Musterlösung zur Klausur Semantik von Programmiersprachen

Stephan Thesing

25. September 1998

I Posets und cpos

Aufgabe I.1 Sei $P = \{a, b\}^*$ die Menge der endlichen Worte über dem Alphabet $\{a, b\}$, geordnet durch $v \sqsubseteq w$ gdw. $\exists x, y \in P : xvy = w$ (Infixordnung).

1. Beweisen Sie, daß P eine Poset ist.
2. Ist P eine cpo? (Beweis oder Gegenbeispiel)
3. Ist die Konkatenation: $\cdot : P \times P \to P$ monoton?
4. Die Funktion $tail : P \to P$ bildet das leere Wort ϵ auf ϵ ab und entfernt das 1. Zeichen von nichtleeren Worten. Ist $tail$ monoton? (Beweis oder Gegenbeispiel)

`Lösung:`

1. Zu zeigen ist, daß $\sqsubseteq$ reflexiv, transitiv und antisymmetrisch ist.
   (a) Reflexivität: Für alle $v \in P$ gilt: $\epsilon v \epsilon = v$, da $\epsilon \in P$ kann man $x = y = \epsilon \in P$ wählen, so daß $xvy = v \Rightarrow v \sqsubseteq v$
   (b) Transitivität: Sei $v \sqsubseteq w,\ w \sqsubseteq z$, dann gibt es $x, y, x', y' \in P$ mit $xvy = w$ und $x'wy' = z$. Damit gilt (Einsetzen): $x'xvyy' = z$. Wähle nun $x'' = x'x$, $y'' = yy' \in P$, so gilt $x''vy'' = z \Rightarrow v \sqsubseteq z$.
   (c) Antisymmetrie: Sei $v \sqsubseteq w$ und $w \sqsubseteq v$, dann gibt es $x, y, x', y' \in P$ mit $xvy = w$ und $x'wy' = v$. Einsetzen liefert $xx'wy'y = w$, was nur für $xx' = y'y = \epsilon$ gelten kann, was sofort auf $x = x' = \epsilon$ und $y = y' = \epsilon$ führt, mithin $v = w$.
2. P ist keine cpo. Betrachte als Gegenbeispiel die Kette $d_i := \overbrace{a \cdots a}^{i\text{–mal}}$. Offensichtlich gilt, mit $x = a, y = \epsilon$: $xd_iy = d_{i+1} \Rightarrow$

$d_i \sqsubseteq d_{i+1}$, d.h. d_i ist eine Kette. Wäre P cpo, so müßte eine kleinste oberste Schranke $d \in P$ existieren. Diese müßte (per definitionem) auch eine obere Schranke der a_i sein.

Da $d \in P$ ist es ein endliches Wort. O.B.d.A. nehmen wir an, es habe Länge m: $|d| = m$. Betrachte nun das Kettenelement d_{m+1}. Da d obere Schranke, muß gelten, daß $d_{m+1} \sqsubseteq d$, anders ausgedrückt: $\exists x, y \in P : xd_{m+1}y = d$. Aber wir haben, daß $|xd_{m+1}y| \geq m + 1 > m = |d|$, was dieser Annahme widerspricht. Daraus folgt, daß es keine oberste Schranke der d_i geben kann, mithin auch keine kleinste oberste Schranke.

3. Die Konkatenation ist nicht monoton. Gegenbeispiel: Wähle $d = (a, a)$, $e = (ab, ab)$. Offensichtlich gilt, daß $d \sqsubseteq_\times e$ in der Produkt-cpo $P \times P$. Wäre die Konkatenation monoton, müßte $d' = \cdot(a, a) = aa \sqsubseteq abab = \cdot(ab, ab)$ sein, was offensichtlich nicht gilt (aa ist kein Infix von $abab$).

4. Die Funktion *tail* ist monoton. Beweis:

$$\begin{array}{lcl} tail(\epsilon) & = & \epsilon \\ tail(c.w) & = & w \text{ , wobei } c \in \{a, b\} \wedge w \in P \end{array}$$

   Sei nun $v, w \in P$ mit $v \sqsubseteq w$. Zu zeigen ist $tail(v) \sqsubseteq tail(w)$. Wir tun dies per Induktion über die Länge von v.

   (a) $|v| = 0$, d.h. $v = \epsilon$. Dann ist $tail(v) = \epsilon \sqsubseteq tail(w) \forall w$, da ϵ das kleinste Element von P ist.

   (b) $|v| > 1$, d.h. $v = c.v'$ mit $c \in \{a, b\}$, $v' \in P$. Da $v \sqsubseteq w$ gibt es $x, y \in P$ mit $xvy = xc.v'y = w$. Wir unterscheiden weiter nach dem Aufbau von x.

      i. $x = \epsilon$. Dann ist $tail(w) = tail(xc.v'y) = tail(c.v'y) = v'y$. Da $tail(v) = tail(c.v') = v'$ kann man offensichtlich $x' = \epsilon$, $y' = y \in P$ wählen mit $x'v'y' = x' tail(v) y' = v'y = tail(xc.v'y) = tail(w)$, woraus die Behauptung folgt.

      ii. $x = c'.x'$. Dann ist $tail(w) = tail(c'.x'.c.v'.y) = x'.c.v'.y = x'.c.tail(v).y$. Wiederum wähle $x'' = x'.c$, $y'' = y$ und man hat $x'' tail(v) y'' = tail(w)$, woraus die Behauptung folgt.

Aufgabe I.2 Sei $(P, \sqsubseteq)$ eine cpo und $f : P \to P$ monoton (nicht notwendig stetig).
Zeigen Sie, daß $S := \{x \in P \mid x \sqsubseteq f(x)\}$ zulässig ist.

Lösung:
Man muß zeigen, daß für jede Kette $d_i \sqsubseteq d_{i+1},\ i \in \omega$ mit $d_i \in S$ auch die kleinste oberste Schranke $d := \bigsqcup_{i\in\omega} d_i$ in S liegt: $d \in S$.

Sei also d_i so eine Kette, dann ist es hinreichend zu zeigen, daß $d \sqsubseteq f(d)$ gilt, weil damit $d \in S$, per definitionem von S.

Da d der lub der d_i ist es auch obere Schranke: $\forall i : d_i \sqsubseteq d$. Da f monoton folgt daraus $\forall i : f(d_i) \sqsubseteq f(d)$. Aus der Annahme, daß $d_i \in S$ folgt $d_i \sqsubseteq f(d_i)$, insgesamt also (Transitivität von $\sqsubseteq$): $\forall i : d_i \sqsubseteq f(d)$. Das bedeutet aber, daß $f(d)$ eine obere Schranke der d_i ist. Da d die *kleinste* obere Schranke ist, folgt $d \sqsubseteq f(d)$, also, wie behauptet $d \in S$.

Aufgabe I.3 Seien P und Q zwei cpos mit kleinsten Elementen $\perp_P$ und $\perp_Q$ und seien $f : P \to P$ und $g : Q \to Q$ stetig, sowie $h : P \times Q \to R$ eine stetige Funktion in eine cpo R mit $h(f\ x, g\ y) \sqsubseteq h(x, y)$ für alle $x \in P, y \in Q$.
Zeigen Sie $h(\mathit{lfp}\ f, \mathit{lfp}\ g) = h(\perp_P, \perp_Q)$.

Lösung: Wir zeigen $h(\mathit{lfp}\ f, \mathit{lfp}\ g) \sqsubseteq h(\perp_P, \perp_Q)$ und $h(\perp_P, \perp_Q) \sqsubseteq h(\mathit{lfp}\ f, \mathit{lfp}\ g)$, woraus sich mit der Antisymmetrie von $\sqsubseteq$ die Behauptung ergibt.

1. $h(\perp_P, \perp_Q) \sqsubseteq h(\mathit{lfp}\ f, \mathit{lfp}\ g)$: Nach Voraussetzung sind $\perp_P$ und $\perp_Q$ die kleinsten Elemente von P bzw. Q. Nach Definition der Produkt-cpo ist damit $(\perp_P, \perp_Q)$ kleinstes Element von $P \times Q$. D.h. insbesondere, daß $(\perp_P, \perp_Q) \sqsubseteq (\mathit{lfp}\ f, \mathit{lfp}\ g)$. Da h stetig ist, ist h auch monoton, woraus dann sofort $h(\perp_P, \perp_Q) \sqsubseteq h(\mathit{lfp}\ f, \mathit{lfp}\ g)$ folgt.
2. $h(\mathit{lfp}\ f, \mathit{lfp}\ g) \sqsubseteq h(\perp_P, \perp_Q)$: Nach Definition von *lfp* folgt, daß

$$A := h(\mathit{lfp}\ f, \mathit{lfp}\ g) = h(\bigsqcup_i f^i \perp_P, \bigsqcup_j g^j \perp_Q)$$

Da h stetig ist und außerdem gilt, daß $\bigsqcup_i (f^i \perp_P, g^i \perp_Q) = (\bigsqcup_i f^i \perp_P, \bigsqcup_j g^j \perp_Q)$, folgt $A = \bigsqcup_i h(f^i \perp_P, g^i \perp_Q)$.

Wie weiter unten gezeigt gilt $\forall i : h(f^i \perp_P, g^i \perp_Q) \sqsubseteq h(\perp_P, \perp_Q)$. D.h. $h(\perp_P, \perp_Q)$ ist obere Schranke der Elemente $h(f^i \perp_P, g^i \perp_Q)$, welche eine Kette bilden.

Damit folgt, daß die kleinste oberste Schranke dieser Kette *kleiner* als $h(\perp_P, \perp_Q)$ ist. Nun ist $\bigsqcup_i h(f^i \perp_P, g^i \perp_Q)$ gerade diese kleinste oberste Schranke, woraus

$$\bigsqcup_i h(f^i \bot_P, g^i \bot_Q) \sqsubseteq h(\bot_P, \bot_Q)$$

folgt.

Wie schon vorhin gezeigt, folgt aus der Stetigkeit von h, daß

$$\bigsqcup_i h(f^i \bot_P, g^i \bot_Q) = h(\bigsqcup_i f^i \bot_P, \bigsqcup_i g^i \bot_Q) = h(\mathit{lfp}\ f, \mathit{lfp}\ g)$$

womit die Behauptung folgt.

3. Wir zeigen noch $\forall i : h(f^i \bot_P, g^i \bot_Q) \sqsubseteq h(\bot_P, \bot_Q)$ per Induktion über i:
   (a) $i = 0$. Die Behauptung folgt sofort aus der Reflexivität von $\sqsubseteq$.
   (b) $i = k + 1$. $h(f^{k+1} \bot_P, g^{k+1} \bot_Q) \sqsubseteq h(f^k \bot_P, g^k \bot_Q) \sqsubseteq h(\bot_P, \bot_Q)$. Wobei die erste Beziehung aufgrund der Voraussetzung des Satzes folgt, die zweite aufgrund der Induktionsannahme.

Aufgabe I.4 Seien P und Q zwei cpos und sei $f : P \to Q$ monoton und $g : Q \to P$ monoton mit $x \sqsubseteq g(f\ x)\ \forall x \in P$ und $f(g\ y) \sqsubseteq y\ \forall y \in Q$.

Zeigen Sie: f ist stetig.

Lösung: Man hat für alle Ketten $d_i \sqsubseteq d_{i+1}$ zu zeigen

$$f(\bigsqcup_i d_i) = \bigsqcup_i f(d_i)$$

Wir zeigen zunächst, daß die $f(d_i)$ eine Kette bilden, mithin die rechte Seite definiert ist. Es gilt ja $d_i \sqsubseteq d_{i+1}$, woraus mit der Monotonie von f sofort $f(d_i) \sqsubseteq f(d_{i+1})$ folgt. Was zeigt, daß die $f(d_i)$ eine Kette bilden.

Um das Leben etwas angenehmer zu machen, definieren wir:

$$d := \bigsqcup_i d_i$$
$$e := \bigsqcup_i f(d_i)$$

Um $f(d) = e$ zu zeigen, zeigen wir $f(d) \sqsubseteq e$ und $e \sqsubseteq f(d)$.

1. $e \sqsubseteq f(d)$: Es gilt ja immer: $d_i \sqsubseteq d$, da d eine obere Schranke der d_i. Mit der Monotonie von f folgt dann $\forall i : f(d_i) \sqsubseteq f(d)$. D.h. $f(d)$ ist eine obere Schranke der $f(d_i)$. Da e aber die kleinste obere Schranke der $f(d_i)$ ist, folgt $e \sqsubseteq f(d)$, wie gewünscht.

2. $f(d) \sqsubseteq e$: Es gilt immer, daß $f(d_i) \sqsubseteq e$, da e eine obere Schranke der $f(d_i)$. Da g monoton ist, gilt auch $g(f(d_i)) \sqsubseteq g(e)$. Nach Voraussetzung gilt $d_i \sqsubseteq g(f(d_i))$, also auch $\forall i : d_i \sqsubseteq g(e)$. Das heißt aber, daß $g(e)$ eine obere Schranke der d_i ist. Damit gilt $d \sqsubseteq g(e)$, da d die kleinste obere Schranke der d_i. Da f monoton ist, gilt $f(d) \sqsubseteq f(g(e))$. Nach Voraussetzung gilt aber $f(g(e)) \sqsubseteq e$, somit $f(d) \sqsubseteq e$, wie verlangt.

II IMP and friends

Aufgabe II.1 Die Sprache IMP sei um eine neue Anweisung der Form

$$\texttt{do}\ C\ \texttt{while}\ B$$

erweitert, wobei C eine Anweisung und B eine Bedingung ist.
Dies stellt eine Schleifenanweisung dar, welche auf jeden Fall zuerst C ausführt, dann die Bedingung B testet. Ist diese `false`, endet die Schleife, ist sie `true` beginnt ein neuer Durchlauf.

Geben Sie die denotationelle Semantik dieser Anweisung an.

Lösung: Am einfachsten führt man die Semantik auf die der `while` Schleife zurück:

$$[\![\texttt{do}\ \ c\ \ \texttt{while}\ b]\!] = [\![c\ \ ;\ \texttt{while}\ b\ \texttt{do}\ c]\!]$$

Etwas ausfürlicher geht's aber auch:

$$[\![\texttt{do}\ \ c\ \ \texttt{while}\ b]\!] = \mathit{fix}\ \lambda f.\lambda\sigma.\mathit{cond}([\![b]\!]_\perp([\![c]\!]\sigma), f_\perp([\![c]\!]\sigma), [\![c]\!]\sigma)$$

Hierbei ist $\mathit{cond} : \mathbf{T}_\perp \times \Sigma_\perp \times \Sigma_\perp \to \Sigma_\perp$ definiert als

$$\mathit{cond}(b, \sigma, \sigma') = \begin{cases} \sigma & \text{, falls } b = \texttt{true} \\ \sigma' & \text{, falls } b = \texttt{false} \\ \perp & \text{, sonst} \end{cases}$$

Aufgabe II.2 Die Sprache IMP soll um eine Anweisung `break` erweitert werden. Wenn diese im Rumpf einer Schleife ausgeführt wird, wird die Schleife sofort verlassen. Tritt es nicht im Rumpf einer Schleife auf, wird das Programm verlassen.

Um diese erweiterte Sprache durch eine *operationale Semantik* beschreiben zu können, muß der Typ der Ergebnissemantik leicht geändert werden, so daß man Aussagen der Form $\langle C \mid \sigma\rangle \to \langle\sigma' \mid b\rangle$ mit einem Wahrheitswert b erhält. Dabei ist $b = \texttt{true}$ gdw. C durch

Ausführung eines `break` endete, das innerhalb von C nicht in einer Schleife liegt, so daß eine Schleife, die C als Rumpf enthält, beendet werden muß.

Durch den geändertern Typ müssen alle semantischen Regeln geändert werden. Ihre Aufgabe besteht darin, dies für `skip`, `break`, $C_1 ; C_2$ und `while` B `do` C zu tun.

Lösung:

$$\langle \texttt{break}, \sigma \rangle \to \langle \sigma, \texttt{true} \rangle$$

$$\langle \texttt{skip}, \sigma \rangle \to \langle \sigma, \texttt{false} \rangle$$

$$\frac{\langle c_1, \sigma \rangle \to \langle \sigma', \texttt{true} \rangle}{\langle c_1 \texttt{ ; } c_2, \sigma \rangle \to \langle \sigma', \texttt{true} \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \to \langle \sigma', \texttt{false} \rangle \quad \langle c_2, \sigma' \rangle \to \langle \sigma'', b \rangle}{\langle c_1 \texttt{ ; } c_2 \rangle \to \langle \sigma'', b \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \texttt{false}}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \langle \sigma, \texttt{false} \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \texttt{true} \quad \langle c, \sigma \rangle \to \langle \sigma', \texttt{true} \rangle}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \langle \sigma', \texttt{false} \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \texttt{true} \quad \langle c, \sigma \rangle \to \langle \sigma', \texttt{false} \rangle \quad \langle \texttt{while } b \texttt{ do } c, \sigma' \rangle \to \langle \sigma'', b' \rangle}{\langle \texttt{while } b \texttt{ do } c, \sigma \rangle \to \langle \sigma'', b' \rangle}$$

III Hoare Kalkül

Aufgabe III.1 Zeigen Sie mit Hilfe der Hoare-Regeln, daß für alle Anweisungen C gilt:

$$\vdash \{\texttt{false}\} C \{\texttt{false}\}$$

Lösung: Beweis durch Induktion nach C.

1. $C \equiv \texttt{skip}$: Die Behauptung folgt unmittelbar aus der Regel für `skip`:

   $$\{P\}\texttt{skip}\{P\}$$

   für $P = \texttt{false}$.

2. $C \equiv X := A$: Da $\texttt{false}$ keine Variablen enthält, gilt immer $\texttt{false}[A/X] \equiv \texttt{false}\ \forall X, A$. Durch Wahl von $P = \texttt{false}$ in dem Axiom

$$\{P[A/X]\}X := A\{P\}$$

folgt die Behauptung sofort.

3. $C \equiv C_1 ; C_2$: Per Induktionsannahme gilt $\{\texttt{false}\}C_i\{\texttt{false}\}$ für $i \in \{1, 2\}$. Wählen von $P = Q = R = \texttt{false}$ in dem Axiom

$$\frac{\{P\}C_1\{Q\} \quad \{Q\}C_2\{R\}}{\{P\}C_1 ; C_2\{R\}}$$

beweist die Behauptung.

4. $C \equiv \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2$: Nach I.A. gilt $\{\texttt{false}\}C_i\{\texttt{false}\}$ für $i \in \{1, 2\}$. Betrachte die Konsequenzregel

$$\frac{\models (P \Rightarrow P') \quad \{P'\}C\{Q'\} \quad \models (Q' \Rightarrow Q)}{\{P\}C\{Q\}}$$

Offensichtlich gilt $\forall B' : \models (\texttt{false} \wedge B' \Rightarrow \texttt{false})$ (Definition von $\models$). Wählt man $C = C_1$, $B' = B$ einerseits und $C = C_2$, $B' = \neg B$ andererseits, ergibt sich aus der Konsequenzregel $\{\texttt{false} \wedge B\}C_1\{\texttt{false}\}$ und $\{\texttt{false} \wedge \neg B\}C_2\{\texttt{false}\}$. Anwendung der Regel für $\texttt{if}$,

$$\frac{\{P \wedge B\}C_1\{Q\} \quad \{P \wedge \neg B\}C_2\{Q\}}{\{P\}\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\{Q\}}$$

mit $P = Q = \texttt{false}$ ergibt dann die Behauptung.

5. $C \equiv \texttt{while}\ B\ \texttt{do}\ C$: Nach I.A. gilt $\{\texttt{false}\}C\{\texttt{false}\}$. Wie bei dem $\texttt{if}$-Fall kann man unter Benutzung der Konsequenzregel $\{\texttt{false} \wedge B\}C\{\texttt{false}\}$ ableiten. Unter Benutzung der $\texttt{while}$ Regel

$$\frac{\{P \wedge B\}C\{P\}}{\{P\}\texttt{while}\ B\ \texttt{do}\ C\{P \wedge \neg B\}}$$

kann man damit $\{\texttt{false}\}\texttt{while}\ B\ \texttt{do}\ C\{\texttt{false} \wedge \neg B\}$ ableiten. Benutzung der Konsequenzregel mit $P = P' = \texttt{false}$ und $Q' = \texttt{false} \wedge \neg B$, $Q = \texttt{false}$ ergibt dann die Behauptung, da allgemein $\models (Q' \Rightarrow Q)$ für diese Q, Q' gilt.

Aufgabe III.2 Welche der folgenden Hoare-Tripel sind gültig ($\models$)? Begründen Sie Ihre Antwort.

1. $\{n = X\}\ X\texttt{:=}1\{n = 1\}$
2. $\{P\}\ X\texttt{:=}Y\ \{P[X/Y]\}\ \ (X, Y \in \mathit{Loc})$
3. $\{X = 1\}$ `while` $X > 0$ `do` $X\texttt{:=}X + 1\ \{X = 1\}$

Lösung:

Um zu zeigen, daß ein Tripel $\{P\}\ C\ \{Q\}$ *nicht* gültig ist, reicht es, ein Gegenbeispiel zu geben. Dazu gibt man einen Zustand σ und eine Interpretation I an, für welche $\sigma \models^I P$ gültig ist, aber nicht $[\![C]\!]\sigma \models^I Q$.

1. $\{n = X\}\ X\texttt{:=}1\ \{n = 1\}$ ist nicht gültig. Gegenbeispiel: $\sigma = \{2/X\}$, $I = \{2/n\}$. Damit $[\![X\texttt{:=}1]\!]\sigma = \{1/X\} =: \sigma'$, da $I(n) = 2 \neq 1 = \sigma'(X)$ folgt, daß $[\![X\texttt{:=}1]\!]\sigma \not\models^I n = 1$.
2. $\{P\}\ X\texttt{:=}Y\ \{P[X/Y]\}$ ist nicht gültig. Gegenbeispiel: Wähle $P \equiv X > Y$, $\sigma = \{2/X, 1/Y\}$, I beliebig. Dann gilt $\sigma \models^I P$. Betrachte nun $\sigma' := [\![X\texttt{:=}Y]\!]\sigma$ und $P[X/Y] \equiv X > X$ ($P[X/Y]$ bedeutet: ersetze Y durch X !!). Für alle $\sigma' \in \Sigma$ gilt, daß $\sigma' \not\models P[X/Y]$. Da $\sigma' = \{1/X, 1/Y\} \neq \bot$, folgt, daß $\sigma' \not\models^I P[X/Y]$.
3. $\{X = 1\}$ `while` X>0 `do` $X\texttt{:=}X\texttt{+}1\ \{X = 1\}$ ist gültig. Schreibe w für die `while` Schleife. Zuerst einmal ist $[\![w]\!]\sigma = \bot$ für alle σ, mit $\sigma(X) > 0$. Erfüllt nun ein σ die Vorbedingung $X = 1$, gilt offenbar $\sigma(X) > 0$, somit terminiert die Schleife nicht und ihre Semantik ist $\bot$. Per definitionem gilt aber für alle I und Prädikate P: $\bot \models^I P$. Also auch für $P \equiv X = 1$. Damit ist das Tripel gültig.

Aufgabe III.3 Zeigen Sie: Für alle Anweisungen C und Prädikate P und Q:

$$wp(C \mid P \vee Q) \cong wp(C \mid P) \vee wp(C \mid Q)$$

Gilt $wp(C \mid \texttt{false}) \cong \texttt{false}$? (Begründen Sie Ihre Antwort!)

Lösung:
Es gilt $\sigma \models^I wp(C \mid P) \iff [\![C]\!]_\bot\sigma \models^I P$. Um Äquivalenz zu zeigen: $P \cong Q \iff (\forall \sigma, I : \sigma \models^I P \iff \sigma \models^I Q)$.

Wir haben also zu zeigen:

$$\forall \sigma, I : \sigma \models^I wp(C \mid P \vee Q) \iff \sigma \models^I (wp(C \mid P) \vee wp(C \mid Q))$$

Seien σ und I nun beliebig:

$$\begin{array}{lll} & \sigma \models^I wp(C \mid P \vee Q) & \\ \iff & [\![C]\!]_\perp\sigma \models^I (P \vee Q) & \text{Def. } wp \\ \iff & [\![C]\!]_\perp\sigma \models^I P \text{ oder } [\![C]\!]_\perp\sigma \models^I Q & \text{Def. } \vee \\ \iff & \sigma \models^I wp(C \mid P) \text{ oder } \sigma \models^I wp(C \mid Q) & \text{Def. } wp \\ \iff & \sigma \models^I (wp(C \mid P) \vee wp(C \mid Q)) & \text{Def. } \vee \end{array}$$

Zur Frage, ob $wp(C \mid \texttt{false}) \cong \texttt{false}$ gilt: Es gilt nicht. Beweis: Gälte es, müßte für alle σ, I gelten:

$$\sigma \models^I \texttt{false} \iff \sigma \models^I wp(C \mid \texttt{false}) \iff [\![C]\!]_\perp\sigma \models^I \texttt{false}$$

Dies gilt jedoch nicht, wenn $[\![C]\!]_\perp\sigma = \perp$ für ein $\sigma \neq \perp$, also etwa für $C \equiv \texttt{while } 1 < 2 \texttt{ do } X := 1$. Es gilt etwa für $\sigma = \{1/X\}$: $\sigma \not\models^I \texttt{false}$, aber $[\![C]\!]_\perp\sigma = \perp \models^I \texttt{false}$.

IV Funktionale Sprachen

Aufgabe IV.1 In der funktionalen Sprache 1. Ordnung sei eine Funktion $f : i \to i$ wie folgt definiert:

```
f n = if n = 0 then 0 else f(n-1)+1
```

1. Geben Sie an, wie `f(1)` operational zu einem Wert reduziert wird (call-by-value und call-by-name).
2. Betrachten Sie die Fixpunktiteration $\varphi_0 \sqsubseteq \varphi_1 \sqsubseteq \cdots$ für die Programmsemantik $[\![P]\!]$. Bestimmen Sie $\varphi_0(f), \varphi_1(f), \varphi_2(f)$, geben Sie eine allgemeine Formel für $\varphi_k(f)$ an und bestimmen Sie $[\![P]\!](f)$. Gibt es dabei einen Unterschied zwischen call-by-name und call-by-value?

Lösung:

1. Reduktion von `f(1)`, siehe Abbildung 1.
2. Bestimmung von $\varphi_i(f)$, $i \in \{0, 1, 2\}$ und $\varphi_k(f)$.

   Sei $F(\varphi) := ([\![d]\!]_{N/V}\varphi)$ mit

   $$d \equiv (\texttt{f n = }\underbrace{\texttt{if n=0 then 0 else (f (n-1))+1}}_{=:E})$$

   Mit $\varphi_0^{N/V}(f) = \lambda n'.\perp$ gilt dann immer: $\varphi_{i+1}^{N/V}(f) = F(\varphi_i^{N/V})(f)$.
   Also:

<table>
<tr><th>Call by value</th><th>Call by name</th></tr>
<tr><td colspan="2" align="center">f(1)</td></tr>
<tr><td colspan="2" align="center">$\downarrow$ Funktionsapplikation</td></tr>
<tr><td colspan="2" align="center">if 1=0 then 0 else f (1-1) +1</td></tr>
<tr><td colspan="2" align="center">$\downarrow$ If1 (+Prim)</td></tr>
<tr><td colspan="2" align="center">if false then 0 else f (1-1) +1</td></tr>
<tr><td colspan="2" align="center">$\downarrow$ If2</td></tr>
<tr><td colspan="2" align="center">f (1-1) +1</td></tr>
<tr><td align="center">$\swarrow$ by value</td><td align="center">$\searrow$ lazy</td></tr>
<tr><td align="center">f(0) +1</td><td align="center">(if (1-1)=0 then 0 else f((1-1)-1)+1)+1</td></tr>
<tr><td align="center">$\downarrow$ Funkapp.</td><td align="center">$\downarrow$ If1 (+Prim)</td></tr>
<tr><td align="center">(if 0=0 then 0 else f(0-1) +1)+1</td><td align="center">(if 0=0 then 0 else f ((1-1)-1)+1)+1</td></tr>
<tr><td align="center">$\downarrow$ If1 (+Prim)</td><td align="center">$\downarrow$ If1 (+Prim)</td></tr>
<tr><td align="center">(if true then 0 else f(0-1) +1)+1</td><td align="center">(if true then 0 else f ((1-1)-1)+1)+1</td></tr>
<tr><td align="center">$\searrow$ If2</td><td align="center">$\swarrow$ If2</td></tr>
<tr><td colspan="2" align="center">0+1</td></tr>
<tr><td colspan="2" align="center">$\downarrow$ Prim</td></tr>
<tr><td colspan="2" align="center">1</td></tr>
</table>

Abbildung 1: Reduktion von `f(1)`

- $\varphi_0^{N/V}(f) = \lambda n'.\bot$
- $\varphi_1^{N/V}(f) = F(\varphi_0^{N/V})(f) = ([\![d]\!]_{N/V}\varphi_0^{N/V})$.

Für call-by-value:

$$
\begin{aligned}
& ([\![d]\!]_V \varphi_0^V) \\
= \; & \lambda\lfloor n' \rfloor . [\![E]\!] \varphi_0^V (n') \\
= \; & \lambda\lfloor n' \rfloor . \begin{cases} \bot & \text{, falls } n' = \bot \\ \mathit{cond}(\; [\![n=0]\!]\varphi_0^V(n'), & \text{, sonst} \\ \qquad [\![0]\!]\varphi_0^V(n'), & \\ \qquad [\![\texttt{f (n-1)+1}]\!]\varphi_0^V(n')) & \end{cases} \\
= \; & \lambda\lfloor n' \rfloor . \begin{cases} \bot & \text{, falls } n' = \bot \\ 0 & \text{, falls } n' = 0 \\ ((\lambda n''.\bot)\ (n'-1)) + 1 & \text{, sonst} \end{cases} \\
= \; & \lambda\lfloor n' \rfloor . \begin{cases} 0 & \text{, falls } n' = 0 \\ \bot & \text{, sonst} \end{cases}
\end{aligned}
$$

Man sieht leicht ein, daß $\varphi_N^1 = \varphi_V^1$, da ja für Argument $n' = \bot$, das Ergebnis auch in diesem Falle $\bot$ ist, wegen der Striktheit von *cond* und der Gleichheit $=$.

- $\varphi_2^{N/V}(f) = ([\![d]\!]_{N/V} \varphi_1^{N/V})$. Ähnliche Argumentation, wie gerade führt auf

$$\varphi_2^{N/V}(f) = \lambda n' . \begin{cases} n' & \text{, falls } 0 \leq n' \leq 1 \\ \bot & \text{, sonst} \end{cases}$$

Hierbei ist nur zu beachten, daß

$$((\varphi_1^{N/V}(f))(n'-1)) + 1 = \begin{cases} 1 & \text{, falls } n' = 1 \\ \bot & \text{, sonst} \end{cases}$$

Allgemein ist dann leicht einzusehen, daß

$$\varphi_k^{N/V}(f) = \lambda n' . \begin{cases} n' & \text{, falls } 0 \leq n' \leq k \\ \bot & \text{, sonst} \end{cases}$$

- $\varphi := [\![P]\!](f)$.

Behauptung: $\varphi(f) = \lambda n' . \begin{cases} n' & \text{, falls } 0 \leq n' \\ \bot & \text{, sonst} \end{cases}$

Beweis:

(a) $\varphi(f)$ ist obere Schranke der φ_k, d.h.

$$
\begin{aligned}
& \varphi_k(f) \sqsubseteq \varphi(f) \\
\iff \; & \forall n' : \varphi_k(f)(n') \neq \bot \Rightarrow \varphi_k(f)(n') = \varphi(f)(n')
\end{aligned}
$$

Dies stimmt offensichtlich.

(b) Zu zeigen bleibt, daß φ kleiner als alle anderen oberen Schranken ist.
Sei also ψ eine andere obere Schranke von φ_k, d.h.

$$\forall n' : \varphi_k(f)(n') \neq \bot \Rightarrow \varphi_k(f)(n') = \psi(f)(n')$$

Zu zeigen ist $\varphi \sqsubseteq \psi$, d.h.

$$\forall n' : \varphi(f)(n') \neq \bot \Rightarrow \varphi(f)(n') = \psi(f)(n')$$

Sei also für ein beliebiges n' $\varphi(f)(n') \neq \bot$. D.h. (nach Definition von φ): $\varphi(f)(n') = n' = \varphi_{n'}(f)(n') \neq \bot$. Da ψ obere Schranke ist, folgt (s.o.)

$$\varphi_{n'}(f)(n') = \psi(f)(n')$$

Also insgesamt unsere Behauptung.

Insgesamt gibt es hier keinen Unterschied zwischen call-by-value und call-by-name.

Aufgabe IV.2 Erweitern Sie die funktionale Sprache 1. Ordnung mit call-by-value Auswertung um Ausdrücke der Form

$$E_1 \texttt{ default } E_2 \text{ mit Typisierung } \frac{\Gamma \vdash E_1 : i \quad \Gamma \vdash E_2 : i}{\Gamma \vdash E_1 \texttt{ default } E_2 : i}$$

Dabei soll E_1 zuerst ausgewertet werden. Wenn dies nicht 0 ergibt, ist dies der Wert des Ausdruckes. Andernfalls liefert die Auswertung von E_2 den Wert des Ausdruckes. Geben Sie eine geeignete operationelle und denotationelle Semantik für die neuen Ausdrücke an und ergänzen Sie den Beweis von

$$[\![E]\!][\![P]\!]\rho_0 = \lfloor v \rfloor \Rightarrow E \xrightarrow{*} v$$

aus der Vorlesung um den Fall der neuen Ausdrücke.

Lösung:

- Operational:

$$\frac{E_1 \rightarrow E_1'}{E_1 \texttt{ default } E_2 \rightarrow E_1' \texttt{ default } E_2}$$

$$0 \texttt{ default } E_2 \rightarrow E_2$$

$$n \texttt{ default } E_2 \rightarrow n \text{ , falls } n \neq 0$$

- Denotationell:

$$[\![E_1 \texttt{ default } E_2]\!]\,\varphi\,\rho = cond([\![E_1]\!]\,\varphi\,\rho \overset{!}{=} 0, [\![E_2]\!]\,\varphi\,\rho, [\![E_1]\!]\,\varphi\,\rho)$$

Wobei $x \overset{!}{=} y := \begin{cases} \bot & \text{, falls } x = \bot \vee y = \bot \\ x = y & \text{, sonst} \end{cases}$

- Beweisergänzung:
Wir haben zu zeigen (siehe Vorlesung):

$$[\![E]\!]\,\varphi_k\,\rho_0 = \lfloor v \rfloor \Rightarrow E \rightarrow^* v$$

Wir behandeln also den Fall $E \equiv E_1 \texttt{ default } E_2$:
Es gilt $[\![E]\!]\,\varphi_k\,\rho_0 = cond([\![E_1]\!]\,\varphi_k\,\rho_0 \overset{!}{=} 0, [\![E_2]\!]\,\varphi_k\,\rho_0, [\![E_1]\!]\,\varphi_k\,\rho_0) \neq \bot$, somit

$$[\![E_1]\!]\,\varphi_k\,\rho_0 \neq \bot \text{ (nach Definition von } \overset{!}{=})$$

Also $[\![E_1]\!]\,\varphi_k\,\rho_0 = \lfloor v_1 \rfloor$ für ein v_1.
Nach I.A. gilt daher $E_1 \rightarrow^* v_1$. Betrachten wir nun v_1:

  - $v_1 \neq 0$, dann gilt $v = v_1$, mit der I.A. haben wir ja auch noch $E_1 \texttt{ default } E_2 \rightarrow^* v_1 \texttt{ default } E_2$, insgesamt also (mit der dritten Regel der operationalen Semantik):

$$E_1 \texttt{ default } E_2 \rightarrow^* v_1 = v$$

  - $v_1 = 0$, dann per I.A. $E_1 \rightarrow^* 0$ und da nun $\lfloor v \rfloor = [\![E_2]\!]\,\varphi_k\,\rho_0$ gilt weiter $E_2 \rightarrow^* v$
Also insgesamt

$$E_1 \texttt{ default } E_2 \rightarrow^* 0 \texttt{ default } E_2 \rightarrow E_2 \rightarrow^* v$$

Wenn man die zweite Regel der operationalen Semantik benutzt. Dies ist aber gerade unsere Behauptung.