Kapitel 7

Programmverifikation

Es ist üblich, zwischen der Spezifikation und der Implementierung eines Dienstes zu unterscheiden. Eine Spezifikation sagt, was ein Dienst leisten soll. Eine Implementierung realisiert einen Dienst mithilfe von Sofware und/oder Hardware. Wenn eine Spezifikation eines Dienstes vorliegt, ist es sinnvoll, für eine Implementierung des Dienstes zu *verifizieren*, dass sie die Spezifikation erfüllt.

Im Folgenden betrachten wir Dienste, die durch reguläre Programme implementiert werden können. Diese Dienste können mithilfe einer Vor- und einer Nachbedingung spezifiziert werden. Wir betrachten eine Methode, mit der gezeigt werden kann, dass ein Programm eine Spezifikation bis auf Terminierung erfüllt. Die Methode arbeitet mit so genannten *Schleifeninvarianten* und hat ihren Ursprung in den Arbeiten von Floyd [1967] und Hoare [1969].

Die in diesem Kapitel entwickelte Theorie wird oft unter der Überschrift *Hoaresche Logik* geführt.

7.1 Spezifikation mit Vor- und Nachbedingungen

Stellen Sie sich vor, dass wir ein Programm schreiben wollen, dass das Produkt zweier ganzer Zahlen berechnet. Dabei nehmen wir an, dass eine der zwei Zahlen nicht negativ ist. Die durch das Programm zu berechnende Relation können wir mithilfe einer Vor- und einer Nachbedingung spezifizieren:

Vorbedingung: $\{X \geq 0\}$
Nachbedingung: $\{Z = X \cdot Y\}$

Die Vorbedingung spezifiziert, für welche Anfangszustände das Programm korrekte Ergebnisse liefern soll. Die Nachbedingung spezifiziert, welche Eigenschaften die Endzustände haben sollen, die das Programm für zulässige Anfangszustände liefert. Wenn wir zusätzlich verlangen, dass das Programm die Lokationen X und Y nicht zuweist, haben wir in der Tat eine brauchbare Spezifikation für das gewünschte Programm.

Es ist einfach, Spezifikationen und ihren Zusammenhang mit regulären Programmen zu formalisieren. Bedingungen formalisieren wir als Zustandsmengen:

$A, B \subseteq \Sigma$ **Bedingung**

Ein **Zustand** σ **erfüllt eine Bedingung** B, wenn $\sigma \in B$ gilt. Eine Bedingung A ist **stärker** [**schwächer**] als eine Bedingung B, wenn $A \subseteq B$ [$A \supseteq B$] gilt. Um Bedingungen bequem schreiben zu können, verwenden wir eine Notation, die wir mit einem Beispiel einführen:

$$\{Z = X \cdot Y\} \rightsquigarrow \{\sigma \in \Sigma \mid \sigma Z = \sigma X \cdot \sigma Y\}$$

Unter einer **Spezifikation** verstehen wir ein Paar (A, B) aus zwei Bedingungen. In der Praxis ist es sinnvoll, neben der Angabe der Vor- und Nachbedingung zusätzlich festlegen zu können, welche Variablen das Programm nicht zuweisen darf.

Im Folgenden verstehen wir unter einem Programm immer ein reguläres Programm.

Sei p ein Programm, σ ein Zustand, und seien A, B Bedingungen. Wir definieren:

$$\begin{aligned}
p \textbf{ terminiert für } \sigma &\overset{\text{def}}{\iff} \sigma \in Dom(\mathcal{R}p) \\
p \textbf{ terminiert für } A &\overset{\text{def}}{\iff} A \subseteq Dom(\mathcal{R}p) \\
A \textbf{ Vorbedingung für } p \textbf{ und } B &\overset{\text{def}}{\iff} \forall(\sigma, \sigma') \in \mathcal{R}p\colon\ \sigma \in A \Longrightarrow \sigma' \in B \\
p \textbf{ erfüllt } (A, B) &\overset{\text{def}}{\iff} p \text{ terminiert für } A \\
&\qquad \text{und } A \text{ Vorbedingung für } p \text{ und } B \\
p \textbf{ erfüllt } (A, B) \textbf{ partiell} &\overset{\text{def}}{\iff} A \text{ Vorbedingung für } p \text{ und } B
\end{aligned}$$

Beachten Sie, dass $(\sigma, \sigma') \in \mathcal{R}p$ genau dann gilt, wenn p für σ den Endzustand σ' liefern kann. Wenn $\mathcal{R}p$ nicht funktional ist, ist es durchaus möglich, dass p für einen Zustand mehr als einen Endzustand liefern kann.

Ein Programm terminiert für einem Zustand, wenn es für diesen Zustand mindestens einen Endzustand liefern kann. Eine Bedingung ist eine Vorbedingung für ein Programm und eine Nachbedingung, wenn das Programm für die die Vorbedingung erfüllenden Zustände nur Endzustände liefern kann, die die Nachbedingung erfüllen.

Machen Sie sich klar, dass eine Spezifikation nur Aussagen über die durch ein Programm beschriebene Relation macht. Wie das Programm die Relation beschreibt spielt also für die Erfüllung einer Spezifikation keine Rolle. Beim Übergang vom Programm zur beschriebenen Relation geht die algorithmische Information verloren (zum Beispiel der Unterschied zwischen verschiedenen Multiplikationsmethoden).

Wenn ein Programm p eine Spezifikation (A, B) erfüllt, ist seine Relation $\mathcal{R}p$ durch die Spezifikation (A, B) nur unvollständig beschrieben. Insbesondere bleibt offen, wie sich $\mathcal{R}p$ für Anfangszustände verhält, die nicht in A sind. Außerdem ist es so, dass die Nachbedingung in der Praxis nur Aussagen über bestimmte Lokationen macht. Beispielsweise macht die Nachbedingung $\{Z = X \cdot Y\}$ nur eine Aussage über die Lokationen X, Y, und Z.

Gegeben eine Spezifikation, heißt ein Programm **partiell korrekt**, wenn es die Spezifikation partiell erfüllt, und **total korrekt**, wenn es die Spezifikation erfüllt.

Ein Blick auf die Definition von Vorbedingungen liefert, dass für jedes Programm und jede Nachbedingung eine eindeutig bestimmte schwächste Vorbedingung existiert. Das motiviert die Definition des so genannten **Notwendigkeitsoperators**:

$$\mathcal{N} \in \mathit{Pro} \to \mathcal{P}(X) \to \mathcal{P}(X)$$
$$\mathcal{N}pB = \{\, \sigma \in \Sigma \mid \forall \sigma' \in \Sigma\colon\ (\sigma, \sigma') \in \mathcal{R}p \Longrightarrow \sigma' \in B \,\}$$

Bei $\mathcal{N}pB$ handelt es sich gerade um die Menge aller Zustände, für die p nur Endzustände in B liefern kann.

Proposition 7.1.1 *Für jedes Programm p gilt:*

$$\mathcal{N}p\emptyset = \{\, \sigma \mid \neg\exists \sigma' \in \Sigma\colon\ (\sigma, \sigma') \in \mathcal{R}p \,\}$$
$$\mathit{Dom}\,(\mathcal{R}p) = \Sigma - \mathcal{N}p\emptyset$$

Proposition 7.1.2 *Sei p ein Programm, σ ein Zustand, und seien A, B Bedingungen. Dann gilt:*

$$\begin{aligned}
A \text{ Vorbedingung für } p \text{ und } B &\iff A \subseteq \mathcal{N}pB \\
p \text{ terminiert für } \sigma &\iff \sigma \notin \mathcal{N}p\emptyset \\
p \text{ terminiert für } A &\iff A \cap \mathcal{N}p\emptyset = \emptyset \\
p \text{ erfüllt } (A, B) \text{ partiell} &\iff A \subseteq \mathcal{N}pB \\
p \text{ erfüllt } (A, B) \text{ total} &\iff A \subseteq \mathcal{N}pB \text{ und } A \cap \mathcal{N}p\emptyset = \emptyset
\end{aligned}$$

Die erste Äquivalenz der Proposition besagt gerade, dass $\mathcal{N}pB$ die schwächste Vorbedingung für p und B ist.

Insgesamt besagt die Proposition, dass sich alle bisher definierten Korrektheitseigenschaften auf Eigenschaften des Notwendigkeitsoperators zurückführen lassen.

In den nachfolgenden Betrachtungen wird der Notwendigkeitsoperator die Hauptrolle spielen. Wir streben ein Verfahren an, mit dem wir für gegebenes p und B die Menge $\mathcal{N}pB$ möglichst explizit darstellen können. Damit werden wir

eine praktikable Verifikationsmethode erhalten, mit der wir zeigen können, dass ein Programm seine Spezifikation partiell erfüllt.

Die folgende Proposition stellt die verschiedenen Sprechweisen und Charakterisierungen für den Begriff der partiellen Korrektheit zusammen.

Proposition 7.1.3 *Für jedes Programm p und alle Bedingungen A, B sind die folgenden Aussagen äquivalent:*

1. *p erfüllt (A, B) partiell*
2. *A Vorbedingung für p und B*
3. $\forall (\sigma, \sigma') \in \mathcal{R}p\colon\ \sigma \in A \Longrightarrow \sigma' \in B$
4. $A \subseteq \mathcal{N}pB$

7.2 Notwendigkeitsoperatoren

Der Notwendigkeitsoperator für Programme ist eine Variante eines Mitglieds einer allgemeinen Klasse von Notwendigkeitsoperatoren. Wir zeigen jetzt einige Eigenschaften von allgemeinen Notwendigkeitsoperatoren, die wir später für den speziellen Notwendigkeitsoperator für Programme benötigen.

Sei X eine Menge. Der **Notwendigkeitsoperator für** X ist wie folgt definiert:

$$\mathcal{N} \in \mathcal{P}(X^2) \to \mathcal{P}(X) \to \mathcal{P}(X)$$
$$\mathcal{N}RB = \{\, x \in X \mid \forall x' \in X\colon\ (x, x') \in R \Longrightarrow x' \in B \,\}$$

Der Notwendigkeitsoperator für X liefert für eine binäre Relation $R \subseteq X^2$ und eine Teilmenge $B \subseteq X$ die Menge aller $x' \in X$, sodass R für x' nur Elemente von B liefert.

Sei im Folgenden X eine Menge und $\mathcal{N}$ der Notwendigkeitsoperator für X.

Proposition 7.2.1 (Monotonie) *$\mathcal{N}$ ist antimonoton im ersten und monoton im zweiten Argument. Mit anderen Worten, für alle $R, R' \subseteq X^2$ und $B, B' \subseteq X$ gilt:*

$$B \subseteq B' \implies \mathcal{N}RB \subseteq \mathcal{N}RB'$$
$$R \subseteq R' \implies \mathcal{N}RB \supseteq \mathcal{N}R'B$$

Proposition 7.2.2 (Distribution) *Sei S eine nichtleere Menge von binären Relationen auf X und $B \subseteq X$. Dann gilt:*

$$\mathcal{N}(\bigcup_{R \in S} R)B = \bigcap_{R \in S} \mathcal{N}RB$$

Beweis Für alle x gilt:

$$\begin{aligned}
x \in \mathcal{N}(\bigcup_{R\in S} R)B &\iff \forall x' \in X\colon\ (x,x') \in \bigcup_{R\in S} R \Longrightarrow x' \in B \\
&\iff \forall x' \in X\colon\ (\exists R \in S\colon\ (x,x') \in R) \implies x' \in B \\
&\iff \forall x' \in X\ \forall R \in S\colon\ (x,x') \in R \Longrightarrow x' \in B \\
&\iff \forall R \in S\ \forall x' \in X\colon\ (x,x') \in R \Longrightarrow x' \in B \\
&\iff \forall R \in S\colon\ x \in \mathcal{N}RB \\
&\iff x \in \bigcap_{R\in S} \mathcal{N}RB
\end{aligned}$$

□

Proposition 7.2.3 (Elimination) *Für alle $R, R' \subseteq X^2$ und $B, B' \subseteq X$ gilt:*

$$\begin{aligned}
\mathcal{N}(Id(B))B' &= (X - B) \cup B' \\
\mathcal{N}(R \circ R')B &= \mathcal{N}R(\mathcal{N}R'B) \\
\mathcal{N}(R \cup R')B &= \mathcal{N}RB \cap \mathcal{N}R'B \\
\mathcal{N}RX &= X \\
\mathcal{N}R\emptyset &= X - Dom\, R
\end{aligned}$$

Proposition 7.2.4 (Iteration) *Für alle $R \subseteq X^2$ und $B \subseteq X$ gilt:*

1. $\mathcal{N}R^*B = \bigcap_{n\in\mathbb{N}} \mathcal{N}R^nB$.
2. $\mathcal{N}R^*B \subseteq B$.
3. $\mathcal{N}R^*B$ *ist die größte Menge* $I \subseteq B$ *mit* $I \subseteq \mathcal{N}RI$.
4. $\mathcal{N}R^*B = \bigcup\{\, I \subseteq B \mid I \subseteq \mathcal{N}RI \,\}$.
5. *Sei* $B \subseteq \mathcal{N}RB$. *Dann* $B \subseteq \mathcal{N}RB \subseteq \mathcal{N}R^2B \subseteq \mathcal{N}R^3B \subseteq \ldots$.
6. $B \subseteq \mathcal{N}RB \iff B = \mathcal{N}R^*B$.

Beweis *Teil 1.* Folgt aus $R^* = \bigcup_{n\in\mathbb{N}} R^n$ und Proposition 7.2.2.

Teil 2. Folgt aus (1), da $\mathcal{N}R^0B = B$.

Teil 3. Wir zeigen zuerst, dass $\mathcal{N}R^*B$ eine Menge mit den gewünschten Eigenschaften ist. Teil (2) liefert die erste Eigenschaft. Es bleibt $\mathcal{N}R^*B \subseteq \mathcal{N}R(\mathcal{N}R^*B)$ zu zeigen. Da $R \circ R^* = R^+ \subseteq R^*$ gilt, folgt mit der Antimonotonie von $\mathcal{N}$ im ersten Argument: $\mathcal{N}R^*B \subseteq \mathcal{N}(R \circ R^*)B = \mathcal{N}R(\mathcal{N}R^*B)$.

Als Nächstes zeigen wir, dass $\mathcal{N}R^*B$ die größte Menge mit den gewünschten Eigenschaften ist. Sei $I \subseteq B$ mit $I \subseteq \mathcal{N}RI$. Wir müssen zeigen, dass $I \subseteq \mathcal{N}R^*B$.

Wegen (1) genügt es zu zeigen, dass $\forall n \in \mathbb{N}\colon\ I \subseteq \mathcal{N}R^nB$. Wir zeigen dies durch Induktion über n.

Sei $n = 0$. Dann $I \subseteq B = \mathcal{N}R^0B = \mathcal{N}R^nB$.

Sei $n > 0$. Dann liefert die Induktionsannahme $I \subseteq \mathcal{N}R^{n-1}B$. Also $I \subseteq \mathcal{N}RI \subseteq \mathcal{N}R(\mathcal{N}R^{n-1}B) = \mathcal{N}(R \circ R^{n-1})B = \mathcal{N}R^nB$.

Teil 4. Folgt unmittelbar aus (3).

Teil 5. Folgt mit Induktion aus der Monotonie von $\mathcal{N}$.

Teil 6. Die Richtung „$\Longrightarrow$“ folgt aus (1) und (5). Die andere Richtung folgt aus (3). □

Wenn R funktional ist, kann man R aus $\mathcal{N}R$ rekonstruieren:

Proposition 7.2.5 (Funktionale Rekonstruktion) *Für alle $R \in X \rightharpoonup X$ gilt:*

$$R = \{\, (x, x') \in X^2 \mid x \in \mathcal{N}R\{x'\} - \mathcal{N}R\varnothing \,\}$$

Beweis Sei $K = \{\, (x, x') \in X^2 \mid x \in \mathcal{N}R\{x'\} - \mathcal{N}R\varnothing \,\}$. Zunächst stellen wir fest, dass

$$K = \{\, (x, x') \in X^2 \mid x \in \mathit{Dom}\, R \cap \mathcal{N}R\{x'\} \,\}$$

gilt. Also ist K funktional und $K \subseteq R$. Da R funktional ist, folgt $R \subseteq K$. Also $R = K$. □

Thomas Bieler, ein Student der Vorlesung, hat im Juli 2003 gezeigt, dass sich auch eine beliebige Relation R aus $\mathcal{N}R$ rekonstruieren lässt:

Proposition 7.2.6 (Relationale Rekonstruktion) *Für alle $R \subseteq X^2$ gilt:*

$$R = \bigcup_{M \subseteq X} D_M \times M \quad \textit{mit}\ D_M = \mathcal{N}RM - \bigcup_{x \in M} \mathcal{N}R(M - \{x\})$$

Beweis „$\subseteq$“. Sei $(x, x') \in R$. Sei $M = \{\, x'' \in X \mid (x, x'') \in R \,\}$. Dann $x \in \mathcal{N}RM$ und $\forall x'' \in M\colon x \notin \mathcal{N}R(M - \{x''\})$. Also $(x, x') \in D_M \times M$.

„$\supseteq$“. Sei $M \subseteq X$, $x \in \mathcal{N}RM$, $\forall x'' \in M\colon x \notin \mathcal{N}R(M - \{x''\})$ und $x' \in M$. Dann $(x, x') \in R$. □

7.3 Bestimmung schwächster Vorbedingungen

Sei $\mathcal{N}$ der Notwendigkeitsoperator für Programme und $\mathcal{N}_\Sigma$ der Notwendigkeitsoperator für die Menge Σ aller Zustände.

$$\mathcal{N}' \in Pro \to \mathcal{P}(X) \to \mathcal{P}(X)$$

$$\begin{aligned}
\mathcal{N}'(X := a)B &= B[X := a] \\
\mathcal{N}'(b?)B &= b \Rightarrow B \\
\mathcal{N}'(p;p')B &= \mathcal{N}'p(\mathcal{N}'p'B) \\
\mathcal{N}'(p + p')B &= \mathcal{N}'pB \cap \mathcal{N}'p'B \\
\mathcal{N}'p^*B &= \bigcup\{ I \subseteq B \mid I \subseteq \mathcal{N}'pI \}
\end{aligned}$$

Abbildung 7.1: Definition von $\mathcal{N}'$

Proposition 7.3.1 *Für jedes Programm p und jede Bedingung B gilt:*

$$\mathcal{N}pB = \mathcal{N}_\Sigma(\mathcal{R}p)B$$

Proposition 7.3.2 (Monotonie) *Für alle Programme p, p' und alle Bedingungen B, B' gilt:*

$$\begin{aligned}
B \subseteq B' &\implies \mathcal{N}pB \subseteq \mathcal{N}pB' \\
\mathcal{R}p \subseteq \mathcal{R}p' &\implies \mathcal{N}pB \supseteq \mathcal{N}p'B
\end{aligned}$$

Beweis Folgt aus Proposition 7.2.1. □

Sei B eine Bedingung, X eine Lokation, a ein arithmetischer Ausdruck und b ein Boolescher Ausdruck. Wir definieren drei Notationen:

$$\begin{aligned}
B[X := a] &\stackrel{\text{def}}{=} \{ \sigma \in \Sigma \mid \sigma[X := \mathcal{A}(a)\sigma] \in B \} \\
b \Rightarrow B &\stackrel{\text{def}}{=} \{ \sigma \in \Sigma \mid \mathcal{B}(b)\sigma = 0 \ \vee \ \sigma \in B \} \\
B \wedge b &\stackrel{\text{def}}{=} \{ \sigma \in B \mid \mathcal{B}(b)\sigma = 1 \}
\end{aligned}$$

Abbildung 7.1 definiert eine Funktion $\mathcal{N}'$ durch strukturelle Rekursion.

Satz 7.3.3 (Koinzidenz) $\mathcal{N} = \mathcal{N}'$.

Beweis Es genügt zu zeigen: $\forall p \in Pro \ \forall B \subseteq \Sigma$: $\mathcal{N}'pB = \mathcal{N}_\Sigma(\mathcal{R}p)B$. Dies gelingt durch strukturelle Induktion über p. Der Beweis orientiert sich an den definierenden Gleichungen von $\mathcal{N}'$. Tests sowie sequentielle und parallele Komposition werden mit den Gleichungen von Proposition 7.2.3 behandelt. Iteration wird mit Teil (4) von Proposition 7.2.4 behandelt. □

Der Koinzidenzsatz und die definierenden Gleichungen von $\mathcal{N}'$ versetzen uns in die Lage, schwächste Vorbedingungen schrittweise zu bestimmen. Wir zeigen dies an zwei Beispielen.

Proposition 7.3.4 $\mathcal{N}(\text{if } b \text{ then } p \text{ else } p')B = (b \Rightarrow \mathcal{N}pB) \cap (\neg b \Rightarrow \mathcal{N}p'B)$.

Beweis Mit dem Koinzidenzsatz und der Definition von $\mathcal{N}'$ folgt:

$$\begin{aligned}\mathcal{N}(\text{if } b \text{ then } p \text{ else } p')B &= \mathcal{N}(b?;p + \neg b?;p')B\\ &= \mathcal{N}(b?;p)B \cap \mathcal{N}(\neg b?;p')B\\ &= \mathcal{N}(b?)(\mathcal{N}pB) \cap \mathcal{N}(\neg b?)(\mathcal{N}p'B)\\ &= (b \Rightarrow \mathcal{N}pB) \cap (\neg b \Rightarrow \mathcal{N}p'B)\end{aligned}$$

□

Proposition 7.3.5 $\mathcal{N}(\text{while } b \text{ do } p)B = \bigcup\{ I \subseteq (\neg b \Rightarrow B) \mid I \subseteq (b \Rightarrow \mathcal{N}pI) \}$.

Beweis Mit dem Koinzidenzsatz und der Definition von $\mathcal{N}'$ folgt:

$$\begin{aligned}\mathcal{N}(\text{while } b \text{ do } p)B &= \mathcal{N}((b?;p)^*;\neg b?)B\\ &= \mathcal{N}(b?;p)^*(\mathcal{N}(\neg b?)B)\\ &= \mathcal{N}(b?;p)^*(\neg b \Rightarrow B)\\ &= \bigcup\{ I \subseteq (\neg b \Rightarrow B) \mid I \subseteq \mathcal{N}(b?;p)I) \}\\ &= \bigcup\{ I \subseteq (\neg b \Rightarrow B) \mid I \subseteq (b \Rightarrow \mathcal{N}pI) \}\end{aligned}$$

□

Zuweisung und Substitution
Es besteht ein wichtiger Zusammenhang zwischen Zuweisung und Substitution. Dieser wird durch die Gleichung

$$\mathcal{N}(X := a)B = B[X := a]$$

hergestellt. Wenn man die Bedingungen durch Formeln darstellt, erhält man die Darstellung der Bedingung $B[X := a]$ gerade dadurch, dass man die Substitution $[X := a]$ auf die B darstellende Formel anwendet. Die Zulässigkeit dieses Vorgehens beruht im Wesentlichen auf dem Substitutionslemma für Formeln. Sei $\mathcal{D}$ die Denotationsfunktion für Formeln. Dann gilt:

$$\begin{aligned}&\{ \sigma \in \Sigma \mid \mathcal{D}F\sigma = 1 \}[X := a]\\ &= \{ \sigma \in \Sigma \mid \mathcal{D}F(\sigma[X := \mathcal{A}(a)\sigma]) = 1 \} && \text{Definition von } B[X := a]\\ &= \{ \sigma \in \Sigma \mid \mathcal{D}(F[X := a])\sigma = 1 \} && \text{Substitutionslemma}\end{aligned}$$

7.4 Vorbedingungsbestimmung mithilfe von Invarianten

Für die Bestimmung schwächster Vorbedingungen benötigen wir eine Technik, mit der wir die durch die Gleichung

$$\mathcal{N}p^*B \;=\; \bigcup\{\, I \subseteq B \mid I \subseteq \mathcal{N}pI \,\}$$

eingebrachten Vereinigungen eliminieren können. Dazu betrachten wir diese Situation genauer.

Sei p ein Programm und B eine Bedingung. Eine Bedingung I heißt **Invariante** von p, wenn $I \subseteq \mathcal{N}pI$ gilt. Eine Bedingung I heißt **zulässige Invariante** für p und B, wenn $I \subseteq \mathcal{N}pI$ und $I \subseteq B$ gilt.

Proposition 7.4.1 (Invarianten) *Für jedes Programm p und jede Bedingungen B gilt:*

1. *Jede zulässige Invariante für p und B ist eine Vorbedingung für p^* und B.*
2. *Die schwächste zulässige Invariante für p und B ist die schwächste Vorbedingung für p^* und B.*

Beweis Folgt aus Proposition 7.2.4. □

Wir wenden uns wieder der Gleichung

$$\mathcal{N}p^*B \;=\; \bigcup\{\, I \subseteq B \mid I \subseteq \mathcal{N}pI \,\}$$

zu. Wenn wir die schwächste zulässige Invariante I' für p und B kennen, ist die Vereinigung kein Problem, da dann

$$\mathcal{N}p^*B \;=\; I'$$

gilt. In der Praxis ist es jedoch oft ausreichend, eine hinreichend schwache Vorbedingung für p^* und B zu bestimmen. Die natürlichen Kandidaten dafür sind die zulässigen Invarianten für p und B. Wegen der Monotonie des Notwendigkeitsoperators liefert dieses Vorgehen auch insgesamt eine Vorbedingung.

Eine **partielle Korrektheitsaussage** ist ein Tripel (A, p, B) aus einem Programm p und zwei Bedingungen A und B. Eine partielle Korrektheitsaussage (A, p, B) ist **gültig**, wenn $A \subseteq \mathcal{N}pB$ gilt. Eine partielle Korrektheitsaussage (A, p, B) ist also genau dann gültig, wenn p die Spezifikation (A, B) partiell erfüllt. Wir werden partielle Korrektheitsaussagen (A, p, B) kurz als ApB schreiben.

$$\frac{}{B[a/X](X := a)B} \qquad \frac{}{(b \Rightarrow B)(b?)B}$$

$$\frac{Ap_1A' \quad A'p_2B}{A(p_1;p_2)B} \qquad \frac{A_1p_1B \quad A_2p_2B}{(A_1 \cap A_2)(p_1 + p_2)B} \qquad \frac{I \subseteq A \quad ApI \quad I \subseteq B}{Ip^*B}$$

Abbildung 7.2: Verifikationsregeln für reguläre Programme

Die in Abbildung 7.2 gezeigten Inferenzregeln für partielle Korrektheitsaussagen beschreiben das skizzierte Verfahren zur Bestimmung von Vorbedingungen.[1] Bei den Regeln handelt es sich um eine relationale Umformung der definierenden Gleichungen von $\mathcal{N}'$, wobei für Iterationen wie besprochen jede zulässige Invariante als Vorbedingung eingebracht werden kann. Das durch die Regeln beschriebene Verfahren durchläuft das vorgelegte Programm mit struktureller Rekursion. Wahlmöglichkeiten gibt es nur bei Iterationen, da hier jede zulässige Invariante als Vorbedingung gewählt werden kann. Wenn man jeweils die schwächste zulässige Invariante wählt, bekommt man insgesamt die schwächste Vorbedingung.

Sei $IPC \subseteq \mathcal{P}(\Sigma) \times Pro \times \mathcal{P}(\Sigma)$ die durch die Inferenzregeln in Abbildung 7.2 definierte Menge.

Proposition 7.4.2 (Korrektheit) *Für alle $ApB \in IPC$ gilt: $A \subseteq \mathcal{N}pB$.*

Beweis Durch Regelinduktion mithilfe des Koinzidenzsatzes, der Definition von $\mathcal{N}'$ und der Monotonie von $\mathcal{N}$. □

Proposition 7.4.3 (Vollständigkeit) *Für jedes Programm p und jede Bedingungen B gilt: $(\mathcal{N}pB)pB \in IPC$.*

Beweis Durch strukturelle Induktion über p mithilfe des Koinzidenzsatzes und der Definition von $\mathcal{N}'$. Für eine Iteration p^* und eine Nachbedingung B verwenden wir $\mathcal{N}p^*B$ als Invariante. Dabei ist Proposition 7.2.4 hilfreich. □

Satz 7.4.4 (Verifikation) *Für jedes Programm p und alle Bedingungen A, B gilt: A ist genau dann eine Vorbedingung für p und B, wenn es eine Bedingung A' gibt, sodass $A \subseteq A'$ und $A'pB \in IPC$.*

Beweis Folgt aus den Propositionen 7.4.2 und 7.4.3. □

[1] Für Experten: Bei den Regeln handelt es sich um eine gerichtete Variante der Hoareschen Regeln.

Korollar 7.4.5 (Rekonstruktion) *Für jedes Programm p und jede Bedingung B gilt: $\mathcal{N}pB = \bigcup\{ A \subseteq \Sigma \mid ApB \in IPC \}$.*

Beweis Folgt aus den Propositionen 7.4.2 und 7.4.3. □

Proposition 7.4.6 *Sei p ein Programm ohne Iterationen. Dann gilt für alle Bedingungen A, B: $ApB \in IPC \Longrightarrow A = \mathcal{N}pB$.*

Beweis Durch Regelinduktion mithilfe des Koinzidenzsatzes und der Definition von $\mathcal{N}'$. □

7.5 Verifikationsbedingungen

Die Seitenbedingungen $I \subseteq A$ und $I \subseteq B$ der Regel für Iterationen

$$\frac{I \subseteq A \quad ApI \quad I \subseteq B}{Ip^*B}$$

werden als **Verifikationsbedingungen** bezeichnet.

Wir betrachten jetzt das Programm

$$p = C := X;\ Z := 0;\ ((C \geq 1)?;\ Z := Z + Z;\ C := C - 1)^*;\ (C < 1)?$$

Mithilfe der Verifikationsregeln in Abbildung 7.2 sind wir in der Lage, für p und eine Spezifikation (A, B) die folgenden Verifikationsbedingungen abzuleiten:

$$I \subseteq (C < 1) \Rightarrow B$$
$$I \subseteq (C \geq 1) \Rightarrow I[C := C - 1][Z := Z + Y]$$
$$A \subseteq I[Z := 0][C := X]$$

Die zwei ersten Verifikationsbedingungen ergeben sich für die Iteration, und die dritte Verfikationsbedingung stellt den Zusammenhang her zwischen der durch Spezifikation eingebrachten Bedingung A und der abgeleiteten Vorbedingung. Es ist nun so, dass p die Spezifikation (A, B) genau dann partiell erfüllt, wenn eine Bedingung I existiert, so dass alle Verifikationsbedingungen erfüllt sind. Bei I handelt es sich also um eine existenziell quantifizierte Variable, die die für die Iteration zu verwendende Invariante darstellt.

Zusammenfassend stellen wir fest, dass wir für ein gegebenes Programm Verifikationsbedingungen ableiten können, ohne die Spezifikation zu kennen. Dabei wird die Spezifikation durch zwei Variablen A, B dargestellt. Zusätzlich wird für jede Iteration eine existenziell quantifizierte Variable I eingeführt, die die für die Iteration zu verwendende Invariante darstellt. Jetzt erfüllt das Programm eine Spezifikation (A, B) genau dann partiell, wenn für die existenziell quantifizierten

$$\frac{ApB \qquad A'p_2B}{(b \Rightarrow A) \cap (\neg b \Rightarrow A')\ (\text{if } b \text{ then } p \text{ else } p')\ B}$$

$$\frac{I \wedge b \subseteq A \quad ApI \quad I \wedge \neg b \subseteq B}{I\ (\text{while } b \text{ do } p)\ B}$$

Abbildung 7.3: Verifikationsregeln für Konditionale und Schleifen

Variablen für die Invarianten Zustandsmengen existieren, sodass alle Verifikationsbedingungen erfüllt sind.

Abbildung 7.3 zeigt zwei Verifikationsregeln für Konditionale und Schleifen, die aus den Basisregeln in Abbildung 7.2 abgeleitet sind. Bei der Regel für Schleifen wird die Bedingung I als **Schleifeninvariante** bezeichnet, und die Seitenbedingungen $I \wedge b \subseteq A$ und $I \wedge \neg b \subseteq B$ als **Verifikationsbedingungen**.

Wir zeigen jetzt, wie die Regel für Schleifen aus den Basisregen abgeleitet ist. Da $(\text{while } b \text{ do } p) = (b?;p)^*;\neg b?$, stehen wir vor dem Problem, mit den Basisregeln eine gültige Vorbedingung für $(b?;p)^*;\neg b?$ und B abzuleiten. Dafür gibt es nur die folgende Möglichkeit:

$$\dfrac{\dfrac{I \subseteq b \Rightarrow A \qquad \dfrac{\dfrac{}{(b \Rightarrow A)(b?)A} \quad ApI}{(b \Rightarrow A)(b?;p)I} \qquad I \subseteq \neg b \Rightarrow B}{I\ ((b?;p)^*)\ (\neg b \Rightarrow B)} \qquad \dfrac{}{(\neg b \Rightarrow B)\ (\neg b?)\ B}}{I\ ((b?;p)^*;\neg b?)\ B}$$

Damit bekommen wir die Verifikationsregel für Schleifen, da gilt:

$$\begin{aligned} I \subseteq b \Rightarrow A &\iff I \wedge b \subseteq A \\ I \subseteq \neg b \Rightarrow B &\iff I \wedge \neg b \subseteq B \end{aligned}$$

Aufgabe 7.1 Leiten Sie die Verifikationsregel für Konditionale aus den Basisregeln ab. □

7.6 Verifikationsbeispiel: Multiplizieren

Unser Ziel ist ein Programm, das das Produkt zweier Zahlen bestimmt. Wir beginnen mit der Nachbedingung

$$B = \{Z = X \cdot Y\}$$

Damit das Programm nicht zu kompliziert wird, verwenden wir die Vorbedingung

$$A = \{X \geq 0\}$$

Mit dieser Vorbedingung können wir das Produkt $X \cdot Y$ durch X-maliges Aufaddieren von Y berechnen:

$$X \cdot Y = 0 \underbrace{+Y + \cdots + Y}_{X\text{-mal}}$$

Diese Idee liefert das folgende Programm:

```
C:=X; Z:=0; while C≥1 do (Z:=Z+Y; C:=C-1)
```

Um zu zeigen, dass das Programm die Spezifikation (A, B) partiell erfüllt, bestimmen wir mithilfe der Verifikationsregeln die Verifikationsbedingungen:

(1) $I \cap \{C < 1\} \subseteq B$
(2) $I \cap \{C \geq 1\} \subseteq I[C := C - 1][Z := Z + Y]$
(3) $A \subseteq I[Z := 0][C := X]$

Dabei stellt I eine noch zu wählende Invariante der Schleife dar. Jetzt instanziieren wir die Verifikationsbedingungen mit unserer Spezifikation (A, B):

(1) $I \cap \{C < 1\} \subseteq \{Z = X \cdot Y\}$
(2) $I \cap \{C \geq 1\} \subseteq I[C := C - 1][Z := Z + Y]$
(3) $\{X \geq 0\} \subseteq I[Z := 0][C := X]$

Als Nächstes wählen wir die Invariante

$$I \stackrel{\text{def}}{=} \{X \cdot Y = Z + C \cdot Y \ \wedge \ C \geq 0\}$$

und rechnen nach, dass die Verifikationsbedingungen erfüllt sind.

Bedingung (1) folgt mit

$$\begin{aligned} I \cap \{C < 1\} &= \{X \cdot Y = Z + C \cdot Y \ \wedge \ C \geq 0 \wedge \ C < 1\} \\ &= \{X \cdot Y = Z + C \cdot Y \ \wedge \ C = 0\} \\ &= \{Z = X \cdot Y \ \wedge \ C = 0\} \\ &\subseteq \{Z = X \cdot Y\} \end{aligned}$$

Bedingung (2) folgt mit

$$\begin{aligned} I[C := C - 1][Z := Z + Y] &= \{X \cdot Y = (Z + Y) + (C - 1) \cdot Y \ \wedge \ (C - 1) \geq 0\} \\ &= \{X \cdot Y = Z + C \cdot Y \ \wedge \ C \geq 1\} \\ &= I \cap \{C \geq 1\} \end{aligned}$$

Bedingung (3) folgt mit

$$\begin{aligned} I[Z := 0][C := X] &= \{X \cdot Y = 0 + X \cdot Y \ \wedge \ X \geq 0\} \\ &= \{X \geq 0\} \end{aligned}$$

Damit ist gezeigt, dass unser Programm unsere Spezifikation partiell erfüllt.

Bemerkungen

In diesem Kapitel haben wir Spezifikationen für reguläre Programme betrachtet, die mit einer Vor- und einer Nachbedingung arbeiten. Wir haben eine Technik entwickelt, mit der man verifizieren kann, dass ein Programm eine Spezikation partiell erfüllt. Dazu wird das Programm zunächst in ein endliches System von Verifikationsbedingungen überführt, das mithilfe von Variablen Bezug auf die Vor- und die Nachbedingung der Spezifikation nimmt. Zusätzlich beinhaltet das System für jede Iteration des Programms eine Variable, die einer zulässigen Invariante der Iteration entspricht. Das Programm erfüllt jetzt eine gegebene Spezifikation genau dann, wenn das zugehörige System lösbar ist, das heißt Zustandsmengen für die Invariantenvariablen so angegeben werden können, dass alle Bedingungen erfüllt sind.

Die hier entwickelte Theorie und Methode geht auf die Arbeiten von Floyd [1967], Hoare [1969] und vielen anderen zurück. Manchmal wird dieser Verfikationsansatz auch als *Hoaresche Logik* bezeichnet.

Unsere Darstellung unterscheidet sich von den in der Literatur üblichen dadurch, dass wir Bedingungen nicht durch Formeln, sondern durch beliebige Zustandsmengen darstellen. Das führt zu beträchtlichen Vereinfachungen. Insbesondere kann die Vollständigkeit der Methode jetzt (erwartungsgemäß) sehr einfach gezeigt werden. In einem zweiten Schritt, den wir im nächsten Kapitel ausführen werden, kann man innerhalb des hier vorgegebenen Rahmens studieren, was passiert, wenn man die Bedingungen durch Formeln einer logischen Sprache darstellt. Die übliche Darstellung von Hoaresche Logik findet man im Buch von Winskel [1993].

In der vorgestellten Theorie spielt der Notwendigkeitsoperator eine prominente Rolle. Dabei handelt es sich um einen modalen Operator im Sinne von modaler Logik. Diese Sicht geht auf Pratt [1976] und Fischer und Ladner [1979] zurück. Wenn sie voll ausgereizt wird, führt sie zur so genannten Dynamischen Logik, die im Buch von Harel, Kozen und Tiuryn [2000] ausführlich behandelt wird.

Floyd [1967] arbeitet noch mit durch Flussdiagrammen dargestellten Programmen und war der erste, der das Konzept der Invariante einführte. Hoare [1969] arbeitet mit IMP-Programmen. Beide hatten noch keine Semantik für Programme zur Verfügung. Hoare [1969] war der erste, der Verifikationsregeln angab. Von

ihm stammt die Idee, die Semantik von Programmen durch Verifikationsregeln zu definieren (so genannte *axiomaische Semantik*). Reguläre Programme mit ihrer einfachen denotationalen Semantik wurden von Fischer und Ladner [1979] eingeführt.

Glossar

Bedingungen, Zustand erfüllt Bedingung, schwächer und stärker.

Spezifikation, Programm erfüllt Spezifikation [partiell].

[Schwächste] Vorbedingung für Programm und Nachbedingung.

Programm terminiert für Zustand, für Bedingung.

Notwendigkeitsoperator.

[Zulässige] Invariante für ein Programm oder eine Schleife.

[Gültige] partielle Korrektheitsaussage.

Verifikationsbedingungen.