7 Axiomatische Semantik

Operationale und denotationale Semantiken legen die Bedeutung eines Programms direkt fest — als Ableitungsbaum, maximale Ableitungsfolge oder partielle Funktion auf den Zuständen. Dies ist eine Art *interner Spezifikation* der Semantik eines Programms: Man beschreibt, was genau die Bedeutungsobjekte sind. Dies ist ein geeigneter Ansatz, wenn man diese Bedeutungsobjekte selbst weiter verwenden möchte – zum Beispiel, um Programmoptimierungen in einem Compiler zu rechtfertigen oder Meta-Eigenschaften der Sprache (wie Typsicherheit) zu beweisen.

Möchte man aber Eigenschaften eines konkreten Programms verifizieren, interessiert man sich nicht für das Bedeutungsobjekt selbst, sondern nur dafür, ob es gewisse Eigenschaften besitzt. Diese Eigenschaften kann man natürlich – wenn die Semantik ausreichende Informationen für die Eigenschaft enthält – von dem Objekt selbst ablesen bzw. beweisen, dies ist aber in vielen Fällen umständlich. Beispielsweise ist man nur an einem Teil der Ergebnisse eines Algorithmus interessiert, also an den Ergebniswerten einzelner Variablen, der Inhalt der anderen (Hilfs-)Variablen ist aber für die spezifische Eigenschaft irrelevant. Der Ableitungsbaum oder die partielle Funktion beinhalten aber auch die gesamte Information über diese Hilfsvariablen, wodurch diese Objekte unnötig viel Information enthalten und damit auch bei einer automatisierten Programmverifikation den Aufwand unnötig erhöhen würden.

Die axiomatische Semantik verfolgt deswegen den Ansatz einer *externen Spezifikation*: Sie legt (mit einem Regelsystem – axiomatisch) fest, welche Eigenschaften das Bedeutungsobjekt jedes Programms haben soll – ohne explizit ein solches Bedeutungsobjekt zu konstruieren. Dadurch werden sämtliche Details einer solchen Konstruktion ausgeblendet (z.B. die Ableitungsfolgen bei der Small-Step- oder die Fixpunktiteration bei der denotationalen Semantik), die für den Nachweis von Programmeigenschaften nur hinderlich sind. Umgekehrt läuft man bei der Erstellung einer axiomatischen Semantik natürlich immer Gefahr, widersprüchliche Bedingungen an die Bedeutungsobjekte zu stellen. Deswegen sollte man zu einer externen Spezifikation immer ein Modell konstruieren, zu einer axiomatischen Semantik also eine operationale oder denotationale finden und die Korrektheit zeigen.

Bei jeder Semantik muss man sich entscheiden, welche Art von Programmeigenschaften man mit ihr ausdrücken können soll. Beispielsweise sind denotationale und Big-Step-Semantik ungeeignet, um die Laufzeit eines Programms, d.h. die Zahl seiner Ausführungsschritte, zu messen. Damit können wir auch keine Eigenschaften über die Laufzeit eines Programms mit diesen Semantiken nachweisen. Zwei wichtige Arten von Eigenschaften lassen sich aber ausdrücken:

Partielle Korrektheit *Falls* das Programm terminiert, dann gilt eine bestimmte Beziehung zwischen Anfangs- und Endzustand.

Totale Korrektheit Das Programm terminiert und es gibt eine bestimmte Beziehung zwischen Anfangs- und Endzustand.

In diesem Kapitel werden wir uns auf partielle Korrektheitsaussagen beschränken.

Beispiel 37. Das niemals terminierende Programm `while (true) do skip` ist korrekt bezüglich allen partiellen Korrektheitseigenschaften. Es ist jedoch für keine solche Beziehung zwischen Anfangs- und Endzustand total korrekt.

7.1 Ein Korrektheitsbeweis mit der denotationalen Semantik

Korrektheitsbeweise von Programmen lassen sich nicht nur mit einer axiomatischer Semantik führen. Auch operationale und denotationale Semantik sind dafür theoretisch vollkommen ausreichend. Dies wollen wir in diesem Abschnitt am Beispiel der partiellen Korrektheit der Fakultät über die denotationale Semantik vorführen: Sei

$$P = \texttt{y := 1; while (not (x == 1)) do (y := y * x; x := x - 1)}$$

Wir wollen zeigen, dass dieses Programm – sofern es terminiert – in y die Fakultät des Anfangswertes in x speichert. Dies lässt sich als eine Eigenschaft $\varphi(f)$ auf den Bedeutungsobjekten f aus $\Sigma \rightharpoonup \Sigma$ formulieren:

$$\varphi(f) = (\forall \sigma\ \sigma'.\ f(\sigma) = \sigma' \implies \sigma'(\mathtt{y}) = (\sigma(\mathtt{x}))! \wedge \sigma(\mathtt{x}) > 0)$$

P ist also korrekt, wenn $\varphi(\mathcal{D}\,[\![P]\!])$ gilt. Da $\mathcal{D}\,[\![P]\!]$ einen Fixpunktoperator enthält, brauchen wir, um dies zu zeigen, noch das Konzept der Fixpunktinduktion.

Definition 45 (Zulässiges Prädikat). Sei $(D, \sqsubseteq)$ eine ccpo. Ein Prädikat $\varphi :: D \Rightarrow \mathbb{B}$ heißt *zulässig*, wenn für alle Ketten Y in $(D, \sqsubseteq)$ gilt: Wenn $\varphi(d) = \mathbf{tt}$ für alle $d \in Y$ gilt, dann auch $\varphi\left(\bigsqcup Y\right) = \mathbf{tt}$.

Zulässige Prädikate sind also stetig auf Ketten, auf denen sie überall gelten. Für zulässige Prädikate gibt es folgendes Induktionsprinzip:

Theorem 48 (Fixpunktinduktion, Scott-Induktion). Sei $(D, \sqsubseteq)$ eine ccpo, $f :: D \Rightarrow D$ eine monotone und kettenstetige Funktion, und sei $\varphi :: D \Rightarrow \mathbb{B}$ zulässig. Wenn für alle $d \in D$ gilt, dass aus $\varphi(d) = \mathbf{tt}$ bereits $\varphi(f(d)) = \mathbf{tt}$ folgt, dann gilt auch $\varphi(\mathrm{FIX}\,(f)) = \mathbf{tt}$.

Beweis. Nach Thm. 33 gilt $\mathrm{FIX}\,(f) = \bigsqcup \{\, f^n(\bot) \mid n \in \mathbb{N} \,\}$. Da φ zulässig ist, genügt es also zu zeigen, dass $\varphi(f^n(\bot))$ für alle $n \in \mathbb{N}$ gilt. Beweis durch Induktion über n:

- Basisfall $n = 0$: Nach Def. 45 ($\emptyset$ ist eine Kette) gilt:

$$\varphi(f^0(\bot)) = \varphi(\bot) = \varphi\left(\bigsqcup \emptyset\right) = \mathbf{tt}$$

- Induktionsschritt $n+1$: Induktionsannahme: $\varphi(f^n(\bot)) = \mathbf{tt}$. Zu zeigen: $\varphi(f^{n+1}(\bot)) = \mathbf{tt}$.
  Aus der Induktionsannahme folgt nach Voraussetzung, dass $\mathbf{tt} = \varphi(f(f^n(\bot))) = \varphi(f^{n+1}(\bot))$. □

Nun zurück zu unserem Beispiel. Mit Thm. 48 können wir jetzt $\varphi\,(\mathcal{D}\,[\![P]\!])$ beweisen. Nach Definition gilt:

$$\mathcal{D}\,[\![P]\!]\,\sigma = \mathcal{D}\,[\![\texttt{while (not (x == 1)) do (y := y * x; x := x - 1)}]\!]\,(\sigma[\mathtt{y} \mapsto 1])$$

und damit

$$\begin{aligned}\varphi(\mathcal{D}\,[\![P]\!]) &= \varphi(\lambda\sigma.\ \mathcal{D}\,[\![\texttt{while (not (x == 1)) do (y := y * x; x := x - 1)}]\!]\,(\sigma[\mathtt{y} \mapsto 1]))\\ &= \varphi(\lambda\sigma.\ \mathrm{FIX}\,(F)\,(\sigma[\mathtt{y} \mapsto 1])\end{aligned}$$

wobei $F(g) = \mathrm{IF}\,(\mathcal{B}\,[\![\texttt{not (x == 1)}]\!],\ g \circ \mathcal{D}\,[\![\texttt{y := y * x; x := x - 1}]\!],\ id)$.

Für die Fixpunkt-Induktion ist φ jedoch zu schwach, da φ nicht unter dem Funktional F erhalten bleibt. Wir brauchen dafür also eine stärkere Eigenschaft φ':

$$\varphi'(f) = (\forall \sigma\ \sigma'.\ f(\sigma) = \sigma' \implies \sigma'(\mathtt{y}) = \sigma(\mathtt{y}) \cdot (\sigma(\mathtt{x}))! \wedge \sigma(\mathtt{x}) > 0)$$

Wenn $\varphi'(\mathrm{FIX}\,(F))$ gilt, dann gilt auch $\varphi(\lambda\sigma.\ \mathrm{FIX}\,(F)\,(\sigma[\mathtt{y} \mapsto 1]))$. Für die Anwendung der Fixpunktinduktion (Thm. 48) auf φ' und F müssen wir noch folgendes zeigen:

- φ' ist zulässig (Induktionsanfang):
  Sei Y beliebige Kette in $(\Sigma \rightharpoonup \Sigma, \sqsubseteq)$ mit $\varphi'(f) = \mathbf{tt}$ für alle $f \in Y$. Zu zeigen: $\varphi'(\bigsqcup Y) = \mathbf{tt}$.

  Seien also σ, σ' beliebig mit $(\bigsqcup Y)\,\sigma = \sigma'$. Dann gibt es nach Definition ein $f \in Y$ mit $f(\sigma) = \sigma'$. Da $\varphi'(f) = \mathbf{tt}$, gilt $\sigma'(\mathtt{y}) = \sigma(\mathtt{y}) \cdot (\sigma(\mathtt{x}))! \wedge \sigma(\mathtt{x}) > 0$, was zu zeigen ist.

- Wenn $\varphi'(f) = \mathtt{tt}$, dann auch $\varphi'(F(f))$ (Induktionsschritt):
  Seien also σ, σ' mit $F(f)(\sigma) = \sigma'$. Zu zeigen: $\sigma'(\mathtt{y}) = \sigma(\mathtt{y}) \cdot (\sigma(\mathtt{x}))!$.
  Beweis durch Fallunterscheidung über $\mathcal{B}\,[\![\mathtt{not\ (x\ ==\ 1)}]\!]\,\sigma$
  - Fall $\mathcal{B}\,[\![\mathtt{not\ (x\ ==\ 1)}]\!]\,\sigma = \mathtt{tt}$: Dann gilt

$$\begin{aligned} F(f)(\sigma) &= (f \circ \mathcal{D}\,[\![\mathtt{y\ :=\ y\ *\ x;\ x\ :=\ x\ -\ 1}]\!])(\sigma) \\ &= (f \circ \mathcal{D}\,[\![\mathtt{x\ :=\ x\ -\ 1}]\!] \circ \mathcal{D}\,[\![\mathtt{y\ :=\ y\ *\ x}]\!])(\sigma) \\ &= (f \circ \mathcal{D}\,[\![\mathtt{x\ :=\ x\ -\ 1}]\!])(\sigma[\mathtt{y} \mapsto \sigma(\mathtt{y}) \cdot \sigma(\mathtt{x})]) \\ &= f(\sigma[\mathtt{y} \mapsto \sigma(\mathtt{y}) \cdot \sigma(\mathtt{x}),\ \mathtt{x} \mapsto \sigma(\mathtt{x}) - 1]) = \sigma' \end{aligned}$$

  Wegen $\varphi'(f) = \mathtt{tt}$ gilt damit $(\sigma[\mathtt{y} \mapsto \sigma(\mathtt{y}) \cdot \sigma(\mathtt{x}),\ \mathtt{x} \mapsto \sigma(\mathtt{x}) - 1])(\mathtt{x}) > 0$ und damit $\sigma(\mathtt{x}) > 1$, also auch insbesondere $\sigma(\mathtt{x}) > 0$. Außerdem gilt:

$$\begin{aligned} \sigma'(\mathtt{y}) &= (\sigma[\mathtt{y} \mapsto \sigma(\mathtt{y}) \cdot \sigma(\mathtt{x}),\ \mathtt{x} \mapsto \sigma(\mathtt{x}) - 1])(\mathtt{y}) \cdot ((\sigma[\mathtt{y} \mapsto \sigma(\mathtt{y}) \cdot \sigma(\mathtt{x}),\ \mathtt{x} \mapsto \sigma(\mathtt{x}) - 1])(\mathtt{x}))! \\ &= (\sigma(\mathtt{y}) \cdot \sigma(\mathtt{x})) \cdot (\sigma(\mathtt{x}) - 1)! = \sigma(\mathtt{y}) \cdot (\sigma(\mathtt{x}))! \end{aligned}$$

  - Fall $\mathcal{B}\,[\![\mathtt{not\ (x\ ==\ 1)}]\!]\,\sigma = \mathbf{ff}$:
    Wegen $F(f)(\sigma) = id(f)(\sigma) = f(\sigma)$ folgt die Behauptung aus $\varphi'(f)$.

Damit gilt also auch $\varphi'(\mathrm{FIX}\,(F)) = \mathtt{tt}$. Demnach auch $\varphi(\mathcal{D}\,[\![P]\!])$.

7.2 Zusicherungen

Nach diesem Ausflug in die denotationale Semantik kommen wir nun wirklich zur axiomatischen Beschreibung der Bedeutungsobjekte zurück. Wir konzentrieren uns hierbei auf Aussagen über die partielle Korrektheit eines Programms, die durch Zusicherungen ausgedrückt werden.

Definition 46 (Zusicherung, Hoare-Tripel, Vorbedingung, Nachbedingung). Eine *Zusicherung* (*Hoare-Tripel*) ist ein Tripel $\{\,P\,\}\,c\,\{\,Q\,\}$, wobei das Zustandsprädikat P die *Vorbedingung* und das Zustandsprädikat Q die *Nachbedingung* der Anweisung c ist. Zustandsprädikate sind Funktionen des Typs $\Sigma \Rightarrow \mathbb{B}$.

Eine Zusicherung ist zuerst einmal also nur eine Notation für zwei Prädikate P und Q und eine Anweisung c, der wir im Folgenden noch eine Semantik geben wollen. Intuitiv soll eine Zusicherung $\{\,P\,\}\,c\,\{\,Q\,\}$ aussagen: Wenn das Prädikat P im Anfangszustand σ erfüllt ist, dann wird – sofern die Ausführung von c im Zustand σ terminiert – das Prädikat Q im Endzustand dieser Ausführung erfüllt sein. Terminiert die Ausführung von c im Anfangszustand σ nicht, so macht die Zusicherung keine Aussage.

Beispiel 38. Für das Fakultätsprogramm aus Kap. 7.1 könnte man folgende Zusicherung schreiben, um die Korrektheit auszudrücken.

$$\{\,\mathtt{x} = n\,\}\ \mathtt{y\ :=\ 1;\ while\ (not\ (x\ ==\ 1))\ do\ (y\ :=\ y\ *\ x;\ x\ :=\ x\ -\ 1)}\ \{\,\mathtt{y} = n! \land n > 0\,\}$$

Dabei ist n eine *logische Variable*, die nicht im Programm vorkommt. Sie wird in der Vorbedingung dazu verwendet, den Anfangswert von `x` zu speichern, damit er in der Nachbedingung noch verfügbar ist. Würde man stattdessen

$$\{\,\mathtt{tt}\,\}\ \mathtt{y\ :=\ 1;\ while\ (not\ (x\ ==\ 1))\ do\ (y\ :=\ y\ *\ x;\ x\ :=\ x\ -\ 1)}\ \{\,\mathtt{y} = \mathtt{x}! \land \mathtt{x} > 0\,\}$$

schreiben, hätte dies eine andere Bedeutung: Dann müsste im Endzustand der Wert von `y` der Fakultät des *Endzustands*werts von `x` entsprechen. Technisch unterscheiden wir nicht zwischen „echten“ und logischen Variablen, wir speichern beide im Zustand. Da logische Variablen aber nicht im Programm vorkommen, stellen sie keine wirkliche Einschränkung der Programmeigenschaften dar und haben im Endzustand immer noch den gleichen Wert wie am Anfang.

Formal gesehen sind Vor- und Nachbedingungen in Zusicherungen Prädikate auf Zuständen, d.h. vom Typ $\Sigma \Rightarrow \mathbb{B}$. Korrekt hätte die Zusicherung in Beispiel 38 also wie folgt lauten müssen:

$$\{\, \lambda\sigma.\ \sigma(\mathtt{x}) = \sigma(n) \,\} \ldots \{\, \lambda\sigma.\ \sigma(\mathtt{y}) = (\sigma(n))! \wedge \sigma(n) > 0 \,\}$$

Diese umständliche Notation macht aber Zusicherungen nur unnötig schwer lesbar. Innerhalb von Vor- und Nachbedingungen lassen wir deshalb das $\lambda\sigma.$ weg und schreiben nur x statt $\sigma(x)$.

7.3 Inferenzregeln für While

Eine axiomatische Semantik gibt man wie eine Big-Step-Semantik als eine Menge von Inferenzregeln an. Diese Regeln definieren die Ableitbarkeit einer Zusicherung $\{\, P \,\}\, c \,\{\, Q \,\}$, geschrieben als $\vdash \{\, P \,\}\, c \,\{\, Q \,\}$. Dies entspricht einem formalen Beweissystem, mit dem man partiell korrekte Eigenschaften eines Programms nachweisen kann. Für While lauten die Regeln:

$$\textsc{Skip}_\textsc{P}\colon \vdash \{\, P \,\}\, \mathtt{skip} \,\{\, P \,\} \qquad \textsc{Ass}_\textsc{P}\colon \vdash \{\, P[x \mapsto \mathcal{A}\,[\![a]\!]] \,\}\, x\ \mathtt{:=}\ a \,\{\, P \,\}$$

$$\textsc{Seq}_\textsc{P}\colon \frac{\vdash \{\, P \,\}\, c_1 \,\{\, Q \,\} \qquad \vdash \{\, Q \,\}\, c_2 \,\{\, R \,\}}{\vdash \{\, P \,\}\, c_1\mathtt{;}\ c_2 \,\{\, R \,\}}$$

$$\textsc{If}_\textsc{P}\colon \frac{\vdash \{\, \lambda\sigma.\ \mathcal{B}\,[\![b]\!]\,\sigma \wedge P(\sigma) \,\}\, c_1 \,\{\, Q \,\} \qquad \vdash \{\, \lambda\sigma.\ \neg\mathcal{B}\,[\![b]\!]\,\sigma \wedge P(\sigma) \,\}\, c_2 \,\{\, Q \,\}}{\vdash \{\, P \,\}\, \mathtt{if}\ \mathtt{(}b\mathtt{)}\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \,\{\, Q \,\}}$$

$$\textsc{While}_\textsc{P}\colon \frac{\vdash \{\, \lambda\sigma.\ \mathcal{B}\,[\![b]\!]\,\sigma \wedge I(\sigma) \,\}\, c \,\{\, I \,\}}{\vdash \{\, I \,\}\, \mathtt{while}\ \mathtt{(}b\mathtt{)}\ \mathtt{do}\ c \,\{\, \lambda\sigma.\ \neg\mathcal{B}\,[\![b]\!]\,\sigma \wedge I(\sigma) \,\}}$$

$$\textsc{Cons}_\textsc{P}\colon \frac{P \implies P' \qquad \vdash \{\, P' \,\}\, c \,\{\, Q' \,\} \qquad Q' \implies Q}{\vdash \{\, P \,\}\, c \,\{\, Q \,\}}$$

wobei $P[x \mapsto f]$ definiert sei durch

$$(P[x \mapsto f])(\sigma) = P(\sigma[x \mapsto f(\sigma)])$$

und $P \implies P'$ für $\forall\sigma.\ P(\sigma) \implies P'(\sigma)$ steht.

Die Regel für `skip` ist einleuchtend: Was vorher galt, muss auch nachher gelten. Die Regel $\textsc{Ass}_\textsc{P}$ für Zuweisungen x `:=` a nimmt an, dass vor Ausführung im Anfangszustand σ das Prädikat P für den Zustand $\sigma[x \mapsto \mathcal{A}\,[\![a]\!]\,\sigma]$ gilt. Dann muss auch der Endzustand P erfüllen – der in unserer operationalen Semantik eben genau $\sigma[x \mapsto \mathcal{A}\,[\![a]\!]\,\sigma]$ ist.

Neben diesen beiden Axiomen bzw. Axiomschemata des Regelsystems sind die Regeln für die anderen Konstrukte Inferenzregeln, die die Ableitung einer Zusicherung einer zusammengesetzten Anweisung aus den einzelnen Teilen bestimmt. Für die Hintereinanderausführung $c_1\mathtt{;}\ c_2$ gilt: Die Zusicherung $\{\, P \,\}\, c_1\mathtt{;}\ c_2 \,\{\, R \,\}$ ist ableitbar, wenn es ein Prädikat Q gibt, das von c_1 unter der Vorbedingung P garantiert wird und unter dessen Voraussetzung c_2 die Nachbedingung R garantieren kann. In der Regel $\textsc{While}_\textsc{P}$ ist I eine Invariante des Schleifenrumpfes, die zu Beginn gelten muss und – falls die Schleife terminiert – auch am Ende noch gilt. Da partielle Korrektheit nur Aussagen über terminierende Ausführungen eines Programms macht, muss an deren Endzustand auch die negierte Schleifenbedingung gelten.

Die letzte Regel, die *Folgeregel* (*rule of consequence*), erlaubt, Vorbedingungen zu verstärken und Nachbedingungen abzuschwächen. Erst damit kann man die anderen Inferenzregeln sinnvoll zusammensetzen.

Beispiel 39. Sei $P = \texttt{if (x == 5) then skip else x := 5}$. Dann gilt:

$$\dfrac{\dfrac{}{\vdash \{\, \mathcal{B}[\![\texttt{x == 5}]\!] \,\}\, \texttt{skip} \,\{\, \texttt{x} = 5 \,\}}\,\textsc{Skip}_\text{P} \qquad \dfrac{x \neq 5 \implies \texttt{tt} \qquad \dfrac{}{\vdash \{\, \texttt{tt} \,\}\, \texttt{x := 5} \,\{\, \texttt{x} = 5 \,\}}\,\textsc{Ass}_\text{P}}{\vdash \{\, \neg\mathcal{B}[\![\texttt{x == 5}]\!] \,\}\, \texttt{x := 5} \,\{\, \texttt{x} = 5 \,\}}\,\textsc{Cons}_\text{P}}{\vdash \{\, \texttt{tt} \,\}\, \texttt{if (x == 5) then skip else x := 5} \,\{\, \texttt{x} = 5 \,\}}\,\textsc{If}_\text{P}$$

Beispiel 40. Die Fakultätsberechnung mit einer Schleife (vgl. Kap. 7.1) ist partiell korrekt:

$$\{\, \texttt{x} = n \,\}\, \texttt{y := 1; while (not (x == 1)) do (y := y * x; x := x - 1)} \,\{\, \texttt{y} = n! \wedge n > 0 \,\}$$

$$\dfrac{\dfrac{}{\vdash \{\, \texttt{x} = n \,\}\, \texttt{y := 1} \,\{\, \texttt{x} = n \wedge \texttt{y} = 1 \,\}}\,\textsc{Ass}_\text{P} \qquad A}{\vdash \{\, \texttt{x} = n \,\}\, \texttt{y := 1; while (not (x == 1)) do (y := y * x; x := x - 1)} \,\{\, \texttt{y} = n! \wedge n > 0 \,\}}\,\textsc{Seq}_\text{P}$$

A: $$\dfrac{\texttt{x} = n \wedge \texttt{y} = 1 \implies I \qquad B \qquad \texttt{x} = 1 \wedge I \implies \texttt{y} = n! \wedge n > 0}{\{\, \texttt{x} = n \wedge \texttt{y} = 1 \,\}\, \texttt{while (not (x == 1)) do (y := y * x; x := x - 1)} \,\{\, \texttt{y} = n! \wedge n > 0 \,\}}\,\textsc{Cons}_\text{P}$$

wobei $I = \texttt{x} \leq 0 \vee (\texttt{y} \cdot \texttt{x}! = n! \wedge \texttt{x} \leq n)$ die Schleifeninvariante ist.

B: $$\dfrac{\dfrac{C \qquad \dfrac{}{\vdash \{\, I[\texttt{x} \mapsto \texttt{x} - 1] \,\}\, \texttt{x := x - 1} \,\{\, I \,\}}\,\textsc{Ass}_\text{P}}{\vdash \{\, \neg\mathcal{B}[\![\texttt{not (x == 1)}]\!] \wedge I \,\}\, \texttt{y := y * x; x := x - 1} \,\{\, I \,\}}\,\textsc{Seq}_\text{P}}{\vdash \{\, I \,\}\, \texttt{while (not (x == 1)) do (y := y * x; x := x - 1)} \,\{\, \texttt{x} = 1 \wedge I \,\}}\,\textsc{While}_\text{P}$$

C: $$\dfrac{D \qquad \dfrac{}{\vdash \{\, (I[\texttt{x} \mapsto \texttt{x} - 1])[\texttt{y} \mapsto \texttt{y} \cdot \texttt{x}] \,\}\, \texttt{y := y * x} \,\{\, I[\texttt{x} \mapsto \texttt{x} - 1] \,\}}\,\textsc{Ass}_\text{P}}{\vdash \{\, x \neq 1 \wedge I \,\}\, \texttt{y := y * x} \,\{\, I[\texttt{x} \mapsto \texttt{x} - 1] \,\}}\,\textsc{Cons}_\text{P}$$

D: $x \neq 1 \wedge I \implies (I[\texttt{x} \mapsto \texttt{x} - 1])[\texttt{y} \mapsto \texttt{y} \cdot \texttt{x}]$, da:

$$\begin{aligned}
((I[\texttt{x} \mapsto \texttt{x} - 1])[\texttt{y} \mapsto \texttt{y} \cdot \texttt{x}])(\sigma) &= (I[\texttt{x} \mapsto \texttt{x} - 1])(\sigma[\texttt{y} \mapsto \sigma(\texttt{y}) \cdot \sigma(\texttt{x})]) \\
&= I(\sigma[\texttt{y} \mapsto \sigma(\texttt{y}) \cdot \sigma(\texttt{x}),\ \texttt{x} \mapsto (\sigma[\texttt{y} \mapsto \sigma(\texttt{y}) \cdot \sigma(\texttt{x})])(\texttt{x}) - 1]) \\
&= I(\sigma[\texttt{y} \mapsto \sigma(\texttt{y}) \cdot \sigma(\texttt{x}),\ \texttt{x} \mapsto \sigma(\texttt{x}) - 1]) \\
&= \sigma(\texttt{x}) - 1 \leq 0 \vee ((\sigma(\texttt{y}) \cdot \sigma(\texttt{x})) \cdot (\sigma(\texttt{x}) - 1)! = \sigma(n)! \wedge \sigma(\texttt{x}) - 1 \leq \sigma(n)) \\
&= \sigma(\texttt{x}) \leq 1 \vee (\sigma(\texttt{y}) \cdot (\sigma(\texttt{x}))! = \sigma(n)! \wedge \sigma(\texttt{x}) < \sigma(n))
\end{aligned}$$

Bemerkenswert ist, dass für den Korrektheitsbeweis im Beispiel 40 keinerlei Induktion (im Gegensatz zu Kap. 7.1 mit der denotationalen Semantik) gebraucht wurde. Stattdessen musste lediglich für die Ableitung eine Regel nach der anderen angewandt werden – die Essenz des Beweises steckt in der Invariante und in den Implikationen der $\textsc{Cons}_\text{P}$-Regel. Damit eignet sich ein solches Beweissystem aus axiomatischen Regeln zur Automatisation: Hat man ein Programm, an dem die Schleifen mit Invarianten annotiert sind, so kann man mit einem *Verifikationsbedingungsgenerator* (*VCG*) automatisch aus den Implikationen der notwendigen $\textsc{Cons}_\text{P}$-Regelanwendungen sogenannte Verifikationsbedingungen generieren lassen, die dann bewiesen werden müssen. Da diese Verifikationsbedingungen Prädikate auf *einem* Zustand sind, braucht man sich für deren Lösung nicht mehr um die Programmiersprache, Semantik oder ähnliches kümmern, sondern kann allgemein verwendbare Entscheidungsverfahren anwenden.

7.4 Korrektheit der axiomatischen Semantik

Wie schon in der Einleitung erwähnt, sollte man zeigen, dass das Regelsystem zur Ableitbarkeit von Zusicherungen nicht widersprüchlich ist, d.h., dass es eine operationale oder denotationale Semantik gibt, deren Bedeutungsobjekte die ableitbaren Zusicherungen erfüllen. Gleichbedeutend damit ist, dass man für eine operationale oder denotationale Semantik beweist, dass die Regeln der axiomatischen Semantik korrekt sind: Wenn $\vdash \{\, P \,\}\, c \,\{\, Q \,\}$, dann gilt auch Q auf allen Endzuständen nach Ausführung von c in Startzuständen, die P erfüllen.

Definition 47 (Gültigkeit). Eine Zusicherung $\{\, P \,\}\, c \,\{\, Q \,\}$ ist *gültig* ($\models \{\, P \,\}\, c \,\{\, Q \,\}$), wenn für alle σ, σ' mit $\langle c,\ \sigma\rangle \Downarrow \sigma'$ gilt: Aus $P(\sigma)$ folgt $Q(\sigma')$.

Theorem 49 (Korrektheit der axiomatischen Semantik).
Wenn $\vdash \{\, P \,\}\, c \,\{\, Q \,\}$, dann $\models \{\, P \,\}\, c \,\{\, Q \,\}$.

Beweis. Beweis durch Regelinduktion über $\vdash \{\, P \,\}\, c \,\{\, Q \,\}$.

- Fall $\textsc{Skip}_{\mathrm{P}}$: Zu zeigen: $\models \{\, P \,\}\, \texttt{skip} \,\{\, P \,\}$.

  Seien σ, σ' beliebig mit $P(\sigma)$ und $\langle \texttt{skip},\ \sigma\rangle \Downarrow \sigma'$. Mit Regelinversion ($\textsc{Skip}_{\mathrm{BS}}$) auf $\langle \texttt{skip},\ \sigma\rangle \Downarrow \sigma'$ folgt $\sigma' = \sigma$, damit gilt auch $P(\sigma')$, was zu zeigen war.

- Fall $\textsc{Ass}_{\mathrm{P}}$: Zu zeigen: $\models \{\, P[x \mapsto \mathcal{A}\,[\![a]\!]] \,\}\, x\ \texttt{:=}\ a \,\{\, P \,\}$.

  Seien σ, σ' beliebig mit $P(\sigma[x \mapsto \mathcal{A}\,[\![a]\!]\,\sigma)$ und $\langle x\ \texttt{:=}\ a,\ \sigma\rangle \Downarrow \sigma'$. Zu zeigen: $P(\sigma')$.
  Mit Regelinversion ($\textsc{Ass}_{\mathrm{BS}}$) folgt $\sigma' = \sigma[x \mapsto \mathcal{A}\,[\![a]\!]\,\sigma]$ und daraus die Behauptung $P(\sigma')$.

- Fall $\textsc{Seq}_{\mathrm{P}}$: Induktionsannahmen: $\models \{\, P \,\}\, c_1 \,\{\, Q \,\}$ und $\models \{\, Q \,\}\, c_2 \,\{\, R \,\}$. Zu zeigen: $\models \{\, P \,\}\, c_1\texttt{;}\ \ c_2 \,\{\, R \,\}$.

  Seien σ, σ' beliebig mit $P(\sigma)$ und $\langle c_1\texttt{;}\ c_2,\ \sigma\rangle \Downarrow \sigma'$. Dann gibt es nach Regelinversion ($\textsc{Seq}_{\mathrm{BS}}$) ein σ^* mit $\langle c_1,\ \sigma\rangle \Downarrow \sigma^*$ und $\langle c_2,\ \sigma^*\rangle \Downarrow \sigma'$. Aus $\langle c_1,\ \sigma\rangle \Downarrow \sigma^*$ folgt mit der Induktionsannahme $\models \{\, P \,\}\, c_1 \,\{\, Q \,\}$ und $P(\sigma)$, dass $Q(\sigma^*)$. Zusammen mit $\langle c_2,\ \sigma^*\rangle \Downarrow \sigma'$ und der Induktionsannahme $\models \{\, Q \,\}\, c_2 \,\{\, R \,\}$ folgt $R(\sigma')$, was zu zeigen war.

- Fall $\textsc{If}_{\mathrm{P}}$: Induktionsannahmen: $\models \{\, \mathcal{B}\,[\![b]\!] \wedge P \,\}\, c_1 \,\{\, Q \,\}$ und $\models \{\, \neg\mathcal{B}\,[\![b]\!] \wedge P \,\}\, c_2 \,\{\, Q \,\}$.
  Zu zeigen: $\models \{\, P \,\}\, \texttt{if (}b\texttt{) then } c_1 \texttt{ else } c_2 \,\{\, Q \,\}$.

  Seien σ, σ' beliebig mit $P(\sigma)$ und $\langle \texttt{if (}b\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \Downarrow \sigma'$. Beweis von $Q(\sigma')$ durch Regelinversion:
  - Fall $\textsc{IfTT}_{\mathrm{BS}}$: Dann gilt $\mathcal{B}\,[\![b]\!]\,\sigma = \texttt{tt}$ und $\langle c_1,\ \sigma\rangle \Downarrow \sigma'$. Wegen $P(\sigma)$ gilt auch $(\mathcal{B}\,[\![b]\!] \wedge P)(\sigma)$ und mit der Induktionsannahme $\models \{\, \mathcal{B}\,[\![b]\!] \wedge P \,\}\, c_1 \,\{\, Q \,\}$ folgt aus $\langle c_1,\ \sigma\rangle \Downarrow \sigma'$, dass $Q(\sigma')$.
  - Fall $\textsc{IfFF}_{\mathrm{BS}}$: Analog mit der Induktionsannahme $\models \{\, \neg\mathcal{B}\,[\![b]\!] \wedge P \,\}\, c_2 \,\{\, Q \,\}$.

- Fall $\textsc{While}_{\mathrm{P}}$: Induktionsannahme I: $\models \{\, \mathcal{B}\,[\![b]\!] \wedge I \,\}\, c \,\{\, I \,\}$.
  Zu zeigen: $\models \{\, I \,\}\, \texttt{while (}b\texttt{) do } c \,\{\, \neg\mathcal{B}\,[\![b]\!] \wedge I \,\}$.

  Seien σ, σ' beliebig mit $\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \Downarrow \sigma'$. Zu zeigen: Wenn $I(\sigma)$, dann $\mathcal{B}\,[\![b]\!]\,\sigma' = \mathbf{ff}$ und $I(\sigma')$. Beweis durch Induktion über $\langle \texttt{while (}b\texttt{) do } c,\ \sigma\rangle \Downarrow \sigma'$:
  - Fall $\textsc{WhileFF}_{\mathrm{BS}}$: Dann $\sigma' = \sigma$ und $\mathcal{B}\,[\![b]\!]\,\sigma = \mathbf{ff}$. Daraus folgt direkt die Behauptung.
  - Fall $\textsc{WhileTT}_{\mathrm{BS}}$: Induktionsannahme II: $\mathcal{B}\,[\![b]\!]\,\sigma = \texttt{tt}$, $\langle c,\ \sigma\rangle \Downarrow \sigma^*$, und wenn $I(\sigma^*)$, dann $\mathcal{B}\,[\![b]\!]\,\sigma' = \mathbf{ff}$ und $I(\sigma')$. Zu zeigen: Wenn $I(\sigma)$, dann $\mathcal{B}\,[\![b]\!]\,\sigma' = \mathbf{ff}$ und $I(\sigma')$

    Mit der Induktionsannahme II genügt es, zu zeigen, dass aus $I(\sigma)$ auch $I(\sigma')$ folgt. Wegen $I(\sigma)$ und $\mathcal{B}\,[\![b]\!]\,\sigma = \texttt{tt}$ gilt $(\mathcal{B}\,[\![b]\!] \wedge P)(\sigma)$. Mit der Induktionsannahme I folgt aus $\langle c,\ \sigma\rangle \Downarrow \sigma^*$, dass $I(\sigma')$.

- Fall $\textsc{Cons}_{\mathrm{P}}$: Induktionsannahmen: $P \implies P'$, $\models \{\, P' \,\}\, c \,\{\, Q' \,\}$ und $Q' \implies Q$.
  Zu zeigen: $\models \{\, P \,\}\, c \,\{\, Q \,\}$.

  Seien σ, σ' beliebig mit $P(\sigma)$ und $\langle c,\ \sigma\rangle \Downarrow \sigma'$. Zu zeigen: $Q(\sigma')$.
  Wegen $P \implies P'$ folgt $P'(\sigma)$ aus $P(\sigma)$. Mit $\langle c,\ \sigma\rangle \Downarrow \sigma'$ folgt aus den Induktionsannahmen, dass $Q'(\sigma')$. Wegen $Q' \implies Q$ gilt damit auch $Q(\sigma')$. $\square$