Kapitel 6

Programme

In diesem Kapitel beschäftigen wir uns mit der Semantik von Programmen. Dazu betrachten wir eine kleine, idealisierte Programmiersprache IMP, die als Teilsprache von C aufgefasst werden kann. IMP realisiert grundlegende imperative Konzepte wie zuweisbare Variablen und Schleifen.

Wir definieren zwei verschiedene Semantiken für IMP und zeigen ihre Übereinstimmung. Die mathematisch elegantere der zwei Semantiken ist die so genannte *denotationale Semantik*, die Programmen mittels struktureller Rekursion eine Funktion zuordnet. Dabei wir die denotationale Semantik von Schleifen mithilfe einer Fixpunktkonstruktion realisiert. Die zweite Semantik für IMP wird als *operationale Semantik* bezeichnet. Sie ist mithilfe von Inferenzregeln definiert und benötigt keine Fixpunktkonstruktion.

Schließlich führen wir die theoretisch motivierte Sprache der regulären Programme ein, mit der eine Obermenge der durch IMP beschreibaren Relationen beschrieben werden kann. Da reguläre Programme statt auf Ausführbarkeit primär auf die Beschreibung von Relationen ausgerichtet sind, lässt sich ihre Semantik sehr einfach definieren. Indem wir IMP als eine Teilsprache der Sprache der regulären Programme identifizieren, erhalten wir eine zweite denotationale Semantik für IMP, die einfacher und eleganter als die erste ist.

Literaturverweise

- David Harel, Dexter Kozen, Jerzy Tiuryn. *Dynamic Logic*. The MIT Press, 2000.
- Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. The MIT Press, 1993.

Das Buch von Winskel richtet sich an Anfänger und behandelt die denotationale und operationale Semantik von IMP ausführlich. Leider finden reguläre Programme bei Winskel keine Erwähnung. Reguläre Programme und die damit verbundene Logik sind der Hauptgegenstand des Buchs von Harel, Kozen und Tiuryn.

X	$\in$	Loc	**Lokation**
n	$\in$	$Con = \mathbb{Z}$	**Konstante**
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**
		$\mid c_1; c_2$	**sequentielle Komposition**
		$\mid$ 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

Abbildung 6.1 definiert die Syntax einer idealisierten Programmiersprache IMP, die man sich als Teilsprache von C vorstellen kann. Die Syntax von IMP ist relativ zu einer nichtleeren Menge *Loc* von so genannten **Lokationen** definiert, die den Variablen von *C* entsprechen. IMP ist nicht als praktisch einsetzbare Programmiersprache entworfen, sondern als Sprache für theoretische Studien. Ein wichtiger Unterschied zwischen IMP und praktischen Programmiersprachen besteht darin, dass IMP mit beliebig großen Zahlen rechnen kann.

Die Lokationen von IMP werden wir auch als **Variablen** bezeichnen, und die Kommandos als **Anweisungen**.

Das intuitive Ausführungsmodell für IMP verfügt über einen *Speicher*, in dem für jede Lokation eine Zahl abgelegt ist. Die Ausführung eines Kommandos erfolgt schrittweise und beginnt mit einem vorgegebenen Speicherzustand, der als **Anfangszustand** bezeichnet wird. Während der Ausführung kann ein Kommando die unter einer Lokation abgelegten Werte zugreifen und verändern (mithilfe des Zuweisungskommandos). Die Ausführung eines Kommandos kann unendlich lange fortschreiten (*Divergenz*), oder nach endlich vielen Schritten *terminieren*. Wenn die Ausführung eines Kommandos terminiert, hinterlässt sie im Speicher den so genannten **Endzustand**.

Hier ist ein Kommando, dessen Ausführung für jeden Anfangszustand divergiert:

```
while true do X:=X+1
```

Die Ausführung des Kommandos legt unter der Lokation X schrittweise immer größere Zahlen ab.

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

Bei X_0 handelt es sich um eine einmal festgelegte Lokation.

Abbildung 6.2: Abkürzungen für IMP

Unser zweites Beispiel ist ein terminierendes Kommando, das das Produkt zweier an den Lokationen X und Y abgelegter Zahlen berechnet und es an der Lokation Z vermerkt:

```
Z:=0; I:=1;
while I≤Y do (Z:=Z+X; I:=I+1)
```

Um IMP-Kommandos lesbarer schreiben können, vereinbaren wir die in Abbildung 6.2 gezeigten notationalen Abkürzungen.

Wir wollen jetzt bestimmte Aspekte des informellen Ausführungsmodells für IMP formalisieren. Die Zustände des Speichers formalisieren wir wie folgt:

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

Für die Darstellung von Divergenz wählen wir ein Objekt $\perp$, das kein Element von Σ ist (zum Beispiel die leere Menge) und definieren:

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

Gemäß dem informellen Ausführungsmodell beschreibt jedes Kommando eine Funktion

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

wie folgt:

1. Wenn die Ausführung des Kommandos für einen Anfangszustand σ divergiert, gilt $\phi\sigma = \perp$.
2. Wenn die Ausführung des Kommandos für einen Anfangszustand σ mit dem Endzustand σ' terminiert, gilt $\phi\sigma = \sigma'$.

Die Funktion ϕ bezeichnen wir als **Denotation** des Kommandos. Die Denotation eines Kommandos beschreibt sein Ein-Ausgabe-Verhalten. Die Denotation enthält viel weniger Information als das Kommando, da sie keinen Algorithmus für die Berechnung des Endzustands vorgibt.

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

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

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

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

Abbildung 6.3: Denotationale Semantik von IMP

Der abschließende Schritt unserer Formalisierung besteht darin, die Funktion

$$C \in Com \to \Sigma \to \Sigma_\perp$$

die jedem Kommando seine Denotation zuordnet, unabhängig vom informellen Ausführungsmodell zu definieren. Um diese Funktion definieren zu können, benötigen wir zunächst zwei weitere Funktionen

$$\mathcal{A} \in Aexp \to \Sigma \to \mathbb{Z}$$
$$\mathcal{B} \in Bexp \to \Sigma \to \mathbb{B}$$

die zu einem Ausdruck und einem Zustand den Wert liefern, den der Ausdruck für diesen Zustand liefern soll. Die formale Definition der Funktionen $\mathcal{A}$, $\mathcal{B}$ und C ist in Abbildung 6.3 angegeben. Man bezeichnet $\mathcal{A}$, $\mathcal{B}$ und C als **Denotationsfunktionen** und ihre Definition als eine **Denotationale Semantik**.

Die Funktionen $\mathcal{A}$, $\mathcal{B}$ und C werden jeweils durch struktureller Rekursion über die syntaktischen Objekte definiert. Bis auf die Gleichung für Schleifen sollte Ihnen die Definition der Denotationsfunktionen unmittelbar klar sein.

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

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

gilt (gemäß unseres intuitiven Verständnisses). Die Gleichung kann als eine rekursive Charakterisierung der Denotation von Schleifen aufgefasst werden. Allerdings kann die Gleichung nicht direkt für die Definition von $C(\text{while } b \text{ do } c)$ verwendet werden, da die durch sie beschriebene Rekursion nicht strukturell ist. Das Prinzip der strukturellen Rekursion verlangt, dass wir $C(\text{while } b \text{ do } c)$ nur mithilfe von $\beta = \mathcal{B}(b)$ und $\phi = C(c)$ beschreiben. Die obigen Gleichung sagt uns, dass es sich bei $\psi = C(\text{while } b \text{ do } c)$ um eine Funktion $\Sigma \to \Sigma_\perp$ handelt, die die folgende Gleichung erfüllt:

$$\psi\sigma = \text{if } \beta\sigma = 0 \text{ then } \sigma \text{ else if } \phi\sigma = \perp \text{ then } \perp \text{ else } \psi(\phi\sigma)$$

Unser VPO-Modell (Abschnitt 5.5) für die rekursive Definition von Funktionen $X \to Y_\perp$ versetzt uns in die Lage, diese Gleichung in der Tat als die Definition einer Funktion $\Sigma \to \Sigma_\perp$ aufzufassen. Dazu verwenden wir das Funktional

$$\Psi \in (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp)$$

$$\Psi\,\psi\,\sigma = \text{if } \beta\sigma = 0 \text{ then } \sigma \text{ else if } \phi\sigma = \perp \text{ then } \perp \text{ else } \psi(\phi\sigma)$$

und definieren ψ mit dem Fixpunktoperator der VPO $\Sigma \to \Sigma_\perp$:

$$\psi = \mathit{fix}\,\Psi$$

Die Monotonie des Funktional 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 $C(\text{while } b \text{ do } c)$ in Abbildung 6.3 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 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 denotationale Semantik von IMP formalisiert wesentliche Aspekte von IMP unabhängig vom informellen Ausführungsmodell. Damit sind wir in der Lage, viele Eigenschaften von IMP im Rahmen eines mathematischen Modells zu formulieren und zu beweisen. Die folgende Propositionen geben dafür Beispiele.

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

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

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

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

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

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

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

□

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

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

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

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

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

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

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

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

Abbildung 6.4: Operationale Semantik von Kommandos

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

$$\sigma \vdash c \Longrightarrow \sigma'$$

schreiben.

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

$$\sigma \vdash c \Longrightarrow \sigma' \iff C(c)\sigma = \sigma'$$

Die operationale Sematik von IMP wird durch die Inferenzregeln in Abbildung 6.4 definiert. Dabei verwenden wir die bereits definierten Denotationsfunktionen für arithmetische und Boolesche Ausdrücke. Wenn man will, kann man die Semantik dieser Ausdrücke ebenfalls operational definieren. Man bekäme dann zwei Relationen

$$\mathit{OAexp} \subseteq \Sigma \times \mathit{Aexp} \times \mathbb{Z}$$
$$\mathit{OBexp} \subseteq \Sigma \times \mathit{Bexp} \times \mathbb{B}$$

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 haben jetzt zwei formale Charakterisierungen der Semantik von IMP. Natürlich erwarten wir, dass die beiden Charakterisierung übereinstimmen. Wir führen den Übereinstimmungsbeweis in zwei Teilen.

Proposition 6.3.1 $\forall \sigma \in \Sigma\ \forall c \in Com\ \forall \sigma' \in \Sigma\colon\ \sigma \vdash c \Longrightarrow \sigma' \Longrightarrow C(c)\sigma = \sigma'.$

Beweis Durch Regelinduktion über die Definition der operationalen Semantik. Dabei legen wir die Aussagemenge

$$P \;=\; \{\, (\sigma, c, \sigma') \in \Sigma \times Com \times \Sigma \mid C(c)\sigma = \sigma' \,\}$$

zugrunde. 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 sequentielle Komposition und Schleifen.

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

1. **Regel für sequentielle Komposition.** Wir nehmen an:

   $$C(c_1)\sigma = \sigma' \ \wedge \ C(c_2)\sigma' = \sigma''$$

   Wir müssen zeigen:

   $$C(c_1; c_2)\sigma = \sigma''$$

   Das folgt mit der Definition von C und den Annahmen:

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

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

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

   Wir müssen zeigen:

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

   Das tun wir unter Benutzung der Fixpunkteigenschaft wie folgt:

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

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

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

   Wir müssen zeigen:

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

   Das tun wir unter Benutzung der Fixpunkteigenschaft wie folgt:

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

□

Proposition 6.3.2 $\forall \sigma \in \Sigma\ \forall c \in Com\ \forall \sigma' \in \Sigma\colon\ C(c)\sigma = \sigma' \Longrightarrow \sigma \vdash c \Longrightarrow \sigma'$.

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 sequentielle Komposition und Schleifen.

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

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

   Mit der Annahme und der Definition von C folgt:

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

   Also existiert ein σ'' mit

   $$C(c_1)\sigma = \sigma'' \ \wedge\ C(c_2)\sigma'' = \sigma'$$

   Jetzt liefert die Induktionsannahme

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

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

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

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

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

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

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

   Wir unterscheiden jetzt zwei Fälle:

   a) **Sei** $\mathcal{B}(b)\sigma = 0$**.** Dann $\sigma = \sigma'$ nach Definition von Γ. Also folgt $\sigma \vdash \mathsf{while}\ b\ \mathsf{do}\ c \Longrightarrow \sigma'$ 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

$$\sigma'' = C(c)\sigma$$

und

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

Mit der äußeren Induktionsannahme folgt

$$\sigma \vdash c \Longrightarrow \sigma''$$

Mit der inneren Induktionsannahme folgt

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

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

□

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

$$\sigma \vdash c \Longrightarrow \sigma' \iff C(c)\sigma = \sigma'$$

Beweis Proposition 6.3.1 und Proposition 6.3.2. □

Aus der Übereinstimmung der Semantiken folgt 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 $\sigma \vdash c \Longrightarrow \sigma'$.*

6.4 Reguläre Programme

Semantisch gesehen sind IMP-Kommandos Beschreibungen von Funktionen $\Sigma \to \Sigma_\bot$. Wir führen jetzt eine zweite Klasse von Beschreibungen für solche Funktionen ein. Die neuen Beschreibungen werden als reguläre Programme bezeichnet. Reguläre Programme sind ein theoretisches Hilfsmittel, das uns bei der semantischen Analyse von IMP hilft.

Es ist hilfreich, eine Parallele zur Aussagelogik zu ziehen. Dort haben wir Boolesche Formeln als primäre Beschreibungen für Boolesche Funktionen. Zusätzlich haben wir mit Entscheidungsbäumen und Klauselformen zwei weitere Beschreibungssysteme für Boolesche Formeln zur Verfügung.

Funktionen des Typs $\Sigma \to \Sigma_\perp$ können wir als partielle Funktionen des Typs $\Sigma \rightharpoonup \Sigma$ auffassen, und diese wiederum sind funktionale binäre Relationen auf Σ:

$$\Sigma \to \Sigma_\perp \;\cong\; \Sigma \rightharpoonup \Sigma \;\subseteq\; \mathcal{P}(\Sigma \times \Sigma)$$

Reguläre Programme sind Beschreibungen für binäre Relationen auf Σ. Sie können alle funktionalen Relationen beschreiben, die durch IMP-Kommandos beschreibar sind. Zusätzlich können sie auch nichtfunktionale Relationen beschreiben.

Sei R eine binäre Relation auf einer Menge X. Wir definieren die folgenden Notationen:

$$\begin{aligned}
R^0 &\stackrel{\text{def}}{=} Id(X) = \{\, (x,x) \mid x \in X \,\} \\
R^n &\stackrel{\text{def}}{=} R \circ R^{n-1} \quad \text{falls } n \geq 1 \\
R^+ &\stackrel{\text{def}}{=} \{\, p \mid \exists n \in \mathbb{N}^+ \colon\; p \in R^n \,\} = \bigcup_{n \in \mathbb{N}^+} R^n && \textbf{transitiver Abschluss} \\
R^* &\stackrel{\text{def}}{=} Id(X) \cup R^+ = \bigcup_{n \in \mathbb{N}} R^n && \textbf{reflexiv transitiver Abschluss}
\end{aligned}$$

Proposition 6.4.1 *Seien R, R_1, R_2 binäre Relationen auf einer Menge X. Dann gelten die folgenden Gleichungen:*

$$\begin{aligned}
R \circ (R_1 \circ R_2) &= (R \circ R_1) \circ R_2 \\
R \circ (R_1 \cup R_2) &= (R \circ R_1) \cup (R \circ R_2) \\
(R_1 \cup R_2) \circ R &= (R_1 \circ R) \cup (R_2 \circ R) \\
R \circ R^* &= R^* \circ R = R^+
\end{aligned}$$

Abbildung 6.5 definiert die Syntax und Semantik von regulären Programmen. Ähnlich wie IMP-Kommandos werden reguläre Programme mit arithmetischen und Booleschen Ausdrücken gebildet. Informell kann die Denotation regulärer Programme wie folgt beschrieben werden:

- Zuweisung ist wie in IMP definiert.
- Ein Test b? beschreibt die Identität auf der Menge aller Zustände, die die Bedingung b erfüllen.
- Sequentielle Komposition beschreibt wie in IMP die Komposition von Relationen.
- Parallele Komposition beschreibt die Vereinigung von Relationen.
- Iteration beschreibt den reflexiv transitiven Abschluss von Relationen.

X	$\in$	Loc	**Lokation**
n	$\in$	$Con = \mathbb{Z}$	**Konstante**
a	$\in$	$Aexp = n \mid X \mid a_1 + a_2$	**arithmetischer Ausdruck**
b	$\in$	$Bexp = a_1 \leq a_2 \mid \neg b$	**Boolscher Ausdruck**
p	$\in$	$Pro =$	**reguläres Programm**
		$X := a$	**Zuweisung**
		$\mid b?$	**Test**
		$\mid p_1; p_2$	**sequentielle Komposition**
		$\mid p_1 + p_2$	**parallele Komposition**
		$\mid p^*$	**Iteration**

$\mathcal{A} \in Aexp \to \Sigma \to \mathbb{Z}$ wie bei IMP

$\mathcal{B} \in Bexp \to \Sigma \to \mathbb{B}$ wie bei IMP

$\mathcal{R} \in Pro \to \mathcal{P}(\Sigma \times \Sigma)$

$$\begin{aligned} \mathcal{R}(X := a) &= \lambda \sigma \in \Sigma . \sigma[X := \mathcal{A}(a)\sigma] \\ \mathcal{R}(b?) &= Id(\{ \sigma \in \Sigma \mid \mathcal{B}(b)\sigma = 1 \}) \\ \mathcal{R}(p_1; p_2) &= \mathcal{R}p_1 \circ \mathcal{R}p_2 \\ \mathcal{R}(p_1 + p_2) &= \mathcal{R}p_1 \cup \mathcal{R}p_2 \\ \mathcal{R}(p^*) &= (\mathcal{R}p)^* \end{aligned}$$

Abbildung 6.5: Syntax und Semantik regulärer Programme

Abbildung 6.6 zeigt einige Abkürzungen für reguläre Programme. Informell lässt sich die Semantik dieser Abkürzungen wie folgt beschreiben:

- skip beschreibt die Identität (wie in IMP).
- fail beschreibt die leere Relation.
- if beschreibt das Konditional aus IMP.
- while beschreibt die Schleife aus IMP.
- case beschreibt ein verallgemeinertes Konditional. Es gilt:

  $\mathsf{if}\ b\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 = \mathsf{case}\ b \to p_1 \mid \neg b \to p_2$

- iter beschreibt eine verallgemeinerte Schleife. Es gilt:

  $\mathsf{while}\ b\ \mathsf{do}\ p = \mathsf{iter}\ b \to p$

$$
\begin{aligned}
\neg(a_1 \le a_2) &\rightsquigarrow a_2 + 1 \le a_1 \\
\mathsf{skip} &\rightsquigarrow \mathsf{true}? \\
\mathsf{fail} &\rightsquigarrow \mathsf{false}? \\
\mathsf{if}\ b\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2 &\rightsquigarrow (b?; p_1) + ((\neg b)?; p_2) \\
\mathsf{while}\ b\ \mathsf{do}\ p &\rightsquigarrow (b?; p)^*; (\neg b)? \\
\mathsf{case}\ b_1 \to p_1 \mid \ldots \mid b_n \to p_n &\rightsquigarrow b_1?; p_1 + \cdots + b_n?; p_n \\
\mathsf{iter}\ b_1 \to p_1 \mid \ldots \mid b_n \to p_n &\rightsquigarrow (b_1?; p_1 + \cdots + b_n?; p_n)^*; (\neg b_1)?; \ldots; (\neg b_n)?
\end{aligned}
$$

Abbildung 6.6: Abkürzungen für reguläre Programme

Zusätzlich zu den hier definierten Abkürzungen werden wir auch die für IMP in Abbildung 6.2 definierten Abkürzungen für arithmetische Ausdrücke und Boolesche Ausdrücke verwenden.

Einfache Programme
Wir definieren eine Teilmenge der regulären Programme, deren Programme wir **einfach** nennen:

1. Jede Zuweisung ist einfach.
2. Die sequentielle Komposition einfacher Programme ist einfach.
3. Ein Konditional $\mathsf{if}\ b\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2$ ist einfach, wenn p_1 und p_2 einfach sind.
4. Eine Schleife $\mathsf{while}\ b\ \mathsf{do}\ p$ ist einfach, wenn p einfach ist.

Offensichtlich entsprechen die einfachen Programme genau den IMP-Kommandos. Jedes IMP-Kommando kann also in ein „bedeutungsgleiches“ einfaches reguläres Programm übersetzt werden.

Proposition 6.4.2 *Zu jedem IMP-Kommando c existiert ein einfaches Programm p, sodass gilt: $\mathcal{R}p = \{ (\sigma, \sigma') \in \Sigma^2 \mid C(c)\sigma = \sigma' \}$.*

Proposition 6.4.3 *Für jedes einfache Programm P gilt: $\mathcal{R}p \in \Sigma \rightharpoonup \Sigma$.*

Das reguläre Programm $(X := 1) + (X := 2)$ beschreibt eine nichtfunktionale Relation. Sie kann durch kein IMP-Kommando beschrieben werden, da IMP-Kommandos immer funktionale Relationen beschreiben.

Äquivalenz
Wir haben die Semantik regulärer Programme nach einem Muster definiert, dass wir bereits mehrfach angewendet haben. Dieses Muster gibt uns die Definitionen

$$p;(p_1;p_2) \models\!\mid (p;p_1);p_2 \tag{6.1}$$
$$p+(p_1+p_2) \models\!\mid (p+p_1)+p_2 \tag{6.2}$$
$$p_1+p_2 \models\!\mid p_2+p_1 \tag{6.3}$$
$$p+p \models\!\mid p \tag{6.4}$$
$$p;(p_1+p_2) \models\!\mid (p;p_1)+(p;p_2) \tag{6.5}$$
$$(p_1+p_2);p \models\!\mid (p_1;p)+(p_2;p) \tag{6.6}$$
$$\mathsf{skip};p \models\!\mid p \tag{6.7}$$
$$p;\mathsf{skip} \models\!\mid p \tag{6.8}$$
$$\mathsf{fail};p \models\!\mid \mathsf{fail} \tag{6.9}$$
$$p;\mathsf{fail} \models\!\mid \mathsf{fail} \tag{6.10}$$
$$p+\mathsf{fail} \models\!\mid p \tag{6.11}$$
$$p;p^* \models\!\mid p^*;p \tag{6.12}$$
$$p^* \models\!\mid \mathsf{skip}+p^* \tag{6.13}$$
$$p^* \models\!\mid \mathsf{skip}+p;p^* \tag{6.14}$$
$$(b?)^* \models\!\mid \mathsf{skip} \tag{6.15}$$
$$p^*;p^* \models\!\mid p^* \tag{6.16}$$

Die Äquivalenzen gelten für alle regulären Programme $p, p_1, p_2 \in \mathit{Pro}$ und alle Ausdrücke $b \in \mathit{Bexp}$.

Abbildung 6.7: Äquivalenzen für reguläre Programme

von Äquivalenz vor und garantiert die Gültigkeit der Ersetzungsregel. Da wir diesmal drei syntaktische Klassen mit getrennten Denotationsfunktionen haben, bekommen wir auch drei Äquivalenzrelationen:

$$\begin{aligned} a_1 \models\!\mid a_2 &\iff \mathcal{A}(a_1) = \mathcal{A}(a_2) \\ b_1 \models\!\mid b_2 &\iff \mathcal{B}(b_1) = \mathcal{B}(b_2) \\ p_1 \models\!\mid p_2 &\iff \mathcal{R}p_1 = \mathcal{R}p_2 \end{aligned}$$

Im Vergleich mit typtheoretischen Sprachen ist die Syntax von IMP außerordentlich beschränkt, da es keine Variablen gibt.

Abbildung 6.7 zeigt einige Äquivalenzen für reguläre Programme. Einige der Äquivalenzen folgen aus den Gleichungen von Proposition 6.4.1.

Mithilfe der Äquivalenzen in Abbildung 6.7 kann man weitere Äquivalenzen zeigen. Hier ist ein Beispiel:

Proposition 6.4.4 *Sei $b \in Bexp$ und $p \in Pro$. Dann gilt:*

$$\text{while } b \text{ do } p \;\models\; \text{if } b \text{ then } p; \text{ while } b \text{ do } p \text{ else skip}$$

Beweis

$$\begin{aligned}
&\text{if } b \text{ then } p; \text{ while } b \text{ do } p \text{ else skip} \\
&\models\; b?;p;(b?;p)^*;\neg b? + \neg b?;\text{skip} && \text{Definition if} \\
&\models\; b?;p;(b?;p)^*;\neg b? + \text{skip};\neg b? && \text{(6.7) und (6.8)} \\
&\models\; ((b?;p);(b?;p)^* + \text{skip});\neg b? && \text{(6.6)} \\
&\models\; (\text{skip} + (b?;p);(b?;p)^*);\neg b? && \text{(6.3)} \\
&\models\; (b?;p)^*;\neg b? && \text{(6.14)} \\
&\models\; \text{while } b \text{ do } p && \text{Definition while}
\end{aligned}$$

□

Reguläre Programme liefern eine neue Sicht auf die durch IMP beschreibaren Relationen. Anders als die Primitive von IMP, die auf Ausführbarkeit ausgerichtet sind, sind die Primitive von IMP so gewählt, dass sie möglichst natürliche Beschreibungen für Relationen darstellen. Das hat zur Folge, dass die denotationale Semantik von regulären Programmen viel einfacher ist als die von IMP. Insbesondere wird die in IMP für Schleifen erforderliche Fixpunktkonstruktion überflüssig, da die Iteration regulärer Programme durch eine Standardoperation für Relationen beschrieben werden kann (reflexiv transitiven Abschluss).

Konzeptuell gesehen ist es am effizientesten, zuerst reguläre Programme einzuführen und dann IMP einfach als Teilsprache zu identifizieren. Reguläre Programme wurden zuerst von Vaughan Pratt [1976] betrachtet.