt aufsteigende Folge $d_0 \sqsubset d_1 \sqsubset \ldots$, die sogar in Δ liegt. Widerspruch! □

Beispiel *Jede endliche partielle Ordnung ist eine dcpo (da sie keine unendlichen aufsteigenden Folgen besitzt).*

Definition 3.4 *Sei D eine Menge. Dann ist $(D, =)$ eine dcpo, die wir als* diskrete dcpo *bezeichnen.*

Definition 3.5 *Sei $(D, \sqsubseteq_D)$ eine partielle Ordnung, und sei $D_\perp = D \cup \{\perp\}$, wobei $\perp$ ein neues Element ist (d.h. $\perp \notin D$). Dann definieren wir auf $D_\perp$ eine Relation $\sqsubseteq_{D_\perp}$ durch:*

$$d \sqsubseteq_{D_\perp} e \quad \Leftrightarrow \quad d = \perp \;\vee\; (d, e \in D \wedge (d, e) \in \sqsubseteq_D)$$

$(D_\perp, \sqsubseteq_{D_\perp})$ ist wieder eine partielle Ordnung, die man als die zu D gehörige geliftete *partielle Ordnung bezeichnet. $\perp$ ist das kleinste Element von $D_\perp$.*

Satz 3.2 *Sei $(D, \sqsubseteq)$ eine partielle Ordnung und sei $S \subseteq D_\perp$, $S \neq \{\perp\}$. Dann gilt:*

(a) *Wenn S in $D_\perp$ gerichtet ist, dann ist $S \setminus \{\perp\}$ in D gerichtet.*

(b) *d ist genau dann obere Schranke von S in $D_\perp$, wenn es obere Schranke von $S \setminus \{\perp\}$ in D ist.*

(c) *Wenn $d = \bigsqcup_D (S \setminus \{\perp\})$ existiert, dann ist auch $d = \bigsqcup_{D_\perp} S$.*

(d) *Wenn D eine dcpo ist, dann ist auch $D_\perp$ eine dcpo.*

Beweis: nur (d)
Sei D dcpo und $\Delta \subseteq D_\perp$ gerichtet in $D_\perp$. Wenn $\Delta = \{\perp\}$, dann ist $\bigsqcup \Delta = \perp$. Wenn $\Delta \neq \{\perp\}$, dann gilt nach (a): $\Delta \setminus \{\perp\}$ ist gerichtet in D. Also ex. nach Voraussetzung $\bigsqcup_D (\Delta \setminus \{\perp\})$ und nach (c) ist dies auch Supremum von Δ in $D_\perp$. □

Spezialfall: Wenn D eine diskrete po ist, dann bezeichnet man $D_\perp$ als die zu D gehörige *flache* po. Jede flache po ist nach Satz 3.1 oder Satz 3.2 eine dcpo.

Definition 3.6 *Seien $(D, \sqsubseteq_D)$ und $(E, \sqsubseteq_E)$ zwei po's. Dann definiert man auf $D \times E$ eine Relation $\sqsubseteq_{D\times E}$ durch:*

$$(d_1, e_1) \sqsubseteq_{D\times E} (d_2, e_2) \quad \Leftrightarrow \quad d_1 \sqsubseteq_D d_2 \ \wedge \ e_1 \sqsubseteq_E e_2$$

Es ist leicht zu sehen, dass $(D \times E, \sqsubseteq_{D\times E})$ eine po ist. Man bezeichnet sie als *Produkt* der po's D und E. Die Projektionsfunktionen bezeichnen wir mit:

$$pr_1 : D \times E \to D \qquad pr_2 : D \times E \to E$$

Satz 3.3 *Seien $(D, \sqsubseteq_D), (E, \sqsubseteq_E)$ po's und sei $S \subseteq D \times E$. Dann gilt:*

(a) Wenn S gerichtet ist, dann sind auch $pr_1(S)$ und $pr_2(S)$ gerichtet.

(b) d ist obere Schranke von S gdw. $pr_1(d)$ obere Schranke von $pr_1(S)$ und $pr_2(d)$ obere Schranke von $pr_2(S)$ ist.

(c) Wenn $\bigsqcup_D pr_1(S)$ und $\bigsqcup_E pr_2(S)$ existiert, dann ex. auch $\bigsqcup_{D\times E} S$ und es gilt:

$$\bigsqcup_{D\times E} \quad = \quad (\bigsqcup_D pr_1(S), \bigsqcup_E pr_2(S))$$

(d.h. das Supremum in $D \times E$ wird komponentenweise gebildet)

(d) Wenn D und E dcpo's sind, dann ist auch $D \times E$ eine dcpo.

(e) Wenn $\perp_D$ und $\perp_E$ existieren, dann ist $\perp_{D\times E} = (\perp_D, \perp_E)$ kleinstes Element von $D \times E$.

Beweis: Übung □

3.2 Stetige Funktionen

Definition 3.7

(a) Seien D, E po's. Eine (totale) Funktion $f : D \to E$ heisst monoton*, wenn sie mit „$\sqsubseteq$" verträglich ist, d.h. wenn für alle $d_1, d_2 \in D$ gilt:*

$$d_1 \sqsubseteq d_2 \quad \Rightarrow \quad f(d_1) \sqsubseteq f(d_2)$$

(b) Seien D, E dcpo's. Eine monotone Funktion $f : D \to E$ heisst stetig*, wenn sie mit den Suprema gerichteter Mengen verträglich ist, d.h. wenn für jede gerichtete Menge $\Delta \subseteq D$ gilt:*

$$f(\bigsqcup_D \Delta) = \bigsqcup_E f(\Delta)$$

Beachte: „$\sqsupseteq$“ folgt schon aus der Monotonie, denn: $\bigsqcup_D \Delta$ ist obere Schranke von Δ, also ist $f(\bigsqcup_D \Delta)$ obere Schranke von $f(\Delta)$, und damit $f(\bigsqcup_D \Delta) \sqsupseteq \bigsqcup_E f(\Delta)$.

Also: Um zu zeigen, dass eine monotone Funktion f stetig ist, genügt es

$$f(\textstyle\bigsqcup_D \Delta) \quad \sqsubseteq \quad \textstyle\bigsqcup_E f(\Delta)$$

zu zeigen.

Beispiel (einer monotonen Funktion, die *nicht* stetig ist) *Sei* $((\mathbb{N} \rightharpoonup \mathbb{N}), \sqsubseteq)$ *die dcpo der partiellen Funktionen von* $\mathbb{N}$ *nach* $\mathbb{N}$. *Für jedes* $n \in \mathbb{N}$ *sei* $f_n : \mathbb{N} \rightharpoonup \mathbb{N}$ *definiert durch:*

$$f_n(i) \quad = \quad \begin{cases} i & \text{falls } 0 \leq i < n \\ \mathit{undef} & \text{sonst} \end{cases}$$

Dann gilt: $f_0 \sqsubseteq f_1 \sqsubseteq \ldots$

Keine der Funktionen f_i *ist total, aber* $\bigsqcup\limits_{n \in \mathbb{N}} f_n$ *ist die Identität auf* $\mathbb{N}$, *also eine totale Funktion.*

Sei

$$\begin{aligned} T : \quad & (\mathbb{N} \rightharpoonup \mathbb{N}) \to \mathit{Bool}_\perp \\ & f \mapsto \begin{cases} \mathit{true} & \text{falls } f \text{ total} \\ \perp & \text{sonst} \end{cases} \end{aligned}$$

also der „Test auf Totalität“.

T *ist monoton, denn: Wenn* $f \sqsubseteq g$ *in* $(\mathbb{N} \rightharpoonup \mathbb{N})$, *dann gilt entweder* $f = g$ *oder* f *ist nicht total. Im ersten Fall ist* $T(f) = T(g)$, *im zweiten Fall ist* $T(f) = \perp \sqsubseteq T(g)$, *also Monotonie erfüllt.*

T *ist* nicht *stetig, denn: Sei* $\Delta = \{f_n \mid n \in \mathbb{N}\}$. Δ *ist gerichtet und enthält nur echt partielle Funktionen, also ist* $T(\Delta) = \{\perp\}$. *Andererseits ist* $\bigsqcup \Delta = id_{\mathbb{N}}$ *eine totale Funktion, also*

$$T(\textstyle\bigsqcup \Delta) = \mathit{true} \neq \perp = \textstyle\bigsqcup\{\perp\} = \textstyle\bigsqcup T(\Delta).$$

Unstetigkeit!

Beispiel (für stetige Funktionen) *Seien* D, E *dcpo's. Dann gilt:*

(a) *Die Identitätsfunktion* $id_D : D \to D, d \mapsto d$ *ist stetig.*

(b) *Für jedes* $e \in E$ *ist die konstante Funktion* $const_e : D \to E, d \mapsto e$ *stetig.*

(c) *Die Projektionsfunktionen* $pr_1 : D \times E \to D$ *und* $pr_2 : D \times E \to E$ *sind stetig.*

Beweis: (a), (b) klar, (c) folgt aus Satz 3.3. □

Satz 3.4 *Seien* B, C, D, E *dcpo's.*

(a) *Wenn* $f : C \to D$ *und* $g : D \to E$ *stetig, dann ist* $f \circ g : C \to E$ *stetig.*

(b) *Wenn* $f : C \to D$ *und* $g : C \to E$ *stetig, dann ist* $\langle f, g \rangle : C \to E, c \mapsto (fc, gc)$ *stetig.*

(c) Wenn $f : B \to D$ und $g : C \to E$ stetig sind, dann ist auch

$$\begin{array}{rl} f \times g : & B \times C \to D \times E \\ & (b, c) \mapsto (f\,b, g\,c) \end{array}$$

stetig.

(d) $f : C \to D \times E$ ist stetig gdw. $pr_1 \circ f : C \to D$ und $pr_2 \circ f : C \to E$ stetig sind.

Beweis: nur (a): Da f und g monoton sind, ist auch $g \circ f$ monoton. Sei $\Delta \subseteq C$ gerichtet. Dann gilt:

$$(g \circ f)(\bigsqcup \Delta) = g(f(\bigsqcup \Delta)) = g(\bigsqcup f(\Delta)) = \bigsqcup (g \circ f)(\Delta)$$ □

Satz 3.5 (Fixpunktsatz von Kleene) *Sei D eine dcpo, die ein kleinstes Element $\bot_D$ besitzt, und $f : D \to D$ stetig. Dann gilt:*

(a) Die Menge $\{f^n(\bot_D) \mid n \in \mathbb{N}\}$ ist gerichtet (genauer: $\bot_D = f^0(\bot_D) \sqsubseteq f(\bot_D) \sqsubseteq \ldots$).

(b) $\mu f = \bigsqcup\{f^n(\bot_D) \mid n \in \mathbb{N}\}$ ist kleinster Fixpunkt von f.

Beweis:

(a) $\bot_D \sqsubseteq f(\bot_D)$, da $\bot_D$ kleinstes Element ist. Wegen der Monotonie von f folgt daraus

$$f(\bot_D) \sqsubseteq f^2(\bot_D),$$

also durch Induktion: $f^n(\bot_D) \sqsubseteq f^{n+1}(\bot_D)$ für alle n.

(b) Zunächst zeigen wir, dass μf ein Fixpunkt von f ist:

$$\begin{array}{rcll} f(\mu f) & = & f(\bigsqcup_{n\in\mathbb{N}} f^n(\bot_D)) & \\ & = & \bigsqcup_{n\in\mathbb{N}} f(f^n(\bot_D)) & \\ & = & \bigsqcup_{n\in\mathbb{N}} f^{n+1}(\bot_D) & \\ & = & \bigsqcup_{n\in\mathbb{N}} f^n(\bot_D) & \text{da } f^0(\bot_D) \text{ nichts zum Supremum beiträgt} \\ & = & \mu f & \end{array}$$

Bleibt noch zu zeigen, dass μf kleinster Fixpunkt ist.

Also zu zeigen: für jeden Fixpunkt d von f gilt: $\mu f \sqsubseteq d$.

Sei d Fixpunkt von f, d.h. $f(d) = d$. Da $\mu f = \bigsqcup_{n\in\mathbb{N}} f^n(\bot_D)$ kleinste obere Schranke aller $f^n(\bot_D)$ ist, genügt es zu zeigen: d ist obere Schranke, also $f^n(\bot_D) \sqsubseteq d$ für alle n:

- $n = 0$

  $f^0(\bot_D) = \bot_D \sqsubseteq d$

- $n \rightsquigarrow n + 1$

  $f^{n+1}(\bot_D) = f(f^n(\bot_D))$ und $f^n(\bot_D) \sqsubseteq d$ nach Induktionsannahme. Daraus folgt

  $$f^{n+1}(\bot_D) \sqsubseteq f(d) = d,$$

  da f monoton ist und d ein Fixpunkt von f ist. □

3.3 Weitere Ergebnisse über stetige Funktionen

Satz 3.6 *Sei D eine dcpo, in der keine unendlichen aufsteigenden Folgenden existieren und sei E eine beliebige dcpo. Dann ist jede monotone Funktion $f : D \to E$ bereits stetig.*

Beweis: Sei $\Delta \subseteq D$ gerichtet und f monoton. Dann ist $\bigsqcup \Delta$ sogar grösstes Element von Δ (vgl. Satz 3.1). Also ist $f(\bigsqcup \Delta)$ grösstes Element von $f(\Delta)$, wegen Monotonie und damit erst recht Supremum von $f(\Delta)$. □

Definition 3.8 *Seien C, D, E dcpo's. Eine monotone Funktion $f : C \times D \to E$ heisst stetig im ersten Argument, wenn für jede gerichtete Menge $\Delta \subseteq C$ und jedes $d \in D$ gilt:*

$$\begin{aligned} f(\textstyle\bigsqcup \Delta, d) &= \textstyle\bigsqcup \{f(c,d) \mid c \in \Delta\} \\ &= \textstyle\bigsqcup f(\Delta \times \{d\}) \end{aligned}$$

<u>Mit anderen Worten:</u>
f ist stetig im ersten Argument, wenn für jedes $d \in D$ die Funktion

$$\begin{aligned} f_d : \ & C \to E \\ & c \mapsto f(c,d) \end{aligned}$$

stetig ist.

<u>Analog:</u> Stetigkeit im zweiten Argument. Verallgemeinerung auf mehr als zwei Argumente.

Satz 3.7 *Seien C, D, E dcpo's. Dann gilt:*

Eine monotone Funktion $f : C \times D \to E$ ist genau dann stetig, wenn sie in jedem Argument stetig ist.

Beweis:

„⇒“ Sei f stetig und $\Delta \subseteq C$ gerichtet. Dann ist auch $\Delta \times \{d\}$ gerichtet und es gilt (nach Satz 3.3):

$$\begin{aligned} \textstyle\bigsqcup(\Delta \times \{d\}) &= (\textstyle\bigsqcup \Delta, \bigsqcup\{d\}) \\ &= (\textstyle\bigsqcup \Delta, d) \end{aligned}$$

Also folgt:

$$\begin{aligned} f(\textstyle\bigsqcup \Delta, d) &= f(\textstyle\bigsqcup(\Delta \times \{d\})) \\ &= \textstyle\bigsqcup f(\Delta \times \{d\}) \end{aligned}$$

Also ist f stetig im ersten Argument, und die Stetigkeit im zweiten Argument folgt analog.

„⇐“ Sei f stetig in beiden Argumenten und sei $\Delta \subseteq C \times D$ gerichtet. Dann gilt:

$$\begin{aligned} f(\textstyle\bigsqcup \Delta) &= f(\textstyle\bigsqcup pr_1(\Delta), \bigsqcup pr_2(\Delta)) \\ &= \textstyle\bigsqcup_{c \in pr_1(\Delta)} f(c, \bigsqcup pr_2(\Delta)) \\ &= \textstyle\bigsqcup_{c \in pr_1(\Delta)} (\bigsqcup_{d \in pr_2(\Delta)} f(c,d)) \\ &= \textstyle\bigsqcup_{c \in pr_1(\Delta), d \in pr_2(\Delta)} f(c,d) \\ &= \textstyle\bigsqcup_{(c,d) \in pr_1(\Delta) \times pr_2(\Delta)} f(c,d) \\ &= \textstyle\bigsqcup f(pr_1(\Delta) \times pr_2(\Delta)) \\ &= \textstyle\bigsqcup f(\Delta) \end{aligned}$$

□

3.4 Funktionenräume

Definition 3.9 *Seien D, E dcpo's. Mit $[D \to E]$ bezeichnen wir die Menge aller* stetigen *Funktionen von D nach E. Auf $[D \to E]$ wird eine Relation $\sqsubseteq_{[D\to E]}$ definiert durch:*

$$f \sqsubseteq_{[D\to E]} g \quad \Leftrightarrow \quad \text{für alle } d \in D \text{ gilt: } f(d) \sqsubseteq_E g(d)$$

$([D \to E], \sqsubseteq_{[D\to E]})$ ist eine partielle Ordnung, die man als Funktionenraum *von D nach E oder* Exponent *von E und D (Schreibweise: E^D) bezeichnet.*

Satz 3.8 *Seien D, E dcpo's und sei $S \subseteq [D \to E]$. Dann gilt:*

(a) Wenn S gerichtet ist, dann ist auch $S\,d = \{f\,d \mid f \in S\}$ gerichtet für jedes $d \in D$.

(b) g ist obere Schranke von S, wenn $g\,d$ obere Schranke von $S\,d$ ist für jedes $d \in D$.

(c) Wenn $\bigsqcup_E S\,d$ für jedes $d \in D$ existiert, dann existiert auch $\bigsqcup_{[D\to E]} S$ und es gilt:

$$(\bigsqcup_{[D\to E]} S)\,d \quad = \quad \bigsqcup_E S\,d \text{ für alle } d \in D$$

(d.h. das Supremum wird „argumentweise“ gebildet)

(d) $([D \to E], \sqsubseteq_{[D\to E]})$ ist eine dcpo.

(e) Wenn E ein kleinstes Element $\bot_E$ besitzt, dann ist die Funktion

$$\begin{aligned} \bot_{[D\to E]} : \quad & D \to E \\ & d \to \bot_E \end{aligned}$$

kleinstes Element von $[D \to E]$.

Beweis:

(a) Sei S gerichtet, zu zeigen $S\,d$ gerichtet. Seien $e_1, e_2 \in S\,d$, d.h. es ex. $f_1, f_2 \in S$ mit $e_1 = f_1\,d$ und $e_2 = f_2\,d$. Da S gerichtet existiert ein $f \in S$ mit $f_1, f_2 \sqsubseteq f$ also $e_1 = f_1\,d \sqsubseteq f\,d \in S\,d$ und $e_2 = f_2\,d \sqsubseteq f\,d \in S\,d$, d.h. $f\,d$ ist gemeinsame obere Schranke von e_1 und e_2.

(b) Trivial, da $\sqsubseteq_{[D\to E]}$ argumentweise definiert ist.

(c) Wenn $\bigsqcup_E S\,d$ für jedes $d \in D$ existiert, so definieren wir

$$\begin{aligned} g : \quad & D \to E \\ & d \mapsto \bigsqcup_E S\,d. \end{aligned}$$

Wenn wir zeigen können, dass g stetig ist, dann folgt leicht, dass $g = \bigsqcup S$, denn:

- obere Schranke g, gilt wegen (b)
- wenn h obere Schranke von S ist, dann ist $h\,d$ obere Schranke von $S\,d$ für jedes d. Also ist $g\,d \sqsubseteq h\,d$ für jedes $d \in D$, da $g\,d$ *kleinste* obere Schranke ist. Also ist $g \sqsubseteq h$.

Es bleibt noch zu zeigen: Wenn D, E dcpo's, $S \subseteq [D \to E]$ gerichtet und $g : D \to E$ definiert durch

$$g\,d \quad = \quad \bigsqcup S\,d,$$

dann ist g stetig (also $g \in [D \to E]$).

Dazu zeigen wir: Für jede gerichtete Menge $\Delta \subseteq D$ gilt: $g(\bigsqcup \Delta) = \bigsqcup g(\Delta)$.

$$\begin{array}{rcll}
g(\bigsqcup \Delta) & = & \bigsqcup S(\bigsqcup \Delta) & \\
& = & \bigsqcup\{f(\bigsqcup \Delta) \mid f \in S\} & \\
& = & \bigsqcup\{\bigsqcup f(\Delta) \mid f \in S\} & \text{denn jedes } f \in S \text{ ist stetig} \\
& = & \bigsqcup\{\bigsqcup\{f(d) \mid d \in \Delta\} \mid f \in S\} & \\
& = & \bigsqcup\{f(d) \mid d \in \Delta, f \in S\} & \text{nach Lemma über Zeilensuprema}
\end{array}$$

$$\begin{array}{rcll}
\bigsqcup g(\Delta) & = & \bigsqcup\{g(d) \mid d \in \Delta\} & \\
& = & \bigsqcup\{\bigsqcup S\,d \mid d \in \Delta\} & \\
& = & \bigsqcup\{\bigsqcup\{f(d) \mid f \in S\} \mid d \in \Delta\} & \\
& = & \bigsqcup\{f(d) \mid d \in \Delta, f \in S\} & \text{nach Lemma über Spaltensuprema}
\end{array}$$

(d) Sei $\Delta \subseteq [D \to E]$ gerichtet. Dann ist wegen (a) auch $\Delta\,d$ gerichtet für jedes $d \in D$, also existiert $\bigsqcup_E \Delta\,d$ für jedes $d \in D$. Also folgt mit (c) die Existenz von $\bigsqcup \Delta$.

(e) Trivial. □

Beispiel (für stetige Funktionen) *Seien C, D, E dcpo's.*

(a) Der Applikationsoperator

$$\begin{array}{l}
\mathit{apply} : [D \to E] \times D \to E \\
\mathit{apply}\,(f, d) = f\,d
\end{array}$$

ist stetig.

(b) Wenn $f : C \times D \to E$ stetig, dann ist auch

$$\begin{array}{l}
\mathit{curry}(f) : C \to [D \to E] \\
\mathit{curry}(f)\,c\,d = f(c, d)
\end{array}$$

wohldefiniert und stetig.

(c) Wenn D ein kleinstes Element $\perp$ besitzt, dann ist der Fixpunktoperator

$$\begin{array}{l}
\mu : [D \to D] \to D \\
\mu(f) = \bigsqcup\{f^n(\perp) \mid n \in \mathbb{N}\}
\end{array}$$

stetig.

Beweis:

(a) *apply* ist stetig in beiden Argumenten, denn:

<u>1.Arg:</u> Sei $\Delta \subseteq [D \to E]$ gerichtet. Dann gilt

$$\begin{array}{rcll} apply(\bigsqcup \Delta, d) & = & (\bigsqcup \Delta)\, d & \\ & = & \bigsqcup(\Delta\, d) & \text{nach Definition von } \bigsqcup \Delta \\ & = & \bigsqcup\{f\, d \mid f \in \Delta\} & \\ & = & \bigsqcup\{apply(f,d) \mid f \in \Delta\} & \text{nach Def. von } apply \end{array}$$

<u>2.Arg:</u> Sei $\Delta \subseteq D$ gerichtet.

$$\begin{array}{rcll} apply(f, \bigsqcup \Delta) & = & f(\bigsqcup \Delta) & \\ & = & \bigsqcup f(\Delta) & \text{da } f \text{ stetig} \\ & = & \bigsqcup\{f(d) \mid d \in \Delta\} & \\ & = & \bigsqcup\{apply(f,d) \mid d \in \Delta\} & \end{array}$$

(b) Wohldefiniertheit: Zu zeigen ist: $curry(f)\, c$ ist stetig für jedes $c \in C$. Sei also $c \in C$ und $\Delta \subseteq D$ gerichtet.

$$\begin{array}{rcll} curry(f)\, c\, (\bigsqcup \Delta) & = & f(c, \bigsqcup \Delta) & \text{nach Def. von } curry \\ & = & \bigsqcup\{f(c,d) \mid d \in \Delta\} & \text{da } f \text{ stetig im 2.Arg.} \\ & = & \bigsqcup\{curry(f)\, c\, d \mid d \in \Delta\} & \text{nach Def. von } curry \\ & = & \bigsqcup curry(f)\, c\, (\Delta) & \end{array}$$

Stetigkeit von $curry(f)$: Sei $\Delta \subseteq C$ gerichtet. Zu zeigen: $curry(f)\, (\bigsqcup \Delta) = \bigsqcup curry(f)\, (\Delta)$

Also zu zeigen: Für jedes $d \in \Delta$ gilt:

$$curry(f)\, (\bigsqcup \Delta)(d) \quad = \quad (\bigsqcup curry(f)\, (\Delta))\, d$$

Dazu:

$$\begin{array}{rcll} curry(f)\, (\bigsqcup \Delta)\, d & = & f(\bigsqcup \Delta, d) & \text{nach Def. von } curry \\ & = & \bigsqcup\{f(c,d) \mid c \in \Delta\} & \text{da } f \text{ stetig im 1.Arg.} \\ & = & \bigsqcup\{curry(f)\, c\, d \mid c \in C\} & \\ & = & (\bigsqcup curry(f)\, (\Delta))\, d & \text{da } \bigsqcup \text{ auf Fktn. argumentweise def.} \end{array}$$

(c) Es genügt zu zeigen: μ ist „punktweises Supremum" von stetigen Funktionen. Für jedes $n \in \mathbb{N}$ sei

$$\begin{array}{rl} \mu_n : & [D \to D] \to D \\ & f \mapsto f^n(\bot) \end{array}$$

Dann gilt $\underbrace{\mu(f) = \bigsqcup\{\mu_n(f) \mid n \in \mathbb{N}\}}_{\text{d.h. } \mu \text{ punktweises Supremum der } \mu_n}$

Bleibt zu zeigen: Jedes μ_n stetig. Vollständige Induktion über n:

$$\mu_0(f) \quad = \quad f^0(\bot) \quad = \quad \bot \text{ für alle } f,$$

also μ_0 stetig, da konstant.

$$\begin{array}{rcl} \mu_{n+1}(f) & = & f^{n+1}(\bot) \\ & = & f(f^n(\bot)) \\ & = & f(\mu_n\, f) \\ & = & apply(f, \mu_n\, f) \\ & = & apply(id\, f, \mu_n\, f) \\ & = & (apply \circ \langle id, \mu_n \rangle)\, (f) \end{array}$$

Also $\mu_{n+1} = (apply \circ \langle id, \mu_n \rangle)$, da id stetig und μ_n stetig nach Induktionsannahme, ist auch μ_{n+1} nach Lemma stetig. $\square$

3.5 Definition der denotationellen Semantik

Definition 3.10 (Semantische Bereiche) *Jedem Typ τ wird eine dcpo $[\![\tau]\!]$ zugeordnet durch Induktion über τ:*

$$\begin{array}{rcl} [\![\mathbf{int}]\!] & = & \mathbb{Z} \textit{ mit diskreter Ordnung} \\ [\![\mathbf{bool}]\!] & = & \textit{Bool mit diskreter Ordnung} \\ [\![\mathbf{unit}]\!] & = & \{()\} \textit{ mit diskreter Ordnung} \\ [\![\tau_1 \to \tau_2]\!] & = & [[\![\tau_1]\!] \to [\![\tau_2]\!]_\perp] \\ & & \textit{(dcpo der stetigen Fktn. von } [\![\tau_1]\!] \to [\![\tau_2]\!]_\perp\textit{)} \end{array}$$

<u>Beachte:</u>

- $[\![\tau_1 \to \tau_2]\!]$ hat stets ein kleinstes Element, nämlich

$$\perp_{[\![\tau_1 \to \tau_2]\!]} \quad = \quad \text{die konstante Funktion mit Resultat } \perp \in [\![\tau_2]\!]_\perp$$

- Insbesondere: Wenn τ_2 schon Funktionstyp ist, so muss man unterscheiden zwischen $\perp_{[\![\tau_2]\!]}$ und dem zušatzlichen $\perp$-Element von $[\![\tau_2]\!]_\perp$.

Beispiel *Der semantische Bereich $[\![\mathbf{int} \to \mathbf{int}]\!]$ hat $\perp_{\mathbf{int}\to\mathbf{int}} : \mathbb{Z} \to \mathbb{Z}_\perp, n \mapsto \perp$ als kleinstes Element. $[\![\mathbf{int} \to \mathbf{int}]\!]_\perp$ hat ein zusätzliches Element, das echt unter $\perp_{[\![\mathbf{int}\to\mathbf{int}]\!]}$ liegt.*

Das ist sinnvoll, denn $\perp_{[\![\mathbf{int}\to\mathbf{int}]\!]}$ dient als Semantik eines Wertes $v :: \mathbf{int} \to \mathbf{int}$, der bei jedem Aufruf divergiert. Das neue $\perp$-Element dient als Semantik eines Ausdrucks $e :: \mathbf{int} \to \mathbf{int}$, der schon selbst *divergiert.*

*Da v und e nicht in jedem Programmkontext gegeneinander austauschbar sind (*nicht beobachtungskongruent*), müssen wir ihnen unterschiedliche Semantiken zuordnen.*

Beispiel eines solchen Programmkontexts:

$$(\lambda x : \mathbf{int} \to \mathbf{int}.0)\,[\_]$$

Einsetzen von $e \leadsto$ Divergenz (wg. call-by-value)
Einsetzen von $v \leadsto$ Resultat 0

Definition 3.11 *Eine Umgebung η ist definiert als partielle Funktion $\eta : Id \rightharpoonup \bigcup_{\tau \in Type} [\![\tau]\!]$.*

- *Env = Menge aller Umgebungen.*
- *Env_Γ = Menge aller zu Γ passenden Umgebungen η, definiert durch:*

$$\begin{array}{rcl} \eta \models \Gamma & \Leftrightarrow & dom\,(\eta) = dom\,(\Gamma) \\ & \wedge & \forall id \in dom\,(\eta) : \eta(id) \in [\![\Gamma(id)]\!] \end{array}$$

Die Mengen Env_Γ betrachten wir als dcpo's mit folgender Ordnung:

$$\eta \sqsubseteq \eta' \quad \Leftrightarrow \quad \eta(id) \sqsubseteq \eta'(id) \textit{ für alle } id \in dom\,(\eta)$$

Definition 3.12 (der Semantikfunktion) *Jeder Konstanten $c :: \tau$ wird eine* Bedeutung $[\![c]\!] \in [\![\tau]\!]$ *zugeordnet. $[\![c]\!]$ ist wie früher definiert (abgesehen davon, dass wir neue semantische Bereich $[\![\tau]\!]$ haben):*

$$\begin{aligned} [\![n]\!] = n &\in \mathbb{Z} = [\![\mathbf{int}]\!] \\ [\![b]\!] = b &\in \mathit{Bool} = [\![\mathbf{bool}]\!] \\ [\![()]\!] = () &\in \mathit{Unit} = [\![\mathbf{unit}]\!] \end{aligned}$$

Sei $op :: \mathbf{int} \to \mathbf{int} \to \beta$. Dann sei $[\![op]\!] = \mathit{curry}(op^I) \in [\![\mathbf{int} \to \mathbf{int} \to \beta]\!]$. Es gilt

$$op^I : [\![\mathbf{int}]\!] \times [\![\mathbf{int}]\!] \to [\![\beta]\!]$$

stetig, da $[\![\mathbf{int}]\!] \times [\![\mathbf{int}]\!]$ diskrete po.

Jedem gültigen Typurteil $\Gamma \rhd e :: \tau$ wird wieder eine Funktion $[\![\Gamma \rhd e :: \tau]\!] : Env_\Gamma \to [\![\tau]\!]_\bot$ zugeordnet durch folgende Induktion über e (bzw. über die Länge der Herleitung des Typurteils):

(i) $[\![\Gamma \rhd c :: \tau]\!]\eta = [\![c]\!]$ *(liegt in $[\![\tau]\!] \subseteq [\![\tau]\!]_\bot$, da $c :: \tau$)*

(ii) $[\![\Gamma \rhd id :: \tau]\!]\eta = \eta(id)$ *(liegt in $[\![\Gamma(id)]\!] = [\![\tau]\!] \subseteq [\![\tau]\!]_\bot$)*

(iii) $[\![\Gamma \rhd e_1\, e_2 :: \tau]\!]\eta = \begin{cases} \bot & \textit{falls } d_1 = \bot \textit{ oder } d_2 = \bot \\ d_1\, d_2 & \textit{sonst} \end{cases}$
mit $d_1 = [\![\Gamma \rhd e_1 :: \tau_2 \to \tau]\!]\eta$ und $d_2 = [\![\Gamma \rhd e_2 :: \tau_2]\!]\eta$.

(iv) $[\![\Gamma \rhd \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 :: \tau]\!]\eta = \begin{cases} \bot & \text{falls } d_0 = \bot \\ [\![\Gamma \rhd e_1 :: \tau]\!]\eta & \text{falls } d_0 = \mathit{true} \\ [\![\Gamma \rhd e_2 :: \tau]\!]\eta & \text{falls } d_0 = \mathit{false} \end{cases}$
mit $d_0 = [\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta$

(v) $[\![\Gamma \rhd \lambda id : \tau'.e :: \tau' \to \tau]\!]\eta = f : [\![\tau']\!] \to [\![\tau]\!]_\bot$
mit $f(d) = \underbrace{[\![\Gamma[^{\tau'}/_{id}] \rhd e :: \tau]\!] \underbrace{\eta[^{d}/_{id}]}_{\in Env_{\Gamma[^{\tau'}/_{id}]}}}_{\in [\![\tau]\!]_\bot}$ *für alle $d \in [\![\tau']\!]$*

Noch zu zeigen: f ist stetig, dann folgt $f \in [\![\tau' \to \tau]\!] \subseteq [\![\tau' \to \tau]\!]_\bot$. Dazu muss man wissen, dass alle $[\![\Gamma \rhd e :: \tau]\!] : Env_\Gamma \to [\![\tau]\!]_\bot$ stetig sind. Später!

(vi) $[\![\Gamma \rhd \mathbf{rec}\, id : \tau.e :: \tau]\!]\eta$ *ist der kleinste Fixpunkt der Funktion $[\![\Gamma \rhd \lambda id : \tau.e :: \tau \to \tau]\!]\eta$ (gültiges Typurteil, da nach Voraussetzung $\Gamma[^{\tau}/_{id}] \rhd e :: \tau$ gilt).*

Problem: Um den Fixpunktsatz anzuwenden, brauchen wir eine stetige Funktion von $[\![\tau]\!]$ nach $[\![\tau]\!]$ (also nicht *$[\![\tau]\!]_\bot$).*

$\leadsto$ *Spracheinschränkung:* $\mathbf{rec}\, id : \tau.e$ *nur erlaubt, wenn e eine λ-Abstraktion ist.*
$\leadsto$ *In wohlgetypten* **rec***-Ausdrücken muss τ Funktionstyp sein.*
Diese Einschränkung ist vernünftig: Sie erlaubt die „üblichen“ rekursiven Funktionsdefinitionen, bei denen immer ein Parameter verlangt wird.

Also: $[\![\Gamma \rhd \mathbf{rec}\, id_1 : \tau \to \tau'.\lambda id_2 : \tau.e :: \tau \to \tau']\!]\eta$ *ist der kleinste Fixpunkt der Funktion*

$$[\![\Gamma \rhd \lambda id_1 : \tau \to \tau'.\lambda id_2 : \tau.e :: (\tau \to \tau') \to (\tau \to \tau')]\!]\eta,$$

die in $[[\![\tau \to \tau']\!] \to [\![\tau \to \tau']\!]]$ liegt, weil $\lambda id_2 : \tau.e$ niemals $\bot$ liefern kann nach Punkt (v).

Noch zu zeigen: $[\![\Gamma \rhd e :: \tau]\!] : Env_\Gamma \to [\![\tau]\!]_\bot$ ist wohldefiniert und *stetig* (in Punkt (v)).

Lemma 3.1 *Seien C, D, E dcpo's, und E mit kleinstem Element $\bot_E$.*

(a) Für jeden Namen $id \in Id$ ist

$$\begin{array}{ll} subst_{id}: & Env_\Gamma \times [\![\tau]\!] \to Env_{\Gamma[^\tau/_{id}]} \\ & (\eta, d) \mapsto \eta[^d/_{id}] \end{array}$$

stetig.

(b) Wenn $f : D \to E$ stetig ist, dann ist die strikte Erweiterung *von f, d.h. die Funktion*

$$\begin{array}{ll} f^\star: & D_\bot \to E \\ & \bot \mapsto \bot_E \\ & d \mapsto f(d) \text{ für } d \in D \end{array}$$

ebenfalls stetig.

(c) Wenn $f : C \times D \to E$ stetig ist, dann ist auch

$$\begin{array}{ll} f^{\star\star}: & C_\bot \times D_\bot \to E \\ & (c,d) \mapsto f(c,d) \text{ falls } c \in C, d \in D \\ & (c,d) \mapsto \bot \text{ sonst, d.h. } c = \bot \text{ oder } d = \bot \end{array}$$

Beweis: Als Übung! □

Lemma 3.2 $[\![\Gamma \triangleright e :: \tau]\!] : Env_\Gamma \to [\![\tau]\!]_\bot$ *ist wohldefiniert und* stetig

Beweis:

(i) $[\![\Gamma \triangleright c :: \tau]\!]$ ist konstant und damit stetig.

(ii) $[\![\Gamma \triangleright id :: \tau]\!]$ ist stetig, weil das Supremum von Umgebungen „punktweise“ definiert ist: Wenn $\Delta \subseteq Env_\Gamma$ gerichtet, dann ist $(\bigsqcup \Delta)\,(id) = \bigsqcup(\Delta\; id)$.

(iii) Folgt einfach:

$$\begin{array}{rl} & [\![\Gamma \triangleright e_1\, e_2 :: \tau]\!]\eta \\ = & \begin{cases} \bot & \text{falls...} \\ \underbrace{([\![\Gamma \triangleright e_1 :: \tau_2 \to \tau]\!]\eta)([\![\Gamma \triangleright e_2 :: \tau_2]\!]\eta)}_{=apply([\![\ldots e_1 \ldots]\!]\eta, [\![\ldots e_2 \ldots]\!]\eta)} & \text{sonst} \end{cases} \\ = & apply^{\star\star}([\![\ldots e_1 \ldots]\!]\eta, [\![\ldots e_2 \ldots]\!]\eta) \\ = & (apply^{\star\star} \circ \langle [\![\ldots e_1 \ldots]\!], [\![\ldots e_2 \ldots]\!] \rangle)\,\eta \end{array}$$

Da *apply* stetig ist und nach Induktionsannahme die $[\![\ldots e_i \ldots]\!]$ stetig sind, folgt mit den Lemmata die Stetigkeit von $[\![\Gamma \triangleright e_1\, e_2 :: \tau]\!]$.

(iv) Es gilt:

$$\begin{array}{rl} & [\![\Gamma \triangleright \textbf{if}\, e_0\, \textbf{then}\, e_1\, \textbf{else}\, e_2 :: \tau]\!]\eta \\ = & \begin{cases} \bot & \text{falls } [\![\Gamma \triangleright e_0 :: \textbf{bool}]\!]\eta = \bot \\ [\![\Gamma \triangleright e_1 :: \tau]\!]\eta & \text{falls } [\![\Gamma \triangleright e_0 :: \textbf{bool}]\!]\eta = \mathit{true} \\ [\![\Gamma \triangleright e_2 :: \tau]\!]\eta & \text{falls } [\![\Gamma \triangleright e_0 :: \textbf{bool}]\!]\eta = \mathit{false} \end{cases} \end{array}$$

Sei

$$\begin{array}{ll} cond: & [\![\mathbf{bool}]\!] \to [[\![\tau]\!]_\perp \times [\![\tau]\!]_\perp \to [\![\tau]\!]_\perp] \\ & true \mapsto pr_1 \\ & false \mapsto pr_2 \end{array}$$

Dann ist $cond$ wohldefiniert, da pr_1, pr_2 stetig sind, und $cond$ ist stetig, da $[\![\mathbf{bool}]\!]$ eine diskrete dcpo ist.

Also ist auch $cond^\star : [\![\mathbf{bool}]\!]_\perp \to [[\![\tau]\!]_\perp \times [\![\tau]\!]_\perp \to [\![\tau]\!]_\perp]$ stetig.
Beachte: $cond^\star(\perp)$ ist das kleinste Element in $[[\![\tau]\!]_\perp \times [\![\tau]\!]_\perp \to [\![\tau]\!]_\perp]$, d.h. die konstante Funktion mit Resultat $\perp$.

Offensichtlich gilt

$$[\![\Gamma \rhd \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 :: \tau]\!]\eta = cond^\star([\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta)\,([\![\Gamma \rhd e_1 :: \tau]\!]\eta, [\![\Gamma \rhd e_2 :: \tau]\!]\eta)$$

Nach Induktionsannahme sind die $[\![\ldots e_i \ldots]\!]$ stetig. Ausserdem ist $cond^\star$ stetig. Also ist $[\![\Gamma \rhd \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 :: \tau]\!]$ stetig, da sie aus stetigen Funktionen „zusammengesetzt“ ist, genauer:

$$\begin{array}{rl} & [\![\Gamma \rhd \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 :: \tau]\!]\eta \\ = & cond^\star([\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta)\,(\langle [\![\Gamma \rhd e_1 :: \tau]\!], [\![\Gamma \rhd e_2 :: \tau]\!]\rangle \eta) \\ = & apply(cond^\star([\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta), \langle \ldots \rangle \eta) \\ = & apply(\langle cond^\star \circ [\![\ldots e_0 \ldots]\!], \langle \ldots \rangle\rangle \eta) \\ = & (apply \circ \langle cond^\star \circ [\![\ldots e_0 \ldots]\!], \langle [\![\ldots e_1 \ldots]\!], [\![\ldots e_2 \ldots]\!]\rangle\rangle)\eta \end{array}$$

(v) $[\![\Gamma \rhd \lambda id : \tau.e :: \tau \to \tau']\!]\eta = f : [\![\tau]\!] \to [\![\tau']\!]_\perp$ mit

$$\begin{array}{rcl} f(d) & = & [\![\Gamma[^\tau/_{id}] \rhd e :: \tau']\!]\eta[^d/_{id}] \\ & = & [\![\Gamma[^\tau/_{id}] \rhd e :: \tau']\!]\,(subst_{id}(\eta, d)) \end{array}$$

Also

$$\begin{array}{rcl} [\![\Gamma \rhd \lambda id : \tau.e :: \tau \to \tau']\!]\eta\, d & = & [\![\Gamma[^\tau/_{id}] \rhd e :: \tau']\!]\,(subst_{id}(\eta, d)) \\ & = & ([\![\Gamma[^\tau/_{id}] \rhd e :: \tau']\!] \circ subst_{id})\,(\eta, d) \end{array}$$

d.h.

$$\underbrace{[\![\Gamma \rhd \lambda id : \tau.e :: \tau \to \tau']\!]}_{Env_\Gamma \to [[\![\tau]\!] \to [\![\tau']\!]_\perp]} \quad = \quad curry(\underbrace{[\![\Gamma[^\tau/_{id}] \rhd e :: \tau']\!] \circ subst_{id}}_{Env_\Gamma \times [\![\tau]\!] \to [\![\tau']\!]_\perp})$$

Nach Induktionsannahme ist $[\![\Gamma[^\tau/_{id}] \rhd e :: \tau']\!]$ stetig. Ausserdem ist $subst_{id}$ stetig und damit auch die Komposition von beiden. $\rightsquigarrow$ Durch $curry$ entsteht eine wohldefinierte, stetige Funktion.

(vi) $[\![\Gamma \rhd \mathbf{rec}\, id_1 : \tau \to \tau'.\lambda id_2 : \tau.e :: \tau \to \tau']\!]\eta$
$= \mu\,(\underbrace{[\![\Gamma \rhd \lambda id_1 : \tau \to \tau'.\lambda id_2 : \tau.e :: (\tau \to \tau') \to (\tau \to \tau')]\!]}_{\text{stetig nach Induktionsannahme}}\,\eta)$

Da μ nach Lemma ebenfalls eine stetige Funktion ist, entsteht also die linke Seite durch Komposition zweier stetiger Funktionen, also gilt die Behauptung. $\square$

Wie früher gelten die folgenden Lemmata:

Lemma 3.3 (Koinzidenzlemma) *Wenn* $\Gamma =_{free\,(e)} \Gamma'$ *und* $\eta =_{free\,(e)} \eta'$, *dann gilt*

$$[\![\Gamma \triangleright e :: \tau]\!]\eta \quad = \quad [\![\Gamma' \triangleright e :: \tau]\!]\eta'$$

Lemma 3.4 (Substitutionslemma) *Wenn* $\Gamma[^{\tau'}/_{id}] \triangleright e :: \tau$ *und* $\Gamma \triangleright e' :: \tau'$, *dann gilt für alle* $\eta \in Env_\Gamma$ *mit* $[\![\Gamma \triangleright e' :: \tau']\!]\eta \neq \bot$*:*

$$[\![\Gamma \triangleright e[^{e'}/_{id}] :: \tau]\!]\eta \quad = \quad [\![\Gamma[^{\tau'}/_{id}] \triangleright e :: \tau]\!]\,(\eta[^{[\![\Gamma \triangleright e' :: \tau']\!]\eta}/_{id}])$$

Beispiel *Sei* η *die leere Funktion.*

$$\begin{aligned} & [\![\mathbf{rec}\, \mathit{fact} : \mathbf{int} \to \mathbf{int}.\lambda x : \mathbf{int}.\mathbf{if}\, x = 0\, \mathbf{then}\, 1\, \mathbf{else}\, x * \mathit{fact}\,(x-1) :: \mathbf{int} \to \mathbf{int}]\!]\eta \\ = \; & \mu([\![\lambda \mathit{fact} : \mathbf{int} \to \mathbf{int}.\lambda x : \mathbf{int}.\ldots]\!]\eta) \\ = \; & \mu F \end{aligned}$$

wobei

$$\begin{aligned} F : \quad & [\![\mathbf{int} \to \mathbf{int}]\!] \to [\![\mathbf{int} \to \mathbf{int}]\!] \\ & g \mapsto [\![[\mathit{fact} : \mathbf{int} \to \mathbf{int}] \triangleright \lambda x : \mathbf{int}.\mathbf{if} \ldots :: \mathbf{int} \to \mathbf{int}]\!]\eta[^g/_{\mathit{fact}}] \end{aligned}$$

Es gilt $\mu F = \underbrace{\bigsqcup_{n \in \mathbb{N}} F^n(\bot_{[\![\mathbf{int} \to \mathbf{int}]\!]})}_{f_n}$

$$\begin{aligned} f_0 \quad &= \quad \bot_{[\![\mathbf{int} \to \mathbf{int}]\!]} = \textit{konstante } \bot\textit{-Funktion} \\ f_1 \quad &= \quad [\![[\ldots] \triangleright \lambda x : \mathbf{int}.\mathbf{if} \ldots :: \mathbf{int} \to \mathbf{int}]\!]\eta[^{f_0}/_{\mathit{fact}}] \end{aligned}$$

also

$$\begin{aligned} f_1 : \quad & [\![\mathbf{int}]\!] \to [\![\mathbf{int}]\!]_\bot \\ & n \mapsto \begin{cases} 1 & \textit{falls } n = 0 \\ \bot & \textit{sonst (da Multiplikation } \bot \textit{ liefert, wenn ein Argument } \bot \textit{ ist)} \end{cases} \end{aligned}$$

Durch Induktion folgt:

$$\begin{aligned} f_i : \quad & [\![\mathbf{int}]\!] \to [\![\mathbf{int}]\!]_\bot \\ & n \mapsto \begin{cases} n! & \textit{falls } 0 \leq n \leq i-1 \\ \bot & \textit{sonst} \end{cases} \end{aligned}$$

Daraus ergibt sich:

$$\begin{aligned} (\mu F) : \quad & [\![\mathbf{int}]\!] \to [\![\mathbf{int}]\!]_\bot \\ & n \mapsto (\bigsqcup_{i \in \mathbb{N}} f_i)\, n = \bigsqcup_{i \in \mathbb{N}} (f_i\, n) = \begin{cases} n! & \textit{falls } n \geq 0 \\ \bot & \textit{sonst} \end{cases} \end{aligned}$$

Jetzt wieder zu zeigen: Adäquatheit der denotationellen Semantik, d.h. für jeden Ausdruck $e :: \beta$ gilt:

$$e \xrightarrow{*} c \quad \Leftrightarrow \quad [\![e :: \beta]\!]\eta = c \quad (\text{mit } \eta = \text{leere Funktion})$$

(Folgerung aus dieser Äquivalenz: e divergiert $\Leftrightarrow$ $[\![e :: \beta]\!]\eta = \bot$)

„$\Rightarrow$“ Zeigt man wieder indem man beweist, dass jeder small step die denotationelle Semantik erhält, d.h.:

Wenn $\Gamma \triangleright e :: \tau$ und $e \to e'$, dann $[\![\Gamma \triangleright e :: \tau]\!] = [\![\Gamma \triangleright e' :: \tau]\!]$.
(Beweis in den Übungen)

„$\Leftarrow$“ Für diese Richtung müssen wieder Relationen $\leq_\tau^0$ und $\leq_\tau$ für alle Typen τ definiert werden, die die Elemente der semantischen Bereiche mit den Werten und Ausdrücken in Beziehung setzen.

Definition 3.13 *Induktion über τ*

$$\begin{array}{lcl} \leq_\tau^0 & \subseteq & [\![\tau]\!] \times \mathit{Val}^\tau \\ \leq_\tau & \subseteq & [\![\tau]\!] \times \mathit{Exp}^\tau \end{array}$$

mit

(i) $d \leq_\beta^0 v \;\Leftrightarrow\; d = v$

(ii) $f \leq_{\tau\to\tau'}^0 v \;\Leftrightarrow\;$ *für alle* $d \in [\![\tau]\!], v' \in \mathit{Val}^\tau$ *gilt: Wenn* $d \leq_\tau^0 v'$, *dann* $f\,d \leq_{\tau'} v\,v'$

(iii) $d \leq_\tau e \;\Leftrightarrow\; d = \bot$ *oder* $d \in [\![\tau]\!]$ *und es ex.* $v \in \mathit{Val}^\tau$ *mit* $e \xrightarrow{*} v$ *und* $d \leq_\tau^0 v$

Beachte: $\bot \leq_\tau e$ gilt für *jeden* Ausdruck e. Intuition: $\bot$ sagt *nichts* über den Ausdruck e aus.

Umgekehrt: Wenn $d \neq \bot$ und $d \leq_\tau e$, dann gilt (insbesondere), dass e terminiert.

Wir wollen zeigen, dass für alle (abgeschlossenen) Ausdrücke $e :: \tau$ gilt:

$$[\![e :: \tau]\!] \quad \leq_\tau \quad e$$

Als Vorbereitung: Eigenschaften der Relationen $\leq_\tau$ und $\leq_\tau^0$.

Lemma 3.5

(i) Wenn $d \sqsubseteq d'$ *und* $d' \leq_\tau^0 v$, *dann* $d \leq_\tau^0 v$.

(ii) Wenn $d \sqsubseteq d'$ *und* $d' \leq_\tau e$, *dann* $d \leq_\tau e$.

(iii) Wenn Δ *gerichtet und* $d \leq_\tau^0 v$ *für alle* $d \in \Delta$, *dann* $\bigsqcup \Delta \leq_\tau^0 v$.

(iv) Wenn Δ *gerichtet und* $d \leq_\tau e$ *für alle* $d \in \Delta$, *dann* $\bigsqcup \Delta \leq_\tau e$.

Beweis:

- (i) $\Rightarrow$ (ii)

  Sei $d \sqsubseteq d'$ und $d' \leq_\tau e$. Wenn $d' = \bot$, dann ist $d = \bot$, also $d \leq_\tau e$ trivialerweise erfüllt.

  Sei $d' \neq \bot$. Dann ex. ein $v \in \mathit{Val}$ mit $e \xrightarrow{*} v$ und $d' \leq_\tau^0 v$. Nach (i) folgt dann $d \leq_\tau^0 v$. Also auch $d \leq_\tau e$.

- (iii) $\Rightarrow$ (iv)

  Es gelte (iii). Sei $d \leq_\tau e$ für alle $d \in \Delta$. Zu zeigen: $\bigsqcup \Delta \leq_\tau e$

  Wenn $\Delta = \{\bot\}$, dann $\bigsqcup \Delta = \bot$, also $\bigsqcup \Delta \leq_\tau e$ trivial.

  Andernfalls ex. ein $d \in \Delta$ mit $d \neq \bot$. Also ex. ein $v \in \mathit{Val}$ mit $e \xrightarrow{*} v$ und $d \leq_\tau^0 v$ (für dieses d). Da „$\to$“ deterministisch ist, muss dann zu *jedem* $d \in \Delta$ das gleiche v passen, d.h. es gilt sogar $d \leq_\tau^0 v$ für alle $d \in \Delta \setminus \{\bot\}$. Also gilt nach (iii) $\bigsqcup \Delta \setminus \{\bot\} \leq_\tau^0 v$. Und wege $\bigsqcup \Delta = \bigsqcup \Delta \setminus \{\bot\}$ folgt $\bigsqcup \Delta \leq_\tau^0 v$ und damit $\bigsqcup \Delta \leq_\tau e$.

- (i) durch Induktion über τ

  Basistyp β: $d \sqsubseteq d' \Leftrightarrow d = d'$, also nichts zu zeigen.

  Funktionstyp $\tau \to \tau'$: Sei $f \sqsubseteq g$ und $g \leq^0_{\tau\to\tau'} v$. Per Definition gilt für alle $d \in [\![\tau]\!]$ und für alle $v' \in Val^\tau$ mit $d \leq^0_\tau v'$: $g\,d \leq_\tau v\,v'$

  Wegen $f \sqsubseteq g$ gilt $f\,d \sqsubseteq g\,d$ für alle $d \in [\![\tau]\!]$, also folgt nach Induktionsvoraussetzung $f\,d \leq_\tau v\,v'$ (für alle $d \in [\![\tau]\!]$, $v' \in Val^\tau$ mit $d \leq^0_\tau v'$). Das bedeutet $f \leq^0_{\tau\to\tau'} v$.

- (iii) durch Induktion über τ

  Basistyp β: Klar, da Δ nur einelementig sein kann.

  Funktionstyp $\tau \to \tau'$: Sei $f \leq^0_{\tau\to\tau'} v$ für alle $f \in \Delta$. Zu zeigen: $\bigsqcup \Delta \leq^0_{\tau\to\tau'} v$. Sei $d \in [\![\tau]\!], v' \in Val^\tau$ mit $d \leq^0_\tau v'$. Zu zeigen: $(\bigsqcup \Delta)\,d \leq_{\tau'} v\,v'$, wobei $(\bigsqcup \Delta)\,d = \bigsqcup\limits_{f\in\Delta} f\,d$.

  Wegen $f \leq^0_{\tau\to\tau'} v$ gilt $f\,d \leq_{\tau'} v\,v'$ für alle $f \in \Delta$. Nach Induktionsannahme für τ' folgt also $\bigsqcup\limits_{f\in\Delta} f\,d \leq_{\tau'} v\,v'$. ☐

Satz 3.9 *Für jeden Ausdruck $e \in Exp^\tau$ gilt:*

$$[\![e :: \tau]\!]\eta \quad \leq_\tau \quad e \quad \textit{(für die leere Umgebung } \eta)$$

Beweis: Behauptung wird auf beliebige (nicht unbedingt abgeschlossene) Ausdrücke e verallgemeinert. Wenn $\underbrace{[id_1 : \tau_1, \ldots, id_n : \tau_n]}_{\Gamma} \rhd e :: \tau$ und $\eta \models \Gamma$, dann gilt für alle $e_i \in Exp^{\tau_i}$ mit $i = 1, \ldots, n$:

$$\text{Wenn } \eta(id) \leq_{\tau_i} e_i \text{ für } i = 1, \ldots, n, \text{ dann } [\![\Gamma \rhd e :: \tau]\!]\eta \leq_\tau e[^{e_i}/_{id_i}]^n_{i=1}.$$

Vorbemerkungen:

- Da $\eta\,(id) \in [\![\tau_i]\!]$, also $\eta\,(id_i) \neq \bot$ bedeutet die Voraussetzung $\eta\,(id_i) \leq_{\tau_i} e_i$ insbesondere, dass die Ausdrücke e_i alle terminieren.
- Wenn $[\![\Gamma \rhd e :: \tau]\!]\eta = \bot$, so ist nichts zu zeigen, also können wir uns auf den Fall $[\![\Gamma \rhd e :: \tau]\!]\eta \neq \bot$ beschränken.

Induktion über die Grösse von e (ausgewählte Fälle):

- Applikation: $e\,e'$

  Sei $[\![\Gamma \rhd e\,e' :: \tau]\!]\eta \neq \bot$. Dann gilt:

  $$[\![\Gamma \rhd e\,e' :: \tau]\!]\eta \quad = \quad ([\![\Gamma \rhd e :: \tau' \to \tau]\!]\eta)\,([\![\Gamma \rhd e' :: \tau']\!]\eta)$$

  Nach Induktionsannahme gilt:
  - $[\![\Gamma \rhd e :: \tau' \to \tau]\!]\eta \leq_{\tau\to\tau'} e[^{e_i}/_{id_i}]^n_{i=1}$
  - $[\![\Gamma \rhd e' :: \tau']\!]\eta \leq_{\tau'} e'[^{e_i}/_{id_i}]^n_{i=1}$

  Also existieren $v \in Val^{\tau'\to\tau}$ und $v' \in Val^{\tau'}$ mit
  - $[\![\Gamma \rhd e :: \tau' \to \tau]\!]\eta \leq^0_{\tau\to\tau'} v$ und $e[^{e_i}/_{id_i}]^n_{i=1} \xrightarrow{*} v$
  - $[\![\Gamma \rhd e' :: \tau']\!]\eta \leq^0_{\tau'} v'$ und $e'[^{e_i}/_{id_i}]^n_{i=1} \xrightarrow{*} v'$

Per Definition von $\leq^0_{\tau\to\tau'}$ folgt:

$$\underbrace{(\llbracket \Gamma \rhd e :: \tau' \to \tau \rrbracket \eta)\,(\llbracket \Gamma \rhd e' :: \tau' \rrbracket \eta)}_{=\llbracket \Gamma \rhd e\,e' :: \tau \rrbracket \eta} \leq_\tau v\,v'$$

d.h. es ex. ein $v'' \in \mathit{Val}^\tau$ mit $\llbracket \Gamma \rhd e\,e' :: \tau \rrbracket \eta \leq^0_\tau v''$ und $v\,v' \xrightarrow{*} v''$.

$$\begin{array}{rcl} (e\,e')[^{e_i}/_{id_i}]_{i=1}^n & = & (e[^{e_i}/_{id_i}]_{i=1}^n)\,(e'[^{e_i}/_{id_i}]_{i=1}^n) \\ & \xrightarrow{*} & v\,v' \\ & \xrightarrow{*} & v'' \end{array}$$

Damit ist bewiesen $\llbracket \Gamma \rhd e\,e' :: \tau \rrbracket \eta \leq_\tau (e\,e')[^{e_i}/_{id_i}]_{i=1}^n$.

- Rekursion: $\mathbf{rec}\,id : \tau \to \tau'.\lambda id' : \tau.e$

  $\llbracket \Gamma \rhd \mathbf{rec}\,id : \tau \to \tau'.\lambda id' : \tau.e :: \tau \to \tau' \rrbracket \eta = \mu F = \bigsqcup_{k\in\mathbb{N}} F^f(\bot_{\llbracket \tau\to\tau' \rrbracket})$, wobei

  $$\begin{array}{rl} F: & \llbracket \tau \to \tau' \rrbracket \to \llbracket \tau \to \tau' \rrbracket \\ & g \mapsto \llbracket \Gamma[^{\tau\to\tau'}/_{id}] \rhd \lambda id' : \tau.e :: \tau \to \tau' \rrbracket \eta[^g/_{id}] \end{array}$$

  Nach dem Lemma genügt es zu zeigen, dass für jedes $k \in \mathbb{N}$ gilt:

  $$F^k(\bot_{\llbracket \tau\to\tau' \rrbracket}) \leq_{\tau\to\tau'} (\mathbf{rec}\,id : \tau \to \tau'.\lambda id' : \tau.e)[^{e_i}/_{id_i}]_{i=1}^n$$

  Induktion über k:

  - $k = 0$

    Zu zeigen: $\bot_{\llbracket \tau\to\tau' \rrbracket} \leq_{\tau\to\tau'} (\mathbf{rec}\ldots)[^{e_i}/_{id_i}]_{i=1}^n$

    d.h. ex. ein $v \in \mathit{Val}^{\tau\to\tau'}$ mit $\bot_{\llbracket \tau\to\tau' \rrbracket} \leq^0_{\tau\to\tau'} v$ und $(\mathbf{rec}\ldots)[^{e_i}/_{id_i}]_{i=1}^n \xrightarrow{*} v$.

    Es ist klar, dass $(\mathbf{rec}\ldots)[^{e_i}/_{id_i}]_{i=1}^n$ terminiert, denn nach (Unfold) ergibt sich eine λ-Abstraktoin. Also bleibt $\bot_{\llbracket \tau\to\tau' \rrbracket} \leq^0_{\tau\to\tau'} v$ zu zeigen. Sei also $d \in \llbracket \tau \rrbracket$ und $v' \in \mathit{Val}^\tau$ mit $d \leq^0_\tau v'$. Dann gilt $\bot_{\llbracket \tau\to\tau' \rrbracket}\, d = \bot \leq_{\tau'} v\,v'$ (unabhängig von v und v').

  - $k \rightsquigarrow k+1$

    Nach Induktionsannahme für k gilt: $F^k(\bot_{\llbracket \tau\to\tau' \rrbracket}) \leq_{\tau\to\tau'} (\mathbf{rec}\ldots)[^{e_i}/_{id_i}]_{i=1}^n$

    Es gilt

    $$\begin{array}{rcl} F^{k+1}(\bot_{\llbracket \tau\to\tau' \rrbracket}) & = & F(F^k(\bot_{\llbracket \tau\to\tau' \rrbracket})) \\ & = & \llbracket \Gamma[^{\tau\to\tau'}/_{id}] \rhd \lambda id' : \tau.e :: \tau \to \tau' \rrbracket\, \underbrace{\eta[^{F^k(\bot_{\llbracket \tau\to\tau' \rrbracket})}/_{id}]}_{\eta'} \end{array}$$

    O.B.d.A.: $id \notin \{id_1, \ldots, id_n\}$

    Wegen $\eta(id_i) \leq_{\tau_i} e_i$ gilt auch $\eta'(id_i) \leq_{\tau_i} e_i$ für $i = 1, \ldots, n$.

    Ausserdem: $\eta'(id) \leq_{\tau\to\tau'} (\mathbf{rec}\ldots)[^{e_i}/_{id_i}]_{i=1}^n$

    Also gilt nach Induktionsannahme für den kleineren Ausdruck $\lambda id' : \tau.e$:

    $$\llbracket \Gamma[^{\tau\to\tau'}/\,] \rhd \lambda id' : \tau.e :: \tau \to \tau' \rrbracket \eta' \quad \leq_{\tau\to\tau'} \quad v$$

    mit $v = (\lambda\ldots)[^{e_i}/_{id_i}]_{i=1}^n[^{(\mathbf{rec}\ldots)[^{e_i}/_{id_i}]_{i=1}^n}/_{id}]$, also der Wert, der sich mit (Unfold) aus dem $\mathbf{rec}$-Ausdruck ergibt.

Damit ist gezeigt: $F^{k+1}(\perp_{[\![\tau\to\tau']\!]}) \leq_{\tau\to\tau'} v$, d.h. $F^{k+1}(\perp_{[\![\tau\to\tau']\!]}) \leq^0_{\tau\to\tau'} v$.
Damit ist

$$F^{k+1}(\perp_{[\![\tau\to\tau']\!]}) \leq_{\tau\to\tau'} (\mathbf{rec}\ldots)[{}^{e_i}/{}_{id_i}]_{i=1}^n$$

bewiesen, da $(\mathbf{rec}\ldots)[{}^{e_i}/{}_{id_i}]_{i=1}^n \xrightarrow{*} v$. □

Satz 3.10 (Adäquatheit der denotationellen Semantik)

(a) *Für jeden abgeschlossenen Ausdruck $e :: \tau$ gilt:*

$$[\![e :: \tau]\!]\eta \neq \perp \quad\Leftrightarrow\quad e \textit{ terminiert} \quad (\textit{für leeres } \eta)$$

(d.h. $[\![e :: \tau]\!] = \perp \Leftrightarrow e$ divergiert)

(b) *Für jeden Ausdruck $e :: \beta$ gilt:*

$$[\![e :: \beta]\!]\eta = v \quad\Leftrightarrow\quad e \xrightarrow{*} v$$

Beweis:

(a) „$\Rightarrow$“ Nach dem letzten Satz gilt: $[\![e :: \tau]\!]\eta \leq_\tau e$. Also terminiert e, weil $[\![e :: \tau]\!]\eta \neq \perp$.

„$\Leftarrow$“ Es gelte $e \xrightarrow{*} v$. Da small steps die denotationelle Semantik erhalten, gilt dann

$$[\![e :: \tau]\!]\eta = \underbrace{[\![v :: \tau]\!]\eta}_{\in[\![\tau]\!]\neq\perp} \;\Rightarrow\; [\![e :: \tau]\!]\eta \neq \perp$$

(für leeres η)

(b) „$\Rightarrow$“ Wenn $[\![e :: \beta]\!] = v$, dann ist $[\![e :: \beta]\!] \neq \perp$, also gilt nach (a) $e \xrightarrow{*} v$. Da small steps die denotationelle Semantik erhalten, folgt:

$$v = [\![e :: \beta]\!] = [\![v' :: \beta]\!] = v'$$

„$\Leftarrow$“ Klar. □

3.6 Folgerung aus der Adäquatheit

Wenn $[\![\Gamma \rhd e :: \tau]\!] = [\![\Gamma \rhd e' :: \tau]\!]$, dann sind e und e' *beobachtungskongruent* bezüglich Γ, das heisst, sie sind in allen (Γ, τ)-Programmkontexten gegeneinander austauschbar, ohne dass sich das beobachtbare Verhalten des Programms verändert (Beweis wie früher: Weil die denotationelle Semantik „kompositionell“ ist, bleibt die denotationelle Semantik des Programms erhalten, wenn man einen Ausdruck im Programm durch einen denotationell äquivalenten Ausdruck ersetzt. $\leadsto$ Wegen der Adäquatheit bleibt auch das beobachtbare Verhalten gleich).
Frage: Gilt auch die Umkehrung?
D.h. müssen zwei beobachtungskongruente Ausdrücke die gleiche denotationelle Semantik haben? (sog. „fully abstraction“)

Antwort: Nein!

Intuitive Begründung: Die denotationellen Bereiche enthalten unnötige Funktionen, wie das „parallele oder“ oder das „parallele **if-then-else**“, die sich in $\mathcal{L}_2^t$ nicht implementieren lassen (weil $\mathcal{L}_2^t$ „sequentiellen Charakter“ hat).

$\rightsquigarrow$ Dies führt in der denotationellen Semantik zu “unnötigen Unterscheidungen“, die operationell nicht auftreten (weil es keinen Programmkontext gibt, der diese Unterscheidung erzwingt).

Lösungsversuche:

(1) Denotationelle Semantik verbessern, indem man Funktionen mit „parallelem“ Verhalten entfernt ($\rightsquigarrow$ keine einfachen Lösungen).

(2) Die Sprache $\mathcal{L}_2^t$ erweitern $\rightsquigarrow$ mehr Programmkontexte $\rightsquigarrow$ Verfeinerung der Beobachtungskongruenz $\rightsquigarrow$ Übereinstimmung von denotationeller Semantik und Beobachtungskongruenz

(wurde von Plotkin 1977 gelöst $\rightsquigarrow$ Hinzunahme des parallelen **if-then-else** genügt, um full abstraction mit unserer dcpo-Semantik zu erreichen)

Welche denotationellen Äquivalenzen gelten in unserer Semantik? Gelten die für $\mathcal{L}_1^t$ bewiesenen auch für $\mathcal{L}_2^t$?

Im allgemeinen nicht! Da $\mathcal{L}_2^t$ mehr Programmkontexte enthält, ist die Beobachtungskongruenz *feiner* als die von $\mathcal{L}_1^t$, d.h. manche Beobachtungskongruenzen von $\mathcal{L}_1^t$ gelten in $\mathcal{L}_2^t$ *nicht*.

Beispiel *In $\mathcal{L}_1^t$ gilt die sog. η-Konversion (Übung 2, Aufgabe 1):*

> *Wenn $id \notin free\,(e)$, dann $[\![\Gamma \triangleright e :: \tau \to \tau']\!] = [\![\Gamma \triangleright \lambda id : \tau.e\; id :: \tau \to \tau']\!]$*

Kann (in dieser Allgemeinheit) für $\mathcal{L}_2^t$ nicht gelten, denn: Wenn e divergiert, dann ist $[\![\Gamma \triangleright e :: \tau \to \tau']\!] = \bot$, aber $[\![\Gamma \triangleright \lambda id : \tau.e\; id :: \tau \to \tau']\!] \neq \bot$.

Stattdessen gilt nur folgende Einschränkung:

> *Wenn $id \notin free\,(e)$ und $[\![\Gamma \triangleright e :: \tau \to \tau']\!] \neq \bot$, dann ist*
> $[\![\Gamma \triangleright e :: \tau \to \tau']\!] = [\![\Gamma \triangleright \lambda id : \tau.e\; id :: \tau \to \tau']\!]$.

Spezialfall (sog. η-value-Regel):

> *Wenn $id \notin free\,(v)$, dann gilt:*
> $[\![\Gamma \triangleright v :: \tau \to \tau']\!] = [\![\Gamma \triangleright \lambda id : \tau.v\; id :: \tau \to \tau']\!]$.

4 Denotationelle Semantik für call-by-name

Ist einfacher als call-by-value, da man keinen Unterschied zwischen der Semantik von Ausdrücken und der Semantik von Werten machen muss.

Definition 4.1 (Semantische Bereiche) Alle *semantischen Bereiche enthalten ein kleinstes Element. Sie sind durch Induktion über τ definiert:*

$$\begin{array}{rcll} [\![\mathbf{int}]\!] & = & \mathbb{Z}_\bot & \\ [\![\mathbf{bool}]\!] & = & \mathit{Bool}_\bot & \\ [\![\mathbf{unit}]\!] & = & \mathit{Unit}_\bot & \\ [\![\tau_1 \to \tau_2]\!] & = & [[\![\tau_1]\!] \to [\![\tau_2]\!]] & \text{(Bereich der stetigen Fktn. mit argumentweiser Ord.)} \\ [\![\tau_1 * \tau_2]\!] & = & [\![\tau_1]\!] \times [\![\tau_2]\!] & \text{(Produktdcpo, d.h. Paare sind komponentenweise geord.)} \end{array}$$

Beispiel

(1) $[\![\mathbf{int} \to \mathbf{int}]\!] = [\mathbb{Z}_\bot \to \mathbb{Z}_\bot]$

Da $\mathbb{Z}_\bot$ nur endliche aufsteigende Folgen hat, sind die stetigen Funktionen genau die monotonen. Für eine monotone Funktion $f : \mathbb{Z}_\bot \to \mathbb{Z}_\bot$ gibt es zwei Möglichkeiten:

(a) f ist strikt, d.h. $f(\bot) = \bot$. Dann kann man $f(n)$ für alle $n \in \mathbb{Z}$ beliebig wählen.

(b) $f(\bot) \in \mathbb{Z}$. Dann gilt für alle $n \in \mathbb{Z}$: $f(\bot) \sqsubseteq f(n)$ und damit $f(\bot) = f(n)$, also f konstant.

Also gilt: $[\![\mathbf{int} \to \mathbf{int}]\!]$ besteht aus den strikten und aus den konstanten Funktionen.

<u>Intuition:</u> Konstante Funktionen werden als Semantik von λ-Abstraktionen benötigt, die ihr Argument ignorieren. Strikte Funktionen stehen für λ-Abstraktionen, die ihr Argument wirklich benutzen, z.B.:

- $[\![\lambda x : \mathbf{int}.0]\!] = f : \mathbb{Z}_\bot \to \mathbb{Z}_\bot$ *mit* $f(\bot) = 0$.
- $[\![\lambda x : \mathbf{int}.x * 0]\!] = g : \mathbb{Z}_\bot \to \mathbb{Z}_\bot$ *mit* $g(\bot) = \bot$, $g(n) = 0$ *für alle* $n \in \mathbb{Z}$

(2) $[\![\mathbf{bool} \to \mathbf{bool} \to \mathbf{bool}]\!] = [\mathit{Bool}_\bot \to [\mathit{Bool}_\bot \to \mathit{Bool}_\bot]]$

enthält z.B. die currifizierte Version des „parallelen oder“, d.h. die Funktion

$$\mathit{por} : \mathit{Bool}_\bot \to [\mathit{Bool}_\bot \to \mathit{Bool}_\bot]$$

$$\mathit{por}\, d\, e = \begin{cases} \mathit{true} & \mathit{falls}\ d = \mathit{true}\ \mathit{oder}\ e = \mathit{true} \\ \mathit{false} & \mathit{falls}\ d = e = \mathit{false} \\ \bot & \mathit{sonst.} \end{cases}$$

Definition 4.2 *Umgebungen η sind wie üblich definiert als partielle Funktionen*

$$\eta : \mathit{Id} \rightharpoonup \bigcup_{\tau \in \mathit{Type}} [\![\tau]\!]$$

<u>Beachte:</u> Es kann $\eta\,(id) = \bot_{[\![\tau]\!]}$ gelten, also insbesondere $\eta\,(id) = \bot_{[\![\mathbf{int}]\!]}$. Das entspricht der call-by-name-Parameterübergabe: Auch divergierende Ausdrücke e können für einen Namen id eingesetzt werden.

Definition 4.3 (Semantik von Ausdrücken) *Jedem gültigen Typurteil $\Gamma \triangleright e :: \tau$ wird eine Funktion*

$$[\![\Gamma \triangleright e :: \tau]\!] : \mathit{Env}_\Gamma \to [\![\tau]\!]$$

zugeordnet (Beachte: $[\![\tau]\!]$ statt $[\![\tau]\!]_\bot$ im Vergleich zu call-by-value) durch Induktion über die Grösse von e:

- $[\![\Gamma \rhd c :: \beta]\!]\eta = c$ *für jede Konstante vom Basistyp*
- $[\![\Gamma \rhd op :: \mathbf{int} \to \mathbf{int} \to \beta]\!]\eta = curry((op^I)^{\star\star})$
  also die doppelt strikte Erweiterung der Funktion op^I, $(op^I)^{\star\star} : \mathbb{Z}_\perp \times \mathbb{Z}_\perp \to [\![\beta]\!]_\perp$
- $[\![\Gamma \rhd fst :: \tau_1 * \tau_2 \to \tau_1]\!]\eta = pr_1 : [\![\tau_1]\!] \times [\![\tau_2]\!] \to [\![\tau_1]\!]$
  (Beachte pr_1 *ist stetig, also* $pr_1 \in [[\![\tau_1]\!] \times [\![\tau_2]\!] \to [\![\tau_1]\!]] = [\![\tau_1 * \tau_2 \to \tau_1]\!]$*)*
- $[\![\Gamma \rhd e_1\, e_2 :: \tau]\!]\eta = \underbrace{([\![\Gamma \rhd e_1 :: \tau' \to \tau]\!])}_{\in[[\![\tau']\!]\to[\![\tau]\!]]=[\![\tau'\to\tau]\!]}\ \underbrace{([\![\Gamma \rhd e_2 :: \tau']\!])}_{\in[\![\tau']\!]}$

  Beachte: Wenn $[\![\Gamma \rhd e_1 :: \tau' \to \tau]\!]\eta = \perp_{[\![\tau'\to\tau]\!]}$, *dann ergibt sich* $\perp_{[\![\tau]\!]}$, *weil* $\perp_{[\![\tau'\to\tau]\!]}$ *die konstante Funktion mit Resultat* $\perp_{[\![\tau]\!]}$ *ist.*

  Andererseits: Wenn $[\![\Gamma \rhd e_2 :: \tau']\!] = \perp_{[\![\tau']\!]}$ *kann trotzdem ein Resultat* $\neq \perp_{[\![\tau]\!]}$ *entstehen.*
- $[\![\Gamma \rhd (e_1, e_2) :: \tau_1 * \tau_2]\!]\eta = ([\![\Gamma \rhd e_1 :: \tau_1]\!]\eta, [\![\Gamma \rhd e_2 :: \tau_2]\!]\eta)$
- $[\![\Gamma \rhd \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 :: \tau]\!]\eta = \begin{cases} \perp_{[\![\tau]\!]} & \text{falls } [\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta = \perp_{[\![\mathbf{bool}]\!]} \\ [\![\Gamma \rhd e_1 :: \tau]\!]\eta & \text{falls } [\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta = true \\ [\![\Gamma \rhd e_2 :: \tau]\!]\eta & \text{falls } [\![\Gamma \rhd e_0 :: \mathbf{bool}]\!]\eta = false \end{cases}$
- $[\![\Gamma \rhd \lambda id : \tau.e :: \tau \to \tau']\!]\eta = f : [\![\tau]\!] \to [\![\tau']\!]$ *mit* $f(d) = \underbrace{[\![\Gamma[{}^\tau/_{id}] \rhd e :: \tau']\!]\ \underbrace{\eta[{}^d/_{id}]}_{\in Env_{\Gamma[{}^\tau/_{id}]}}}_{\in[\![\tau']\!]}$ *für alle* $d \in [\![\tau]\!]$
- $[\![\Gamma \rhd \mathbf{rec}\, id : \tau.e :: \tau]\!]\eta = \mu(\underbrace{[\![\Gamma \rhd \lambda id : \tau.e :: \tau \to \tau]\!]}_{\in[\![\tau\to\tau]\!]=[[\![\tau]\!]\to[\![\tau]\!]]}$

  Beachte: $[\![\tau]\!]$ *besitzt kleinstes Element* $\perp_{[\![\tau]\!]}$, *also existiert der kleinste Fixpunkt jeder stetigen Funktion.*

Beispiel

- $[\![\mathbf{rec}\, f : \mathbf{int} \to \mathbf{int}.f]\!]\eta = \mu(\underbrace{\lambda f : \mathbf{int} \to \mathbf{int}.f}_{\text{Identität auf } [\![\mathbf{int} \to \mathbf{int}]\!]}) = \perp_{[\![\mathbf{int}\to\mathbf{int}]\!]}$
- $[\![\mathbf{rec}\, f : \mathbf{int} \to \mathbf{int}.\lambda x : \mathbf{int}.f\, x]\!]\eta = \mu([\![\lambda f : \mathbf{int} \to \mathbf{int}.\lambda x : \mathbf{int}.f\, x]\!]\eta) = \mu F$ *wobei*

  $$F(g) \quad = \quad \underbrace{[\![[f : \mathbf{int} \to \mathbf{int}] \rhd \lambda x : \mathbf{int}.f\, x :: \mathbf{int} \to \mathbf{int}]\!]\eta[{}^g/_f]}_{=g}$$

  also $\mu F = \perp_{[\![\mathbf{int}\to\mathbf{int}]\!]}$

<u>Also:</u> Beide Ausdrücke haben die *gleiche* denotationelle Semantik, obwohl sie sich operationell unterschiedlich verhalten (der eine divergiert selbst, der andere erst, wenn er mit einem Argument aufgerufen wird).

Kann unter diesen Umständen Adäquatheit gelten?

Satz 4.1 (Adäquatheit der call-by-name Semantik) *Für jeden Ausdruck* $e :: \beta$ *(abgeschlossen,* Basistyp*) gilt:*

$$e \xrightarrow{*} c \quad \Leftrightarrow \quad [\![e :: \beta]\!]\eta = c \quad \textit{(leere Umgebung } \eta\textit{)}$$

(und damit: e divergiert $\Leftrightarrow [\![e :: \beta]\!] = \perp_{[\![\beta]\!]}$*)*

Beachte: Die Adäquatheit gilt hier, weil wir *nur* abgeschlossene Ausdrücke vom Basistyp „beobachten" können. Könnte man auch Ausdrücke vom Funktionstyp „beobachten" (bezgüglich Terminieren oder Divergenz), dann wäre diese Semantik nicht adäquat, denn dann könnte man $\mathbf{rec}\, f.f$ und $\mathbf{rec}\, f.\lambda x.f\, x$ unterscheiden.

4.1 Vergleich zu call-by-value

In call-by-value Semantik muss $\mathbf{rec}\, f.f$ und $\mathbf{rec}\, f.\lambda x.f\, x$ auf jeden Fall unterschieden werden (auch wenn man nur Beobachtungen vom Basistyp zulässt), denn die Programme

$$(\lambda f : \mathbf{int} \to \mathbf{int}.0)\, (\mathbf{rec}\, f.f)$$

und

$$(\lambda f : \mathbf{int} \to \mathbf{int}.0)\, (\mathbf{rec}\, f.\lambda x.f\, x)$$

verhalten sich verschieden in call-by-value.

Intuitive Erklärung: In call-by-name Semantik gibt es nur *eine* Möglichkeit, einen Ausdruck vom Funktionstyp auszuwerten: Man muss ihn mit einem Argument aufrufen (Regel (App-Left)).
Dann verhält sich aber ein selbst divergierender Ausdruck wie $\mathbf{rec}\, f.f$ genauso wie ein Ausdruck, der erst beim Aufruf divergiert, wie $\mathbf{rec}\, f.\lambda x.f\, x$.

5 Vergleich call-by-value vs. call-by-name

Substitutionslemma: Gilt in der cbv-Semantik nur in eingeschränkter Form, nämlich:

Wenn $[\![\Gamma \triangleright e' :: \tau']\!]\eta \in [\![\tau']\!]$ (also $\neq \bot$), dann gilt:

$$[\![\Gamma \triangleright e[^{e'}/_{id}] :: \tau]\!]\eta \quad = \quad [\![\Gamma[^{\tau'}/_{id}] \triangleright e :: \tau]\!]\eta[^{[\![\Gamma \triangleright e' :: \tau']\!]}/_{id}]$$

In cbn-Semantik gilt diese Gleichung uneingeschränkt, also *ohne* Voraussetzung über e'.

Folgerung: β-value- bzw. β-Regel.

- In cbv gilt

  $$[\![\Gamma \triangleright (\lambda id : \tau'.e)\, e' :: \tau]\!]\eta \quad = \quad [\![\Gamma \triangleright e[^{e'}/_{id}] :: \tau]\!]\eta$$

  falls $[\![\Gamma \triangleright e' :: \tau']\!]\eta \neq \bot$, also insbesondere die sog. β-value-Regel:

  $$[\![\Gamma \triangleright (\lambda id : \tau'.e)\, v :: \tau]\!] \quad = \quad [\![\Gamma \triangleright e[^{v}/_{id}] :: \tau]\!]$$

- In cbn gilt die 1. Gleichung *ohne* Voraussetzung, also die sog. β-Regel:

  $$[\![\Gamma \triangleright (\lambda id : \tau'.e)\, e' :: \tau]\!] \quad = \quad [\![\Gamma \triangleright e[^{e'}/_{id}] :: \tau]\!]$$

η-value- bzw. η-Regel:

- In cbv gilt: Wenn $id \notin \mathit{free}\,(e)$ und $[\![\Gamma \triangleright e :: \tau \to \tau']\!]\eta \neq \bot$, dann ist

  $$[\![\Gamma \triangleright \lambda id : \tau.e\; id :: \tau \to \tau']\!]\eta \quad = \quad [\![\Gamma \triangleright e :: \tau \to \tau']\!]\eta$$

  Insbesondere gilt die η-value-Regel:

  Wenn $id \notin \mathit{free}\,(v)$, dann $[\![\Gamma \triangleright \lambda id : \tau.v\; id :: \tau \to \tau']\!] \quad = \quad [\![\Gamma \triangleright v :: \tau \to \tau']\!]$

- In cbn gilt die 1. Gleichung *ohne* Einschränkung für e, d.h. es gilt die η-Regel:

  Wenn $id \notin \mathit{free}\,(e)$, dann $[\![\Gamma \triangleright \lambda id : \tau.e\; id :: \tau \to \tau']\!] \quad = \quad [\![\Gamma \triangleright e :: \tau \to \tau']\!]$

α-Konversion: In beiden Semantiken gilt die α-Konversion (= Umbenennung von Parametern):

Wenn $id' \notin \mathit{free}\,(e)$, dann $[\![\Gamma \triangleright \lambda id : \tau.e :: \tau \to \tau']\!] \quad = \quad [\![\Gamma \triangleright \lambda id' : \tau.e[^{id'}/_{id}] :: \tau \to \tau']\!]$

Currifizierung:

Definition 5.1 *Seien D, E dcpo's. Eine Isomorphie zwischen D und E ist eine bijektive Funktion $\phi : D \to E$, für die gilt: ϕ und ϕ^{-1} sind stetig.*

In cbn-Semantik gilt:

$$\begin{array}{l} \mathit{curry} : [\![\tau_1 * \tau_2 \to \tau_3]\!] \to [\![\tau_1 \to \tau_2 \to \tau_3]\!] \\ \mathit{curry}\, f\, d\, e = f(d, e) \end{array}$$

ist eine Isomorphie. *curry* ist stetig, denn *curry* ist Semantik eines abgeschlossenen Ausdrucks:

$$\mathit{curry} \quad = \quad [\![\lambda f : \tau_1 * \tau_2 \to \tau_3 . \lambda x : \tau_1 . \lambda y : \tau_2 . f\,(x, y) :: \ldots]\!]$$

Umkehrfunktion von *curry* ist die Funktion

$$\begin{array}{l} \mathit{decurry} : [\![\tau_1 \to \tau_2 \to \tau_3]\!] \to [\![\tau_1 * \tau_2 \to \tau_3]\!] \\ \mathit{decurry}\, g\, (d, e) = g\, d\, e \end{array}$$

decurry ist ebenfalls stetig, denn

$$\mathit{decurry} \quad = \quad [\![\lambda g : \tau_1 \to \tau_2 \to \tau_3 . \lambda z : \tau_1 * \tau_2 . g\,(\mathit{fst}\, z)\,(\mathit{snd}\, z) :: \ldots]\!]$$

curry und *decurry* sind tatsächlich invers zueinander, wie man leicht sehen kann.

In cbv sind *curry* und *decurry* *keine* Isomorphismen (da sie nicht bijektiv sind). Für die semantischen Bereiche gilt:

$$[\![\tau_1 * \tau_2 \to \tau_3]\!] \quad = \quad [[\![\tau_1]\!] \times [\![\tau_2]\!] \to [\![\tau_3]\!]_\perp],$$

enthält also 2 Elemente, wenn $\tau_i = \mathbf{unit}$ für $i = 1, \ldots, 3$. Hingegen

$$[\![\tau_1 \to \tau_2 \to \tau_3]\!] \quad = \quad [[\![\tau_1]\!] \to [[\![\tau_2]\!] \to [\![\tau_3]\!]_\perp]_\perp]$$

enthält 3 Elemente für $\tau_i = \mathbf{unit}$.

curry ist nicht surjektiv, weil die konstante Funktion mit Resultat $\perp$ (die, die schon mit einem Argument divergiert) liegt nicht im Bild von *curry*.

decurry ist nicht injektiv, weil die Funktion f_1, die mit einem Argument schon $\perp$ liefert, und die Funktion f_2, die erst im 2. Argument stets $\perp$ liefert, werden durch *decurry* beide auf die konstante $\perp$-Funktion abgebildet.