6 Denotationale Semantik

Eine denotationale Semantik kümmert sich nur um den Effekt einer Programmausführung. Im Gegensatz dazu haben sich die operationalen Semantiken auch explizit um die Zwischenzustände gekümmert: Am deutlichsten ist dies bei der Small-Step-Semantik, deren Ableitungsfolgen erst einmal alle Zwischenschritte enthalten – das interessante Gesamtverhalten erhält man erst durch die Abstraktion zur transitiven Hülle.

Nun könnte man einfach weiter abstrahieren: Da die operationalen Semantiken für While deterministisch sind (vgl. Thm. 2 und Kor. 5), kann man diese Relationen auch als (partielle) Funktionen $\mathcal{D}[\![c]\!]$ auf Zuständen auffassen: Nimmt man die Big-Step-Semantik, erhält man:

$$\mathcal{D}[\![c]\!]\,\sigma = \begin{cases} \sigma' & \text{falls } \langle c,\ \sigma\rangle \Downarrow \sigma' \\ \bot & \text{falls } \nexists \sigma'.\ \langle c,\ \sigma\rangle \Downarrow \sigma' \end{cases}$$

Auch auf Basis der Small-Step-Semantik ließe sich diese Funktion $\mathcal{D}[\![c]\!]$ für das Programm c entsprechend definieren:

$$\mathcal{D}[\![c]\!]\,\sigma = \begin{cases} \sigma' & \text{falls } \langle c,\ \sigma\rangle \overset{*}{\to}_1 \langle \texttt{skip},\ \sigma'\rangle \\ \bot & \text{falls } \langle c,\ \sigma\rangle \overset{\infty}{\to}_1 \end{cases}$$

In beiden Varianten gewinnt man so allerdings nichts, da Beweise von Aussagen über $\mathcal{D}[\![c]\!]$ erst einmal die Definition auffalten müssen und man dann wieder bei den Ausführungsdetails der operationalen Semantik landet. Der große Vorteil einer denotationellen Semantik liegt allerdings genau darin, dass man viele Theoreme *ohne* Rückgriff auf Ableitungsbäume zeigen und verwenden kann.

Eine denotationale Semantik sollte *kompositional* sein, d.h., je eine Funktion für die syntaktischen Kategorien (Aexp, Bexp und Com) *rekursiv über dem Syntaxbaum* definieren. Dadurch ordnet sie jedem Syntaxelement (Syntaxbaum) ein mathematisches Objekt, i.d.R. eine Funktion, zu, das den Effekt bzw. das Ergebnis dieses syntaktischen Konstrukts beschreibt.

Beispiel 22. Die Auswertungsfunktionen $\mathcal{A}[\![\_]\!]$ bzw. $\mathcal{B}[\![\_]\!]$ für arithmetische bzw. boolesche Ausdrücke sind bereits so definiert. Eigentlich sind dies bereits denotationale Semantiken und keine operationalen. Hier nochmals die Defintion von $\mathcal{A}[\![\_]\!]$ vom Typ $\mathsf{Aexp} \Rightarrow \Sigma \Rightarrow \mathbb{N}$:

$$\begin{array}{rcl} \mathcal{A}[\![n]\!]\,\sigma & = & \mathcal{N}[\![n]\!] \\ \mathcal{A}[\![x]\!]\,\sigma & = & \sigma(x) \\ \mathcal{A}[\![a_1\ \texttt{-}\ a_2]\!]\,\sigma & = & \mathcal{A}[\![a_1]\!]\,\sigma - \mathcal{A}[\![a_2]\!]\,\sigma \\ \mathcal{A}[\![a_1\ \texttt{*}\ a_2]\!]\,\sigma & = & \mathcal{A}[\![a_1]\!]\,\sigma \cdot \mathcal{A}[\![a_2]\!]\,\sigma \end{array}$$

Übung: Finden Sie eine operationale Semantik für arithmetische Ausdrücke, sowohl Big-Step als auch Small-Step.

6.1 Denotationale Semantik

Bei unserer imperativen While-Sprache ist das Ergebnis einer Ausführung eines Programms c die Zustandsänderung. Es bietet sich an, diese Änderung als Funktion von Anfangs- zu Endzustand zu beschreiben. Da ein Programm bei manchen Anfangszustände möglicherweise nicht terminiert, müssen diese Funktionen partiell sein. Die mathematischen Bedeutungsobjekte für Com sind damit partielle Funktionen auf Zuständen:

$$\mathcal{D}[\![\_]\!] :: \mathsf{Com} \Rightarrow (\Sigma \rightharpoonup \Sigma)$$

Die obige Definition von $\mathcal{D}[\![\_]\!]$ über die Big-Step-Semantik entspricht genau diesem Typ. Sie ist aber nicht kompositional, selbst wenn man die Definition der Big-Step-Semantik einsetzt: Für `while (`b`) do` c ergibt sich aus den Regeln $\textsc{WhileFF}_{\text{BS}}$ und $\textsc{WhileTT}_{\text{BS}}$:

$$\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!](\sigma) = \begin{cases} \sigma & \text{falls } \mathcal{B}[\![b]\!]\sigma = \mathbf{ff} \\ \mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!](\mathcal{D}[\![c]\!](\sigma)) & \text{falls } \mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \end{cases}$$

Dies ist aber keine kompositionale Definition, weil der zu definierende Term $\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!]$ sowohl links wie auch rechts des Gleichheitszeichens in gleicher Form auftritt. Genau genommen ist dies erst einmal überhaupt keine Definition, weil nicht klar ist, dass es eine Funktion mit dieser Eigenschaft überhaupt gibt und dass diese Eigenschaft die Funktion eindeutig festlegt.

Der Standard-Ansatz, um solche rekursiven Gleichungen in eine Definition zu verwandeln, ist, das zu Definierende in einen Parameter eines Funktionals umzuwandeln und es als einen ausgewählten Fixpunkt des Funktionals zu definieren. Damit ergibt sich folgendes Funktional $F :: (\Sigma \rightharpoonup \Sigma) \Rightarrow \Sigma \rightharpoonup \Sigma$ aus der rekursiven Gleichung für $\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!]$:

$$F(f)(\sigma) = \begin{cases} \sigma & \text{falls } \mathcal{B}[\![b]\!]\sigma = \mathbf{ff} \\ f(\mathcal{D}[\![c]\!]\sigma) & \text{falls } \mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \end{cases}$$

Beispiel 23. Für das Programm `while (not (x == 0)) do x := x - 1` vereinfacht sich dieses Funktional zu:

$$F(f)(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ f(\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1]) & \text{falls } \sigma(\mathtt{x}) \neq 0 \end{cases}$$

Ein Fixpunkt f von F erfüllt $F(f) = f$. Damit ergeben sich folgende Anforderungen an alle möglichen Lösungen f:

- f muss alle Zustände σ mit $\sigma(\mathtt{x}) = 0$ auf σ selbst abbilden.
- f muss alle Zustände σ mit $\sigma(\mathtt{x}) \neq 0$ auf den gleichen Zustand abbilden wie $\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1]$.

Sehr viele Funktionen erfüllen diese Kriterien, z.B. alle der Form

$$f(\sigma) = \begin{cases} \sigma[\mathtt{x} \mapsto 0] & \text{falls } \sigma(\mathtt{x}) \geq 0 \\ \sigma' & \text{falls } \sigma(\mathtt{x}) < 0 \end{cases}$$

wobei $\sigma' \in \Sigma$ beliebig ist. Ebenso ist

$$f^*(\sigma) = \begin{cases} \sigma[\mathtt{x} \mapsto 0] & \text{falls } \sigma(\mathtt{x}) \geq 0 \\ \bot & \text{falls } \sigma(\mathtt{x}) < 0 \end{cases}$$

ein Fixpunkt von F. All diese Fixpunkte unterscheiden sich nur an den Zuständen, für die die Ausführung des Programms nicht terminiert.

Beispiel 24. Es gibt auch Funktionale, die gar keinen Fixpunkt haben: $F(f) = \begin{cases} f_1 & \text{falls } f = f_2 \\ f_2 & \text{sonst} \end{cases}$ hat für $f_1 \neq f_2$ keinen Fixpunkt.

Die Lösung des Problems mit der Existenz und Eindeutigkeit einer Lösung der rekursiven Spezifikation wird sein:

1. eine Menge von Funktionalen des Typs $(\Sigma \rightharpoonup \Sigma) \Rightarrow (\Sigma \rightharpoonup \Sigma)$ zu finden, die immer einen Fixpunkt haben, und dann den „undefiniertesten“ Fixpunkt auswählen (Kap. 6.2);
2. zu zeigen, dass alle Funktionale, die durch While-Programme entstehen, in dieser Menge enthalten sind (Kap. 6.3).

Definition 31 (Fixpunktiteration). Eine Funktion $f :: D \Rightarrow D$ kann wie folgt *iteriert* werden:

$$f^0(d) = d \qquad f^{n+1}(d) = f(f^n(d))$$

Man kann nun ein Funktional iterieren und erhält dadurch schrittweise eine Annäherung an die Lösung der Gleichung, für die das Funktional steht. Dabei startet man mit der überall undefinierten Funktion $\bot$ (für alle σ gilt $\bot(\sigma) = \bot$), die keinerlei Information trägt.

Beispiel 25. Für das Funktional F aus Beispiel 23 ergibt sich folgende Fixpunkt-Iteration:

$$F^0(\bot) = \bot$$

$$F^1(\bot)(\sigma) = F(F^0(\bot))(\sigma) = F(\bot)(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ \bot(\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1) = \bot & \text{falls } \sigma(\mathtt{x}) \neq 0 \end{cases}$$

$$F^2(\bot)(\sigma) = F(F^1(\bot))(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ F^1(\bot)(\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1]) & \text{falls } \sigma(\mathtt{x}) \neq 0 \end{cases}$$

$$= \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1] & \text{falls } \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1](\mathtt{x}) = 0 \text{ und } \sigma(\mathtt{x}) \neq 0 \\ \bot & \text{falls } \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1](\mathtt{x}) \neq 0 \text{ und } \sigma(\mathtt{x}) \neq 0 \end{cases}$$

$$= \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ \sigma[\mathtt{x} \mapsto 0] & \text{falls } \sigma(\mathtt{x}) = 1 \\ \bot & \text{sonst} \end{cases}$$

$$F^3(\bot)(\sigma) = F(F^2(\bot))(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ F^2(\bot)(\sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1]) & \text{falls } \sigma(\mathtt{x}) \neq 0 \end{cases}$$

$$= \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1] & \text{falls } \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1](\mathtt{x}) = 0 \text{ und } \sigma(\mathtt{x}) \neq 0 \\ \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1,\ \mathtt{x} \mapsto 0] & \text{falls } \sigma[\mathtt{x} \mapsto \sigma(\mathtt{x}) - 1](\mathtt{x}) = 1 \text{ und } \sigma(\mathtt{x}) \neq 0 \\ \bot & \text{sonst} \end{cases}$$

$$= \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ \sigma[\mathtt{x} \mapsto 0] & \text{falls } \sigma(\mathtt{x}) \in \{\, 1,\ 2 \,\} \\ \bot & \text{sonst} \end{cases}$$

Allgemein gilt:

$$F^{n+1}(\bot)(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(\mathtt{x}) = 0 \\ \sigma[\mathtt{x} \mapsto 0] & \text{falls } \sigma(\mathtt{x}) \in \{\, 1, \ldots, n \,\} \\ \bot & \text{sonst} \end{cases}$$

Die Glieder der Folge $F^n(\bot)$ sind mit wachsendem n an immer mehr Stellen definiert. Keines der Elemente der Folge ist selbst ein Fixpunkt, erst der „Grenzwert“ der Folge für $n \to \infty$, im Beispiel 23 die Funktion f^*.

Beispiel 26. Die Fixpunktiteration tritt auch in der Analysis auf, beispielsweise beim Newton-Verfahren zur Nullstellenberechnung für eine stetig differenzierbare Funktion $f :: \mathbb{R} \Rightarrow \mathbb{R}$. Dazu bildet man das Funktional $F :: \mathbb{R} \Rightarrow \mathbb{R}$ mit

$$F(x) = x - \frac{f(x)}{f'(x)}$$

Für alle Nullstellen x^* von f, an denen die Ableitung nicht verschwindet ($f'(x^*) \neq 0$), gilt:

$$F(x^*) = x^* - \frac{f(x^*)}{f'(x^*)} = x^* - \frac{0}{f'(x^*)} = x^*$$

d.h., x^* ist ein Fixpunkt von F. Um nun eine Nullstelle anzunähern, berechnet man die Folge $F^n(x_0)$ für einen (geeigneten) Startwert x_0. Der gesuchte Fixpunkt ist dann der (analytische) Grenzwert dieser Folge, falls sie konvergiert.

Kehren wir nun zur Definition von $\mathcal{D}[\![\_]\!]$ zurück. Nehmen wir vorläufig an, dass wir einen Fixpunktoperator FIX :: $((\Sigma \rightharpoonup \Sigma) \Rightarrow (\Sigma \rightharpoonup \Sigma)) \Rightarrow (\Sigma \rightharpoonup \Sigma)$ zur Verfügung haben, der den Grenzwert der Fixpunktiteration des übergebenen Funktionals berechnet. Im folgenden Abschnitt über Fixpunkttheorie werden wir sehen, dass es einen solchen für unsere Zwecke ausreichenden Operator gibt.

Definition 32 (Denotationale Semantik).
Die denotationale Semantik $\mathcal{D}[\![\_]\!]$ für While ist definiert durch

$$\begin{aligned}
\mathcal{D}[\![\texttt{skip}]\!] &= id \\
\mathcal{D}[\![x \texttt{ := } a]\!] &= \lambda\sigma.\ \sigma[x \mapsto \mathcal{A}[\![a]\!]\,\sigma] \\
\mathcal{D}[\![c_1 \texttt{;}\ c_2]\!] &= \mathcal{D}[\![c_2]\!] \circ \mathcal{D}[\![c_1]\!] \\
\mathcal{D}[\![\texttt{if (}b\texttt{) then } c_1 \texttt{ else } c_2]\!] &= \mathrm{IF}\,(\mathcal{B}[\![b]\!],\ \mathcal{D}[\![c_1]\!],\ \mathcal{D}[\![c_2]\!]) \\
\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!] &= \mathrm{FIX}\,(\lambda f.\ \mathrm{IF}\,(\mathcal{B}[\![b]\!],\ f \circ \mathcal{D}[\![c]\!],\ id))
\end{aligned}$$

mit folgenden Hilfsfunktionen:

- $f \circ g$ ist die normale Funktionskomposition, nur dass sie $\bot$ propagiert:

$$(f \circ g)(\sigma) = \begin{cases} \bot & \text{falls } g(\sigma) = \bot \\ f(g(\sigma)) & \text{sonst} \end{cases}$$

- IF :: $((\Sigma \Rightarrow \mathbb{B}) \times (\Sigma \rightharpoonup \Sigma) \times (\Sigma \rightharpoonup \Sigma)) \Rightarrow (\Sigma \rightharpoonup \Sigma)$ wählt abhängig vom ersten Parameter einen der beiden anderen aus:

$$\mathrm{IF}\,(p,\ f,\ g)\,(\sigma) = \begin{cases} f(\sigma) & \text{falls } p(\sigma) = \mathbf{tt} \\ g(\sigma) & \text{falls } p(\sigma) = \mathbf{ff} \end{cases}$$

Zum FIX-Operator noch ein paar Überlegungen: Von der Big-Step-Semantik wissen wir, dass die Programme `while (`b`) do` c und `if (`b`) then` c`; while (`b`) do` c `else skip` äquivalent sind (Lem. 1). Demnach sollten auch die denotationalen Bedeutungen der beiden Programme gleich sein:

$$\begin{aligned}
\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!] &= \mathcal{D}[\![\texttt{if (}b\texttt{) then } c\texttt{; while (}b\texttt{) do } c \texttt{ else skip}]\!] \\
&= \mathrm{IF}\,(\mathcal{B}[\![b]\!],\ \mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!] \circ \mathcal{D}[\![c]\!],\ id)
\end{aligned}$$

Dies entspricht der versuchten Definitionsgleichung für $\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!]$ am Anfang dieses Kapitels. Sie drückt aus, dass $\mathcal{D}[\![\texttt{while (}b\texttt{) do } c]\!]$ ein Fixpunkt des Funktionals $\lambda f.\ \mathrm{IF}\,(\mathcal{B}[\![b]\!],\ f \circ \mathcal{D}[\![c]\!],\ id)$ sein soll. Wenn der Fixpunktoperator also wirklich einen Fixpunkt berechnet, dann erfüllt die denotationale Semantik-Definition diese Gleichung.

Die möglicherweise verschiedenen Fixpunkte des Funktionals unterscheiden sich nur auf Zuständen, für die das Programm nicht terminiert. Die fehlende Information über einen Endzustand in der Big-Step-Semantik äußert sich als Unterspezifikation in der denotationalen Semantik. Mit der Wahl des „undefiniertesten“ Fixpunkt drückt Undefiniertheit also Nichttermination aus. Dies entspricht der Nicht-Ableitbarkeit in der Big-Step-Semantik.

Beispiel 27. Für `while (x <= 0) do skip` ergibt sich folgendes Funktional F:

$$F(f) = \mathrm{IF}\,(\mathcal{B}[\![\texttt{x <= 0}]\!],\ f \circ \mathcal{D}[\![\texttt{skip}]\!],\ id) = \lambda\sigma.\ \begin{cases}\sigma & \text{falls } \sigma(\texttt{x}) > 0\\ f(\sigma) & \text{falls } \sigma(\texttt{x}) \leq 0\end{cases}$$

Die Fixpunktiteration von F ergibt:

$$\begin{aligned}
F^0(\bot) &= \bot\\
F^1(\bot)(\sigma) &= \begin{cases}\sigma & \text{falls } \sigma(\texttt{x}) > 0\\ \bot(\sigma) & \text{falls } \sigma(\texttt{x}) \leq 0\end{cases}\\
F^2(\bot)(\sigma) &= \begin{cases}\sigma & \text{falls } \sigma(\texttt{x}) > 0\\ F^1(\bot)(\sigma) & \text{falls } \sigma(\texttt{x}) \leq 0\end{cases} = \begin{cases}\sigma & \text{falls } \sigma(\texttt{x}) > 0\\ \bot(\sigma) & \text{falls } \sigma(\texttt{x}) \leq 0\end{cases} = F^1(\bot)(\sigma)\\
F^3(\bot)(\sigma) &= F(F^2(\bot))(\sigma) = F(F^1(\bot))(\sigma) = F^2(\bot)(\sigma) = F^1(\bot)(\sigma)
\end{aligned}$$

Mittels Induktion erhält man allgemein $F^{n+1}(\bot) = F^1(\bot)$ für alle n. Die Folge der Fixpunktiteration besteht nur aus den Elementen $F^0(\bot)$ und $F^1(\bot)$, der Grenzwert ist damit $F^1(\bot)$. Es gilt also:

$$\mathcal{D}[\![\texttt{while (x <= 0) do skip}]\!] = F^1(\bot) = \lambda\sigma.\ \begin{cases}\sigma & \text{falls } \sigma(\texttt{x}) > 0\\ \bot & \text{sonst}\end{cases}$$