Kapitel 6

Programme mit Schleifen

Wir betrachten jetzt eine idealisierte imperative Programmiersprache IMP mit zuweisbaren Variablen und Schleifen. Unser Ziel ist es, die Syntax und Semantik von IMP mit den bisher eingeführten Methoden zu definieren. Nachdem wir geklärt haben, welche Arten von semantischen Objekten den syntaktischen Objekte der Sprache zugeordnet werden sollen, ist die strukturell rekursive Definition der Denotationsfunktion bis auf eine Ausnahme Routine. Die Ausnahme ist durch Schleifen gegeben, deren Denotation mithilfe eines Fixpunktoperators definiert werden muss.

Die Standardmethode für die Definition der Semantik von Programmiersprachen ist die *operationale Methode*. Wir nutzen die Gelegenheit und demonstrieren die operationale Methode am Beispiel von IMP. Wir werden zeigen, dass die mit der operationalen Methode definierte *operationale Semantik* dieselbe Ein-Ausgabe-Relation liefert wie die zuvor mit der denotationalen Methode definierte *denotationale Semantik*.

Lesematerial

Unsere Sprache IMP entspricht im Wesentlichen der Sprache IMP in [Winskel], wir haben lediglich auf einige unwesentliche Sprachkonstrukte verzichtet. Da wir die mathematischen Grundlagen bereits gelegt haben, beginnen wir sofort mit der Definition der denotationalen Semantik. Aus demselben Grund arbeiten wir statt mit partiellen Funktionen $\Sigma \rightharpoonup \Sigma$ sofort mit totalen Funktionen $\Sigma \to \Sigma_\perp$.

[Winskel, Kapitel 2 und 5]

$n \in Con = \mathbb{Z}$		*Konstante*
$X \in Loc$		*Lokation*
$a \in Aexp = n \mid X \mid a_1 + a_2$		*Arithmetischer Ausdruck*
$b \in Bexp = a_1 \leq a_2$		*Boolscher Ausdruck*
$c \in Com =$		*Kommando*
	$X := a$	*Zuweisung*
	$\mid c_1; c_2$	*Sequentialisierung*
	$\mid$ if b then c_1 else c_2	*Konditional*
	$\mid$ while b do c	*Schleife*

Abbildung 6.1: Syntax von IMP

6.1 Syntax und denotationale Semantik von IMP

IMP ist eine idealisierte imperative Programmiersprache mit zuweisbaren Variablen und Schleifen, die man sich als Teilsprache von C oder Java vorstellen kann. Abbildung 6.1 definiert die Syntax von IMP relativ zu einer gegebenen, nichtleeren Menge *Loc*. Die Lokationen von IMP bezeichnet man auch als *Variablen*, und die Kommandos als *Anweisungen*. Um die folgenden Definitionen und Beweise einfach zu halten, haben wir IMP auf ein Minimum von Konstrukten beschränkt.

Damit wir Beispiele lesbarer schreiben können, vereinbaren wir einige notationale Abkürzungen:

$a - 1$	$\mapsto a + (-1)$	$a_1 < a_2$	$\mapsto (a_1 + 1) \leq a_2$
false	$\mapsto 1 \leq 0$	$a_1 \geq a_2$	$\mapsto a_2 \leq a_1$
true	$\mapsto 1 \leq 1$	$a_1 > a_2$	$\mapsto a_2 < a_1$
skip	$\mapsto X_0 := X_0$		

Bei X_0 handelt es sich um eine einmal festgelegte Lokation.

Als Beispiel betrachten wir ein Kommando, das das Produkt der an den Lokationen x und y abgelegten Zahlen berechnet und dieses an der Lokation z vermerkt:

```
z:=0;
while x≥1 do (x:=x-1; z:=z+y)
```

Mit Schleifen können wir divergierende Programme schreiben:

```
while true do skip
```


Für die Semantik von IMP benötigen wir Funktionen

$$\sigma \in \Sigma \stackrel{\text{def}}{=} Loc \to \mathbb{Z} \qquad \textit{Zustand}$$

die Lokationen auf Zahlen abbilden und die wir als *Zustände* bezeichnen. Außerdem wählen wir ein Objekt $\perp$, das kein Element von Σ ist (zum Beispiel die leere Menge) und definieren:

$$\Sigma_\perp \stackrel{\text{def}}{=} \Sigma \cup \{\perp\}$$

Um ein Kommando c auszuführen, benötigen wir einen *Anfangszustand*. Wenn die Ausführung des Kommandos terminiert, bekommen wir einen *Endzustand*. Der Effekt einer terminierenden Ausführung besteht also darin, einen Anfangszustand in einen Endzustand zu überführen. Wir können also jedem Kommando c eine Funktion

$$\phi \in \Sigma \to \Sigma_\perp$$

zuordnen. Dabei bedeutet $\phi\sigma = \perp$, dass die Ausführung von c mit dem Anfangszustand σ divergiert, und $\phi\sigma = \sigma'$, dass die Ausführung von c mit dem Anfangszustand σ mit dem Endzustand σ' terminiert.

Wir wählen die gerade beschriebene Funktion als Denotation eines Kommandos. Die Denotation eines Kommandos beschreibt sein Ein-Ausgabe-Verhalten. Die Denotation enthält weniger Information als das Kommando, da sie keinen Algorithmus für die Berechnung des Endzustands vorgibt.

Abbildung 6.2 definiert die denotationale Semantik von IMP. Für jede der drei Phrasenklassen wird eine eigene Denotationsfunktion mithilfe von struktureller Rekursion definiert. Bis auf die Gleichung für Schleifen sollten Ihnen alle Gleichungen unmittelbar klar sein.

Wir erklären jetzt die Gleichung für Schleifen. Zunächst stellen wir fest, dass für alle b und c die Gleichung

$$\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c) = \mathcal{C}(\mathsf{if}\ b\ \mathsf{then}\ c;\ \mathsf{while}\ b\ \mathsf{do}\ c\ \mathsf{else\ skip})$$

gelten muss. Diese Gleichung legt nahe, dass die Denotation von Schleifen rekursiv definiert werden muss. So wie sie ist, kann die Gleichung nicht für die Definition von $\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c)$ verwendet werden, da die durch sie beschriebene Rekursion nicht strukturell ist. Das Prinzip der strukturellen Rekursion verlangt, dass wir $\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c)$ nur mithilfe von $\beta = \mathcal{B}(b)$ und $\phi = \mathcal{C}(c)$ beschreiben. Wir versuchen jetzt, $\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c)$ durch die folgende, rekursiv definierte

$$
\begin{aligned}
&\mathcal{A} \in Aexp \to \Sigma \to \mathbb{Z}\\
&\qquad \mathcal{A}(n)\sigma = n\\
&\qquad \mathcal{A}(X)\sigma = \sigma(X)\\
&\qquad \mathcal{A}(a_1 + a_2)\sigma = \mathcal{A}(a_1)\sigma + \mathcal{A}(a_2)\sigma
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{B} \in Bexp \to \Sigma \to \mathbb{B}\\
&\qquad \mathcal{B}(a_1 \le a_2)\sigma = \text{if } \mathcal{A}(a_1)\sigma \le \mathcal{A}(a_2)\sigma \text{ then 1 else 0}
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{C} \in Com \to \Sigma \to \Sigma_\perp\\
&\qquad \mathcal{C}(X := a)\sigma = \sigma[\mathcal{A}(a)\sigma/X]\\
&\qquad \mathcal{C}(c_1; c_2)\sigma = \text{if } \mathcal{C}(c_1)\sigma = \perp \text{ then } \perp \text{ else } \mathcal{C}(c_2)(\mathcal{C}(c_1)\sigma)\\
&\qquad \mathcal{C}(\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2)\sigma = \text{if } \mathcal{B}(b)\sigma = 0 \text{ then } \mathcal{C}(c_2)\sigma \text{ else } \mathcal{C}(c_1)\sigma\\
&\qquad \mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c)\sigma = \mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ \sigma
\end{aligned}
$$

$$
\begin{aligned}
&\Gamma \in (\Sigma \to \mathbb{B}) \times (\Sigma \to \Sigma_\perp) \to [(\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)]\\
&\qquad \Gamma(\beta, \phi)\ \psi\ \sigma \;=\; \text{if } \beta\sigma = 0 \text{ then } \sigma \text{ else if } \phi\sigma = \perp \text{ then } \perp \text{ else } \psi(\phi\sigma)
\end{aligned}
$$

Abbildung 6.2: Denotationale Semantik von IMP

Funktion zu beschreiben:

$$
\begin{aligned}
&\psi \in \Sigma \to \Sigma_\perp\\
&\psi\sigma \;=\; \mathsf{if}\ \beta\sigma = 0\ \mathsf{then}\ \sigma\ \mathsf{else\ if}\ \phi\sigma = \perp\ \mathsf{then}\ \perp\ \mathsf{else}\ \psi(\phi\sigma)
\end{aligned}
$$

Die für die Definition von ψ verwendete Rekursion ist mathematisch unzulässig, da sie im Allgemeinen nicht terminiert. Dieser Problem können wir umgehen, indem wir auf das der rekursiven Definition entsprechende Funktional

$$
\begin{aligned}
&\Psi \in (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)\\
&\Psi\ \psi\ \sigma \;=\; \mathsf{if}\ \beta\sigma = 0\ \mathsf{then}\ \sigma\ \mathsf{else\ if}\ \phi\sigma = \perp\ \mathsf{then}\ \perp\ \mathsf{else}\ \psi(\phi\sigma)
\end{aligned}
$$

ausweichen und ψ mit dem Fixpunktoperator der VPO $\Sigma \to \Sigma_\perp$ definieren: $\psi \stackrel{\text{def}}{=} \mathit{fix}\ \Psi$. Die Monotonie von Ψ ist offensichtlich. Die Stetigkeit folgt mit Proposition 5.5.4, da $\Psi\psi\sigma$ die Funktion ψ nur für das Argument $\phi\sigma$ benötigt. Die Definition von $\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c)$ in Abbildung 6.2 realisiert die beschriebene Idee mit einer Hilfsfunktion Γ.

Realisierung in Standard ML

Die gerade für IMP definierte Syntax und Semantik lässt sich unmittelbar in Standard ML realisieren. Der für Schleifen benötigte Fixpunktoperator kann durch die Prozedur `fix` aus Abschnitt 5.2 realisiert werden. Sie sollten diese Übungsaufgabe unbedingt vollständig ausführen. Sie lernen dabei, dass es sich bei unseren Definitionen für die Syntax und Semantik von IMP um funktionale Programme in mathematischer Notation handelt. Die Prozedur für die Denotationsfunktion $\mathcal{C}$ liefert einen Interpreter für Kommandos. Damit können wir die Korrektheit unserer Semantik durch Experimente überprüfen.

Die Realisierung der denotationalen Semantik in Standard ML unterscheidet sich gegenüber der mathematischen Formulierung dadurch, dass die Prozedur, die die Denotation eines Kommandos berechnet, divergiert, falls das Kommando divergiert. Es stellt sich die Frage, ob man die denotationale Semantik so implementieren kann, dass die für ein Kommando gelieferte Prozedur immer terminiert (indem sie im Falle von Divergenz einen Wert liefert, der $\perp$ entspricht). Diese Frage werden wir später mit nein beantworten (da das Halteproblem von IMP unentscheidbar ist).

Zwei Propositionen

Die folgende Propositionen formulieren zwei wichtige Eigenschaften der Denotation von Schleifen.

Proposition 6.1.1 *Seien $b \in Bexp$, $c \in Com$ und $\sigma, \sigma' \in \Sigma$. Dann:*

$$\mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma = \sigma' \iff \exists n \in \mathbb{N}\colon\ (\Gamma(\mathcal{B}(b), \mathcal{C}(c)))^n\,(\lambda\sigma\,.\,\perp)\,\sigma = \sigma'$$

Beweis Folgt sofort aus der Definition von $\mathcal{C}$ und den Eigenschaften der VPO $\Sigma \to \Sigma_\perp$. □

Proposition 6.1.2 *Sei $b \in Bexp$ und $c \in Com$. Dann:*

$$\mathcal{C}(\textsf{while } b \textsf{ do } c) = \mathcal{C}(\textsf{if } b \textsf{ then } c; \textsf{while } b \textsf{ do } c \textsf{ else skip})$$

Beweis Sei $\sigma \in \Sigma$. Dann:

$$\begin{aligned}
\mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma &= \mathit{fix}\,(\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ \sigma \\
&= (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ (\mathit{fix}\,(\Gamma(\mathcal{B}(b),\ \mathcal{C}(c))))\ \sigma \\
&= (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ (\mathcal{C}(\textsf{while } b \textsf{ do } c))\ \sigma \\
&= \text{if } \mathcal{B}(b)\sigma = 0 \text{ then } \sigma \\
&\quad\ \text{else if } \mathcal{C}(c)\sigma = \perp \text{ then } \perp
\end{aligned}$$

$$\begin{aligned}
&\quad \text{else } (\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c))(\mathcal{C}(c)\sigma) \\
&= \text{if } \mathcal{B}(b)\sigma = 0 \text{ then } \mathcal{C}(\mathsf{skip})\sigma \\
&\quad \text{else } \mathcal{C}(c;\ \mathsf{while}\ b\ \mathsf{do}\ c)\sigma \\
&= \mathcal{C}(\mathsf{if}\ b\ \mathsf{then}\ (c;\ \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else\ skip})\sigma
\end{aligned}$$

Die zweite Gleichung folgt aus der Fixpunkteigenschaft von *fix*, alle anderen Gleichungen folgen direkt aus den Definitionen. □

Äquivalenz und Kongruenzeigenschaften

Wir haben die denotationale Semantik von IMP nach einem Muster definiert, dass wir bereits mehrfach angewendet haben. Dieses Muster gibt uns Definitionen für Äquivalenz und Substitution vor und garantiert die üblichen Kongruenzeigenschaften.

Da wir diesmal drei syntaktische Klassen mit getrennten Denotationsfunktionen haben, bekommen wir auch drei Äquivalenzrelationen:

$$\begin{aligned}
a_1 \models\!\!\!\dashv a_2 &\iff \mathcal{A}(a_1) = \mathcal{A}(a_2) \\
b_1 \models\!\!\!\dashv b_2 &\iff \mathcal{B}(b_1) = \mathcal{B}(b_2) \\
c_1 \models\!\!\!\dashv c_2 &\iff \mathcal{C}(c_1) = \mathcal{C}(c_2)
\end{aligned}$$

Die Kongruenzeigenschaften für arithmetische Ausdrücke lassen sich wie gehabt mithilfe der Substitutionsoperation formulieren. Da die Syntax für IMP jedoch keine Variablen wir Boolesche Ausdrücke und Kommandos vorsieht, lassen sich die Kongruenzeigenschaften dieser syntaktischen Objekte nicht durch Rückgriff auf Variablensubstitution formulieren.

Wir halten fest, dass die folgenden Substitutionseigenschaften gelten:

$$\begin{aligned}
\mathcal{A}(a[a'/X])\sigma &= \mathcal{A}(a)(\sigma[\mathcal{A}(a')\sigma/X]) \\
\mathcal{B}(b[a'/X])\sigma &= \mathcal{B}(b)(\sigma[\mathcal{A}(a')\sigma/X])
\end{aligned}$$

6.2 Operationale Semantik

Wir definieren jetzt eine zweite Semantik für IMP, die als *operationale Sematik* bezeichnet wird. Dazu definieren wir mithilfe von Inferenzregeln eine Relation

$$OCom \subseteq \Sigma \times Com \times \Sigma$$

so dass $\langle \sigma, c, \sigma' \rangle \in OCom$ genau dann gilt, wenn die Ausführung des Kommandos c für den Anfangszustand σ mit dem Endzustand σ' terminiert. Statt

$$\frac{\mathcal{A}(a)\sigma = n \qquad \sigma' = \sigma[n/X]}{\sigma \vdash X := a \Rightarrow \sigma'}$$

$$\frac{\sigma \vdash c_1 \Rightarrow \sigma' \qquad \sigma' \vdash c_2 \Rightarrow \sigma''}{\sigma \vdash c_1; c_2 \Rightarrow \sigma''}$$

$$\frac{\mathcal{B}(b)\sigma = 0 \qquad \sigma \vdash c_2 \Rightarrow \sigma'}{\sigma \vdash \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \Rightarrow \sigma'}$$

$$\frac{\mathcal{B}(b)\sigma = 1 \qquad \sigma \vdash c_1 \Rightarrow \sigma'}{\sigma \vdash \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \Rightarrow \sigma'}$$

$$\frac{\mathcal{B}(b)\sigma = 0}{\sigma \vdash \textsf{while } b \textsf{ do } c \Rightarrow \sigma}$$

$$\frac{\mathcal{B}(b)\sigma = 1 \qquad \sigma \vdash c \Rightarrow \sigma' \qquad \sigma' \vdash \textsf{while } b \textsf{ do } c \Rightarrow \sigma''}{\sigma \vdash \textsf{while } b \textsf{ do } c \Rightarrow \sigma''}$$

Abbildung 6.3: Operationale Semantik von Kommandos

$\langle \sigma, c, \sigma' \rangle \in OCom$ werden wir

$$\sigma \vdash c \Rightarrow \sigma'$$

schreiben.

Die operationale Sematik von IMP wird durch die Inferenzregeln in Abbildung 6.3 definiert. Dabei verwenden wir die bereits definierten Denotationsfunktionen für arithmetische und Boolesche Ausdrücke. Selbstverständlich kann man die Semantik dieser Ausdrücke ebenfalls operational definieren (siehe [Winskel]). Man bekäme dann zwei Relationen

$$\begin{aligned} OAexp &\subseteq \Sigma \times Aexp \times \mathbb{Z} \\ OBexp &\subseteq \Sigma \times Bexp \times \mathbb{B} \end{aligned}$$

Im nächsten Abschnitt werden wir beweisen, dass die denotationale und die operationale Semantik von Kommandos übereinstimmen. Übereinstimmung bedeutet dabei, dass für alle $c \in Com$ und alle $\sigma, \sigma' \in \Sigma$ gilt:

$$\sigma \vdash c \Rightarrow \sigma' \iff \mathcal{C}(c)\sigma = \sigma'$$

Denotationale versus operationale Semantik

Wir gehen jetzt auf die Unterschiede zwischen den beiden Sematiken ein:

1. Die denotationale Semantik arbeitet mit funktionaler Notation, die operationale Semantik mit relationaler Notation.

2. Die denotationale Semantik verwendet zwei getrennte Rekursionen: Strukturelle Rekursion für die Interpretation der Syntax, und Fixpunktrekursion für die Berechnung von Schleifen. Die operationale Semantik behandelt beide Aspekte mit nur einer Rekursion.

3. Die denotationale Semantik ist der bessere Ausgangspunkt für den Beweis von Spracheigenschaften (wegen der unter (1) und (2) angeführten Punkte).

4. Da die Definition der operationalen Semantik ohne die Verwendung eines Fixpunktoperators auskommt, ist sie für den mathematisch unerfahrenen Leser leichter zu verstehen als die denotationale Semantik.

Die operationale Methode ist die Standardmethode für die Beschreibung der Semantik von Programmiersprachen. Die Definition der Semantik von Standard ML ist ein gutes Beispiel für die Anwendung der operationalen Methode auf eine komplexe Sprache.[1] Die operationale Methode hat den Vorteil, dass sie im Gegensatz zur denotationalen Methoden auf alle Sprachtypen anwendbar ist. Die denotationale Methode versagt insbesondere bei nebenläufigen Sprachen.

6.3 Übereinstimmung der Semantiken

Wir führen den Übereinstimmungsbeweis in zwei Teilen.

Proposition 6.3.1 $\forall \sigma \in \Sigma\ \forall c \in Com\ \forall \sigma' \in \Sigma:\ \sigma \vdash c \Rightarrow \sigma' \Rightarrow \mathcal{C}(c)\sigma = \sigma'$.

Beweis Durch Regelinduktion über die Definition der operationalen Semantik. Von besonderem Interesse sind die Regeln für Schleifen, die Beweisteile für die anderen Regeln sind einfach. Wir beschränken uns auf die Beweisteile für die Regeln für Sequentialisierungen und Schleifen.

1. *Regel für Sequentialisierung.* Wir nehmen an:

$$\mathcal{C}(c_1)\sigma = \sigma' \ \wedge\ \mathcal{C}(c_2)\sigma' = \sigma''$$

[1] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen, *The Definition of Standard ML (Revised)*, MIT Press, Cambridge, Massachusetts, 1997.

Wir müssen zeigen:

$$\mathcal{C}(c_1; c_2)\sigma = \sigma''$$

Das folgt mit der Definition von $\mathcal{C}$ und den Annahmen:

$$\begin{aligned}\mathcal{C}(c_1; c_2)\sigma &= \text{if } \mathcal{C}(c_1)\sigma = \bot \text{ then } \bot \text{ else } \mathcal{C}(c_2)(\mathcal{C}(c_1)\sigma) \\ &= \mathcal{C}(c_2)(\mathcal{C}(c_1)\sigma) \\ &= \sigma''\end{aligned}$$

2. *Erste Regel für Schleifen.* Wir nehmen an:

$$\mathcal{B}(b)\sigma = 0$$

Wir müssen zeigen:

$$\mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma = \sigma$$

Das tun wir unter Benutzung der Fixpunkteigenschaft wie folgt:

$$\begin{aligned}\mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma &= \mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ \sigma \\ &= (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ (\mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c))))\ \sigma \\ &= \text{if } \mathcal{B}(b)\sigma = 0 \text{ then } \sigma \text{ else } \cdots \\ &= \sigma\end{aligned}$$

3. *Zweite Regel für Schleifen.* Wir nehmen an:

$$\mathcal{B}(b)\sigma = 1 \ \wedge\ \mathcal{C}(c)\sigma = \sigma' \ \wedge\ \mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma' = \sigma''$$

Wir müssen zeigen:

$$\mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma = \sigma''$$

Das tun wir unter Benutzung der Fixpunkteigenschaft wie folgt:

$$\begin{aligned}\mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma &= \mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ \sigma \\ &= (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ (\mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c))))\ \sigma \\ &= \text{if } \mathcal{B}(b)\sigma = 0 \text{ then } \sigma \\ &\quad\ \text{else if } \mathcal{C}(c)\sigma = \bot \text{ then } \bot \\ &\qquad\quad \text{else } (\mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c))))\ (\mathcal{C}(c)\sigma) \\ &= \mathit{fix}\ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ \sigma' \\ &= \mathcal{C}(\textsf{while } b \textsf{ do } c)\sigma' \\ &= \sigma''\end{aligned}$$

□

Proposition 6.3.2 $\forall \sigma \in \Sigma\ \forall c \in Com\ \forall \sigma' \in \Sigma\colon\ \mathcal{C}(c)\sigma = \sigma' \Rightarrow \sigma \vdash c \Rightarrow \sigma'$.

Beweis Durch strukturelle Induktion über *Com*. Wer benötigen also einen Beweisteil für jede Gleichung der denotationalen Semantik. Interessant ist der Beweisteil für Schleifen, die restlichen Beweisteile sind einfach. Wir zeigen die Beweisteile für Sequentialsierung und Schleifen.

1. Sei $\mathcal{C}(c_1; c_2)\sigma = \sigma'$. Es genügt zu zeigen, dass ein $\sigma'' \in \Sigma$ existiert mit

   $$\sigma \vdash c_1 \Rightarrow \sigma'' \ \wedge\ \sigma'' \vdash c_2 \Rightarrow \sigma'$$

   Mit der Annahme und der Definition von $\mathcal{C}$ folgt:

   $$\begin{aligned} \sigma' &= \mathcal{C}(c_1; c_2)\sigma \\ &= \text{if } \mathcal{C}(c_1)\sigma = \bot \text{ then } \bot \text{ else } \mathcal{C}(c_2)(\mathcal{C}(c_1)\sigma) \\ &= \mathcal{C}(c_2)(\mathcal{C}(c_1)\sigma) \end{aligned}$$

   Also existiert ein σ'' mit

   $$\mathcal{C}(c_1)\sigma = \sigma'' \ \wedge\ \mathcal{C}(c_2)\sigma'' = \sigma'$$

   Jetzt liefert die Induktionsannahme

   $$\sigma \vdash c_1 \Rightarrow \sigma'' \ \wedge\ \sigma'' \vdash c_2 \Rightarrow \sigma'$$

2. Sei $\mathcal{C}(\mathsf{while}\ b\ \mathsf{do}\ c)\sigma = \sigma'$. Wir müssen zeigen, dass

   $$\sigma \vdash \mathsf{while}\ b\ \mathsf{do}\ c \Rightarrow \sigma'$$

   gilt. Wegen Proposition 6.1.1 genügt es, die folgende Aussage zu zeigen:

   $$\begin{aligned} &\forall n \in \mathbb{N}\ \forall \sigma \in \Sigma\ \forall \sigma' \in \Sigma\colon \\ &\quad \sigma' = (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))^n(\lambda\, \sigma\, .\, \bot))\ \sigma \ \Rightarrow\ \sigma \vdash \mathsf{while}\ b\ \mathsf{do}\ c \Rightarrow \sigma' \end{aligned}$$

   Diese Behauptung zeigen wir durch Induktion über $n \in \mathbb{N}$.

   Sei $n = 0$. Dann ist die Implikation trivialerweise erfüllt, da $\sigma' \neq \bot$.

   Sei $n > 0$. Weiter sei $\sigma' = (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))^n(\lambda\, \sigma\, .\, \bot)\ \sigma$. Dann

   $$\sigma' \ = \ (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))\ ((\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))^{n-1}(\lambda\, \sigma\, .\, \bot))\ \sigma$$

   Wir unterscheiden jetzt zwei Fälle:

   a) *Sei* $\mathcal{B}(b)\sigma = 0$. Dann $\sigma = \sigma'$ nach Definition von Γ. Also folgt $\sigma \vdash \mathsf{while}\ b\ \mathsf{do}\ c \Rightarrow \sigma'$ mit der ersten Regel für Schleifen.

b) *Sei* $\mathcal{B}(b)\sigma = 1$. Dann folgt aus der Definition von Γ, dass ein σ'' existiert mit

$$\sigma'' = \mathcal{C}(c)\sigma$$

und

$$\sigma' = (\Gamma(\mathcal{B}(b),\ \mathcal{C}(c)))^{n-1}(\lambda\,\sigma\,.\,\bot)\ \sigma''$$

Mit der äußeren Induktionsannahme folgt

$$\sigma \vdash c \Rightarrow \sigma''$$

Mit der inneren Induktionsannahme folgt

$$\sigma'' \vdash \textsf{while}\ b\ \textsf{do}\ c \Rightarrow \sigma'$$

Also haben wir $\sigma \vdash \textsf{while}\ b\ \textsf{do}\ c \Rightarrow \sigma'$ mit der zweiten Regel für Schleifen.

□

Satz 6.3.3 (Übereinstimmung) *Für alle $c \in Com$ und alle $\sigma, \sigma' \in \Sigma$ gilt:*

$$\sigma \vdash c \Rightarrow \sigma' \iff \mathcal{C}(c)\sigma = \sigma'$$

Beweis Proposition 6.3.1 und Proposition 6.3.2. □

Aus der Übereinstimmung der Semantiken bekommen wir sofort eine wichtige Eigenschaft der operationalen Semantik.

Korollar 6.3.4 *Für jedes Kommando $c \in Com$ und jeden Zustand $\sigma \in \Sigma$ gibt es höchstes einen Zustand $\sigma' \in \Sigma$ mit $\sigma \vdash c \Rightarrow \sigma'$.*