Dijkstras *wp*-Funktion als Formeltransformer und ihre Verbindung zur denotationellen Semantik

Rudolf Berghammer

Bericht Nr. 9708

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität Kiel
Preusserstraße 1-9
24105 Kiel

Zusammenfassung

In diesem Artikel wird der von R. Back stammende Ansatz, schwächste Vorbedingungen durch infinitären Formeln zu axiomatisieren, verfeinert, um auch Variablensubstitution in der gewohnten Weise handhabbar zu machen. Als Hauptresultat wird die Korrektheit einer rein syntaktischen *wp*-Funktion bezüglich des mittels denotationeller Semantik definierten Begriffs einer schwächsten Vorbedingung für totale Korrektheit bewiesen.

1 Einleitung

Neben dem Hoare-Kalkül [11] ist E.W. Dijkstras Kalkül der schwächsten Vorbedingungen [5, 6], kurz *wp*-Kalkül genannt, die bekannteste Methode zur axiomatischen Semantik von imperativen Programmen. Üblicherweise operiert die ihm zugrundeliegende *wp*-Funktion, neben den (syntaktischen) Programmen, noch auf semantischen Objekten wie Speichermengen oder Prädikaten auf Speichern. Manchmal wird auch zwischen Syntax und Semantik nicht exakt unterschieden, was, wie die Literatur zeigt, zu Ungereimtheiten führen kann.

Rein syntaktische Auffassungen von *wp* als Funktion auf Programmen und Formeln einer Logik sind selten. R. Back publizierte im Jahr 1981 einen Ansatz [1], in dem die, sich unmittelbar aus den originalen Gesetzen von [5, 6] ergebende, Funktion $wp : \mathcal{P} \times \mathcal{L}_{\omega_1\omega} \to \mathcal{L}_{\omega_1\omega}$ ein Programm und eine Formel erster Stufe mit möglicherweise abzählbar-unendlichen Disjunktionen und Konjunktionen in eine ebensolche Formel abbildet. Zur Rechtfertigung des Ansatzes wird in [1] auch demonstriert, daß für Programme mit einer Schleife die üblichen finitären Formeln erster Stufe zu schwach zur Axiomatisierung von schwächsten Vorbedingungen sind.

In diesem Artikel wird der Ansatz aus [1] etwas verfeinert, um auch Variablensubstitution in infinitären Formeln in der gewohnten Weise handhabbar zu machen. Diese Problematik wird in [1] nicht angesprochen. Als eigentliches Resultat wird die Verbindung zwischen der *wp*-Axiomatisierung und der denotationellen Semantik hergestellt, d.h. genaugenommen, die Korrektheit der *wp*-Funktion bezüglich des mittels denotationeller Semantik definierten Begriffs „schwächste Vorbedingung für totale Korrektheit“ bewiesen. Dabei beschränken wir uns, der Einfachheit halber, auf deterministische Programme, d.h. die klassische Programmiersprache der While-Programme.

Bei der eben erwähnten Korrektheitsaussage scheint es sich um ein „folk statement“ im Sinne von D. Harels Artikel [10] zu handeln. Ein formaler Beweis, wie er nachfolgend präsentiert wird, konnte jedoch in der Literatur nicht gefunden werden.

Der Artikel ist wie folgt gegliedert. In Abschnitt 2 definieren wir zuerst syntaktisch die Sprache $\mathcal{PRG}$ der While-Programme und zu einem Programm $s \in \mathcal{PRG}$ eine denotationelle Semantik $[\![s]\!] : \mathsf{Store}^\perp \to \mathsf{Store}^\perp$ als Funktion auf der flachen Cpo $\mathsf{Store}^\perp$ der Speicher. Weiterhin führen wir die infinitäre Formelsprache $\mathcal{L}_{\omega_1\omega}$ ein und für $\sigma \in \mathsf{Store}$ und $\varphi \in \mathcal{L}_{\omega_1\omega}$ die Gültigkeitsbegriffe $\models \varphi[\sigma]$ und $\models \varphi$. Aufbauend auf Semantik und Gültigkeit ist es dann möglich, die totale Korrektheit $[\psi]\, s\, [\varphi]$ des While-Programs $s \in \mathcal{PRG}$ bezüglich der Vorbedingung $\psi \in \mathcal{L}_{\omega_1\omega}$ und der Nachbedingung $\varphi \in \mathcal{L}_{\omega_1\omega}$ und auch den Begriff einer schwächsten Vorbedingung für totale Korrektheit formal festzulegen. Abschnitt 3 behandelt die *wp*-Funktion. Wird diese, analog zu [1], induktiv über den Aufbau der While-Programme definiert, so tritt im Fall der Zuweisung eine Substitution eines Terms für alle (freien) Vorkommen einer Programmvariablen in einer Formel auf. Substitution in $\mathcal{L}_{\omega_1\omega}$-Formeln kann bei einer, wie üblich als abzählbar-unendlich vorausgesetzten, Variablenmenge jedoch problematisch werden. Um sie als offensichtliche Erweiterung der klassischen einfachen Art mit ggf. „frischen“ Variablen festlegen zu können, betrachten wir deshalb zuerst die Menge $\mathcal{ASS}$ der Formeln $\varphi \in \mathcal{L}_{\omega_1\omega}$ mit endlicher Menge $var(\varphi)$ darin vorkommender Variablen. Dann beweisen wir, daß die aus [1] sich ergebende Restriktion

$$wp : \mathcal{PRG} \times \mathcal{ASS} \to \mathcal{ASS} \tag{1}$$

der *wp*-Funktion auf $\mathcal{ASS}$ wohldefiniert ist, d.h. für alle $s \in \mathcal{PRG}$ und $\varphi \in \mathcal{L}_{\omega_1\omega}$ die Endlichkeit von $var(\varphi)$ die Endlichkeit von $var(wp(s,\varphi))$ impliziert. Im Abschnitt 4 stellen wir schließlich die Verbindung zwischen der *wp*-Funktion (1) und der denotationellen Semantik von $\mathcal{PRG}$ her. Das entscheidende Ergebnis hierbei ist die Äquivalenz

$$\models wp(s,\varphi)[\sigma] \iff [\![s]\!](\sigma) \neq \perp \text{ und } \models \varphi[[\![s]\!](\sigma)]\,. \tag{2}$$

Zwei unmittelbare Folgerungen von (2) – das Hauptergebnis des Artikels – sind nämlich

$$[wp(s,\varphi)]\, s\, [\varphi] \qquad\qquad [\psi]\, s\, [\varphi] \implies \models \psi \to wp(s,\varphi)\,, \tag{3}$$

welche zeigen, daß die Formel $wp(s,\varphi)$ tatsächlich eine schwächste Vorbedingung für totale Korrektheit des While-Programms s zur der Nachbedingung φ ist. Weiterhin folgt aus (2) und (3) sofort die Äquivalenz von $\models \psi \to wp(s,\varphi)$ und $[\psi]\, s\, [\varphi]$. In Verbindung mit weiteren beweisbaren Eigenschaften, wie den „Healthiness conditions“ und dem „Loop invariant theorem“ (siehe etwa [6, 9]), wird $\models \psi \to wp(s,\varphi)$ üblicherweise verwendet, um totale Korrektheit zu verifizieren.

2 Programme und infinitäre Formeln

Zur syntaktischen Konstruktion der (schematischen) While-Programme gehen wir als Basis von einer Signatur $\Sigma = (S, K, F)$ mit Sortenmenge S, Konstantenmenge K und Operationenmenge F aus, wobei jeder Konstanten genau eine Sorte aus S und jeder Operation genau eine Funktionalität aus $S^+ \times S$ zugeordnet ist. Von dieser Signatur fordern wir nur, daß die Menge S die Sorte *bool* enthalte. Weiterhin nehmen wir eine abzählbar-unendliche Menge X von Variablen als gegeben an, welche die Vereinigung von paarweise disjunkten Mengen X_m, den Variablen der Sorte $m \in S$, ist. Mit $\mathcal{E}_m$ bezeichnen wir die Menge der Σ-Terme der Sorte $m \in S$ mit Variablen aus X. Die Familie $(\mathcal{E}_m)_{m\in S}$ dieser Termmengen sei wie üblich induktiv definiert.

2.1 Syntax der While-Programme. Die Menge $\mathcal{PRG}$ der While-Programme (oder Anweisungen) ist induktiv wie folgt definiert:

1. Die leere Anweisung `skip` ist ein While-Programm, d.h. $\texttt{skip} \in \mathcal{PRG}$.
2. Die undefinierte Anweisung `abort` ist ein While-Programm, d.h. $\texttt{abort} \in \mathcal{PRG}$.
3. Falls $x \in X_m$ und $t \in \mathcal{E}_m$, dann gilt auch $(x := t) \in \mathcal{PRG}$.
4. Falls $b \in \mathcal{E}_{bool}$ und $s_1, s_2 \in \mathcal{PRG}$, dann gilt auch $\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{fi} \in \mathcal{PRG}$.
5. Falls $b \in \mathcal{E}_{bool}$ und $s \in \mathcal{PRG}$, dann gilt auch $\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od} \in \mathcal{PRG}$.
6. Falls $s_1, s_2 \in \mathcal{PRG}$, dann gilt auch $(s_1; s_2) \in \mathcal{PRG}$. ■

Den üblichen Konventionen folgend, lassen wir in Zukunft die Klammern der Fälle 3 und 6 weg. Weiterhin bezeichnen wir mit $var(s) \subseteq X$ die (endliche) Menge der in $s \in \mathcal{PRG}$ vorkommenden Variablen. Eine induktive Definition dieser Menge ist offensichtlich.

Die Interpretation der am Anfang erwähnten Signatur $\Sigma = (S, K, F)$ erfolgt durch eine Σ-Algebra $A = ((m^A)_{m \in S}, (c^A)_{c \in K}, (f^A)_{f \in F})$. In ihr werden den Sorten m Mengen m^A zugeordnet, den Konstanten c Elemente c^A dieser Mengen und den Operationen f partielle Funktionen f^A auf diesen Mengen, so daß die Verträglichkeit mit den gegebenen Sorten bzw. Funktionalitäten gegeben ist. Details findet man in [17]. Wir wählen die Standardinterpretation $\mathbb{B} = \{tt, ff\}$ für die Sorte *bool* und die entsprechenden Operationen, insbesondere die Negationsoperation *not*.

Im folgenden benötigen wir einige elementare Begriffe aus der Fixpunkttheorie, wie (flache) Cpo, Monotonie, Stetigkeit, Kleinster-Fixpunkt-Operator μ, die wir beim Leser als bekannt voraussetzen. Andernfalls verweisen wir auf die Literatur, beispielsweise [14, 16].

Es bezeichne $i[m]$ die durch Adjunktion des kleinsten Elements $\bot_m$ an die Menge m^A entstehende flache Cpo. Weiterhin seien Store die Menge aller Funktionen $\sigma : X \to \bigcup_{m \in S} i[m]$, für die $\sigma(x) \in i[m]$ äquivalent ist zu $x \in X_m$, und $\mathsf{Store}^\bot$ die aus Store durch Adjunktion des kleinsten Elements $\bot$ entstehende flache Cpo. Wir nennen $\sigma \in \mathsf{Store}$ einen definierten Speicher und $\bot$ den undefinierten Speicher. Für $\sigma \in \mathsf{Store}$, $x \in X_m$ und $u \in i[m]$ ist der definierte Speicher $\sigma[x \leftarrow u]$ erklärt durch $\sigma[x \leftarrow u](x) = u$ und $\sigma[x \leftarrow u](y) = \sigma(y)$ für $y \neq x$. Schließlich bezeichne zu einem Term $t \in \mathcal{E}_m$ und einem Speicher $\sigma \in \mathsf{Store}^\bot$ der Ausdruck $[\![t]\!](\sigma)$ den Wert von t bezüglich σ. Dieser Wert ist ein Element aus $i[m]$. Wir verzichten hier auf eine formale induktive Definition von $[\![t]\!](\sigma)$ und erwähnen nur die entscheidende Striktheits-Eigenschaft $[\![t]\!](\bot) = \bot_m$.

2.2 Semantik der While-Programme. Die denotationelle Semantik von $p \in \mathcal{PRG}$ ist als Funktion $[\![p]\!] : \mathsf{Store}^\bot \to \mathsf{Store}^\bot$ über den Aufbau von p wie folgt festgelegt:

1. *Leere Anweisung*:
$$[\![\texttt{skip}]\!](\sigma) = \sigma$$
2. *Undefinierte Anweisung*:
$$[\![\texttt{abort}]\!](\sigma) = \bot$$
3. *Zuweisung*:
$$[\![x := t]\!](\sigma) = \begin{cases} \sigma[x \leftarrow [\![t]\!](\sigma)] & : \quad [\![t]\!](\sigma) \neq \bot_m \\ \bot & : \quad [\![t]\!](\sigma) = \bot_m \end{cases}$$
4. *Alternative*:
$$[\![\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{fi}]\!](\sigma) = \begin{cases} [\![s_1]\!](\sigma) & : \quad [\![b]\!](\sigma) = tt \\ [\![s_2]\!](\sigma) & : \quad [\![b]\!](\sigma) = ff \\ \bot & : \quad [\![b]\!](\sigma) = \bot_{bool} \end{cases}$$

5. *While-Schleife*:

$$[\![\texttt{while } b \texttt{ do } s \texttt{ od}]\!](\sigma) = \mu_\tau(\sigma)\,,$$

wobei das Funktional τ auf der Funktions-Cpo $(\mathsf{Store}^\perp \to \mathsf{Store}^\perp)$ der monotonen Funktionen von $\mathsf{Store}^\perp$ nach $\mathsf{Store}^\perp$ definiert ist durch

$$\tau[h](\rho) = \begin{cases} h([\![s]\!](\rho)) & : \quad [\![b]\!](\rho) = tt \\ \rho & : \quad [\![b]\!](\rho) = ff \\ \perp & : \quad [\![b]\!](\rho) = \perp_{bool}\,. \end{cases}$$

6. *Sequentielle Komposition*:

$$[\![s_1; s_2]\!](\sigma) = [\![s_2]\!]([\![s_1]\!](\sigma))$$ ■

Das bei der Semantikdefinition der While-Schleife verwendete Funktional τ ist stetig und damit erhalten wir, nach dem Fixpunktsatz für stetige Funktionen auf Cpos, für seinen kleinsten Fixpunkt μ_τ die Darstellung $\mu_\tau = \bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega]$ mit der Funktion $\Omega(\sigma) = \perp$ als dem kleinsten Element der Funktions-Cpo $(\mathsf{Store}^\perp \to \mathsf{Store}^\perp)$. Durch Induktion über den Aufbau von $s \in \mathcal{PRG}$ kann weiterhin gezeigt werden, daß die Funktion $[\![s]\!] : \mathsf{Store}^\perp \to \mathsf{Store}^\perp$ strikt und damit, wegen der Flachheit der Cpo $\mathsf{Store}^\perp$, auch stetig ist.

Wenn man Vor- und Nachbedingungen von While-Programmen durch Formeln einer Logik beschreibt, so greift man zu Spezifikationszwecken normalerweise auch auf nicht in der Basis-Signatur Σ vorhandene Operationen zurück. Im weiteren setzen wir deshalb eine Erweiterung Σ_e von Σ um zusätzliche Operationen voraus und bezeichnen mit $\mathcal{T}_m$ die Menge der Σ_e-Terme der Sorte m. Die Interpretation der erweiterten Signatur geschieht durch eine Σ_e-Algebra A_e, deren Redukt auf Σ mit der Σ-Algebra A übereinstimmt. Wie schon im Fall der Terme aus $\mathcal{E}_m$, so schreiben wir auch $[\![t]\!](\sigma)$ für den Wert des Terms $t \in \mathcal{T}_m$ bezüglich des Speichers $\sigma \in \mathsf{Store}^\perp$ und nehmen wiederum $[\![t]\!](\perp) = \perp_m$ an.

2.3 Infinitäre Formeln erster Stufe. Wir verwenden, wie schon in der Einleitung erwähnt, R. Backs Ansatz aus [1] zur Formalisierung des *wp*-Kalküls, der auf die infinitäre Sprache $\mathcal{L}_{\omega_1\omega}$ (vergleiche mit [12, 13]) aufbaut. Gegenüber den finitären Σ_e-Formeln $\mathcal{L}$ erster Stufe, mit der Menge $\mathcal{T}_{bool}$ der Booleschen Terme als den atomaren Formeln, treten für die Sprache $\mathcal{L}_{\omega_1\omega}$ die Zeichen $\bigvee$ (für unendliche Disjunktion), $\bigwedge$ (für unendliche Konjunktion) und *def* (für den Definiertheitstest) hinzu. Die syntaxdefinierenden Regeln bestehen aus den Regeln von $\mathcal{L}$, mit $\mathcal{L}_{\omega_1\omega}$ an Stelle von $\mathcal{L}$, und den nachfolgenden drei Regeln:

1. Für alle $t \in \mathcal{T}_m$ gilt $def[t] \in \mathcal{L}_{\omega_1\omega}$.
2. Ist $\Phi \subseteq \mathcal{L}_{\omega_1\omega}$ abzählbar, dann gilt auch $\bigvee \Phi \in \mathcal{L}_{\omega_1\omega}$.
3. Ist $\Phi \subseteq \mathcal{L}_{\omega_1\omega}$ abzählbar, dann gilt auch $\bigwedge \Phi \in \mathcal{L}_{\omega_1\omega}$.

Der Gültigkeitsbegriff für die erweiterte Menge $\mathcal{L}_{\omega_1\omega}$ von Formeln ergibt sich für einen definierten Speicher $\sigma \in \mathsf{Store}$ dadurch, daß man die Klauseln für die finitären Formeln $\mathcal{L}$ erster Stufe (wobei ein Boolescher Term $b \in \mathcal{T}_{bool}$ genau dann gültig ist, wenn sein Wert tt ergibt) wie nachfolgend aufgeführt ergänzt:

1. *Definiertheitstests*:

$$\models def[t][\sigma] \;:\Longleftrightarrow\; [\![t]\!](\sigma) \neq \perp$$

2. *Unendliche Disjunktion*:

$$\models \bigvee \Phi\,[\sigma] \;:\Longleftrightarrow\; \text{Es gibt ein } \varphi \in \Phi \text{ mit } \models \varphi[\sigma]$$

3. *Unendliche Konjunktion*:

$$\models \bigwedge \Phi\,[\sigma] \;:\Longleftrightarrow\; \text{Für alle } \varphi \in \Phi \text{ gilt } \models \varphi[\sigma]$$

Wie üblich schreiben wir $\models \varphi$, falls die Beziehung $\models \varphi[\sigma]$ für alle definierten Speicher $\sigma \in \mathsf{Store}$ zutrifft. ■

Normalerweise ist der Definiertheitstest nicht Teil von infinitären Sprachen. Er wurde hier nur wegen der möglichen Undefiniertheit von Termen mit aufgenommen, da diese bei der beabsichtigten Axiomatisierung des *wp*-Kalküls zu berücksichtigen ist. Man beachte ferner, daß, wegen der Gültigkeitsdefinition für Boolesche Terme, für die Negationsoperation $not \in F$ der Signatur Σ die Beziehungen $\models not(b)[\sigma]$ und $[\![b]\!](\sigma) = \mathit{ff}$ äquivalent sind. Hingegen gilt die Äquivalenz von $\models not(b)[\sigma]$ und $\models \neg b[\sigma]$ nur dann, falls $[\![b]\!](\sigma) \neq \bot_{bool}$.

Aufbauend auf die Semantikdefinition 2.2 und den in 2.3 erklärten Gültigkeitsbegriff für infinitäre Formeln, können wir nun die folgenden wichtigen Begriffe festlegen:

2.4 Totale Korrektheit und schwächste Vorbedingung. a) Ein While-Programm $s \in \mathcal{PRG}$ heißt *total korrekt* bezüglich der Vorbedingung $\psi \in \mathcal{L}_{\omega_1\omega}$ und der Nachbedingung $\varphi \in \mathcal{L}_{\omega_1\omega}$, in Zeichen $[\psi]\,s\,[\varphi]$, falls für alle definierten Speicher $\sigma \in \mathsf{Store}$ gilt

$$\models \psi[\sigma] \implies [\![s]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s]\!](\sigma)]\,.$$

b) Zu einem While-Programm $s \in \mathcal{PRG}$ und einer Nachbedingung $\varphi \in \mathcal{L}_{\omega_1\omega}$ heißt eine Formel $\alpha \in \mathcal{L}_{\omega_1\omega}$ eine *schwächste Vorbedingung* (zur totalen Korrektheit), falls die nachfolgenden Bedingungen gelten:

(i) $[\alpha]\,s\,[\varphi]$ (ii) Für alle $\psi \in \mathcal{L}_{\omega_1\omega}$ gilt $[\psi]\,s\,[\varphi] \implies \models \psi \to \alpha\,.$ ■

In Punkt a) von 2.4 ist auf der Metaebene die Konjunktion natürlich sequentiell gemeint, d.h. sie gilt nicht, falls $[\![s]\!](\sigma) = \bot$ zutrifft, unabhängig von der (in diesem Fall nicht definierten) Beziehung $\models \varphi[\bot]$. Man kann auf die sequentielle Auffassung der Konjunktion verzichten, wenn man, wie etwa in [16], zusätzlich festlegt, daß jede Formel bezüglich des undefinierten Speichers $\bot$ gültig ist.

3 Die *wp*-Funktion als Formeltransformator

Im folgenden bezeichne $var(\varphi) \subseteq X$ die Menge der in der Formel φ vorkommenden Variablen. Bei einer rein syntaktischen Auffassung des *wp*-Kalküls benötigt man, wie schon in der Einleitung erwähnt, bei der Axiomatisierung der Zuweisung $x := t$ die Ersetzung $\varphi[t/x]$ der freien Vorkommen der Variablen $x \in X_m$ in der Formel φ durch den Term $t \in \mathcal{E}_m$. Termersetzung ist auch ein für das praktische Arbeiten mit Formeln unerläßliches Hilfmittel, etwa wenn im Laufe von Formelmanipulationen Variablenumbenennungen notwendig werden.

Bei finitären Formeln erfolgt die formale Definition der Substitution durch Induktion über den Aufbau von φ. Wir geben nachfolgend nur den Fall der Quantifizierung $Q \in \{\forall, \exists\}$ an, wo

man verhindern muß, daß durch die Ersetzung parasitäre Bindungen entstehen:

$$(Q\,y : \varphi)[t/x] = \begin{cases} Q\,y : \varphi & : \; x \text{ kommt nicht frei vor in } Q\,y : \varphi \\ Q\,y : \varphi[t/x] & : \; x \text{ kommt frei vor in } Q\,y : \varphi \text{ und } y \notin var(t) \\ Q\,z : \varphi[z/y][t/x] & : \; \text{sonst} \end{cases}$$

In dieser Festlegung ist z eine „frische“, d.h. nirgends in der gegebenen Formel $Q\,y : \varphi$ oder dem Substitutionsterm t vorkommende, Variable, normalerweise die erste „frische“ in der unterstellten Aufzählung $x_0, x_1, x_2, \ldots$ der Variablen aus X, damit die Substitution eindeutig wird.

Substitution in quantifizierten $\mathcal{L}_{\omega_1\omega}$-Formeln auf die eben gezeigte einfache Weise ist bei der abzählbar-unendlichen Variablenmenge X jedoch problematisch, wenn $var(\varphi) = X$ gilt, wie etwa bei

$$\exists\, x_0 : \bigvee \{x_0 = x_i : i \in \mathbb{N} \setminus \{0\}\}\,.$$

Wir betrachten deshalb zur Axiomatisierung des *wp*-Kalküls die nachfolgend erklärte Teilmenge von $\mathcal{L}_{\omega_1\omega}$-Formeln, bei der die klassische Definition der Substitution anwendbar ist.

3.1 Zusicherungen. Eine Formel $\varphi \in \mathcal{L}_{\omega_1\omega}$ heißt eine *Zusicherung*, falls $var(\varphi)$ endlich ist. Die Menge der Zusicherungen bezeichnen wir mit $\mathcal{ASS}$. ■

Durch Induktion über den Aufbau der Zusicherungen kann man leicht die folgende Substitutionseigenschaft zeigen (man vergleiche beispielsweise mit [7], Substitutionslemma 8.3), die wir später noch benötigen werden.

3.2 Satz (Substitutionseigenschaft). Gegeben seien $\varphi \in \mathcal{ASS}$, $x \in X_m$ und $t \in \mathcal{T}_m$. Dann gilt für jeden definierten Speicher $\sigma \in \mathsf{Store}$ die Äquivalenz

$$\models \varphi[t/x][\sigma] \iff \models \varphi[\sigma[x \leftarrow [\![t]\!](\sigma)]]\,.$$ ■

Nach allen diesen Vorbereitungen, sind wir nun in der Position, die *wp*-Funktion auf Zusicherungen formulieren zu können, d.h. als Formeltransformator. Die nachfolgend angegebene Definition von *wp* entspricht genau der Restriktion der von R. Back in [1] angegebenen *wp*-Funktion von $\mathcal{L}_{\omega_1\omega}$ auf $\mathcal{ASS}$, wobei zusätzlich noch die in [1] verwendeten *wp*-Formeln für die nichtdeterministische bewachte Auswahl und die allgemeine `do-od`-Schleife auf ihre Spezialfälle Alternative und While-Schleife in der von uns verwendeten Programmiersprache angepaßt sind.

3.3 Die *wp*-Funktion. Die Funktion $wp : \mathcal{PRG} \times \mathcal{ASS} \to \mathcal{ASS}$ ist induktiv über den Aufbau des ersten Arguments wie folgt definiert:

1. *Leere Anweisung*:
$$wp(\texttt{skip}, \varphi) = \varphi$$
2. *Undefinierte Anweisung*:
$$wp(\texttt{abort}, \varphi) = \mathit{false}$$
3. *Zuweisung*:
$$wp(x := t, \varphi) = \mathit{def}\,[t] \wedge \varphi[t/x]$$
4. *Alternative*:
$$wp(\texttt{if } b \texttt{ then } s_1 \texttt{ else } s_2 \texttt{ fi}, \varphi) = (b \wedge wp(s_1, \varphi)) \vee (\mathit{not}(b) \wedge wp(s_2, \varphi))$$

5. *While-Schleife*:

$$wp(\texttt{while } b \texttt{ do } s \texttt{ od}, \varphi) = \bigvee \{\varphi_i : i \in \mathbb{N}\}\,,$$

wobei die abzählbar-unendliche Menge $\{\varphi_i : i \in \mathbb{N}\}$ von Formeln induktiv definiert ist durch

$$\varphi_0 = not(b) \wedge \varphi \qquad \varphi_{i+1} = (b \wedge wp(s, \varphi_i)) \vee (not(b) \wedge \varphi_i)\,.$$

6. *Sequentielle Komposition*:

$$wp(s_1; s_2, \varphi) = wp(s_1, wp(s_2, \varphi))$$ ■

Durch den Definiertheitstest $def[t]$ auf der rechten Seite des Zuweisungsfalls 3 wird sichergestellt, daß die Implikation

$$[\![t]\!](\sigma) = \bot_m \implies \not\models wp(x := t, \varphi)[\sigma]$$

zutrifft. In den Fällen 4 und 5 der Alternative und der While-Schleife konnte auf den rechten Seiten jeweils auf die Definiertheitstests $def[b]$ verzichtet werden. Dies liegt daran, daß undefinierte Boolesche Terme nicht gültig sind, denn dies impliziert wiederum, daß für eine undefinierte Bedingung b sowohl die wp-Formel der Alternative als auch die der While-Schleife nicht gültig ist.

Wegen der in 3.1 erhobenen Forderung an die Zusicherungen, daß sie nur endlich viele Variablen enthalten dürfen, haben wir noch den nachfolgenden Satz 3.4 zu beweisen, der die Wohldefiniertheit der wp-Funktion zeigt, d.h., daß diese Funktion tatsächlich Zusicherungen auf Zusicherungen abbildet. In seinem Beweis verwenden wir die übliche Voraussetzung, daß bei einer Formel $wp(s, \varphi)$ die im While-Programm $s \in \mathcal{PRG}$ vorkommenden Variablen in der Zusicherung $\varphi \in \mathcal{ASS}$ nicht durch einen Quantor gebunden werden dürfen. Diese Voraussetzung, *die wir ab jetzt global unterstellen*, erspart die Separierung der sogenannten Programmvariablen aus X von den restlichen (logischen) Variablen, über die natürlich quantifiziert werden darf.

3.4 Satz (Wohldefiniertheit). Für jedes While-Programm $p \in \mathcal{PRG}$ und jede Zusicherung $\varphi \in \mathcal{ASS}$ gilt die Inklusion $var(wp(p, \varphi)) \subseteq var(p) \cup var(\varphi)$. Insbesondere enthält $wp(p, \varphi)$ nur endlich viele Variablen.

Beweis. Wir führen eine Induktion über den Aufbau von p.

Der Induktionsbeginn besteht aus drei Fällen. Ist p die leere Anweisung `skip`, so gilt die Gleichung

$$var(wp(\texttt{skip}, \varphi)) = var(\varphi)\,,$$

was alles zeigt. Falls p die undefinierte Anweisung `abort` ist, so folgt die Behauptung aus

$$var(wp(\texttt{abort}, \varphi)) = var(false) = \emptyset\,.$$

Den Fall, daß p eine Zuweisung $x := t$ ist, zeigt man schließlich durch

$$var(wp(x := t, \varphi)) = var(def[t]) \cup var(\varphi[t/x]) \subseteq var(t) \cup var(\varphi)\,,$$

da aufgrund der oben gemachten globalen Voraussetzung die Variable x des Programms nur frei in der Formel φ vorkommen und die gebundene Variable y nicht im Term t vorkommen kann, die Substitution also (nach der klassischen Definition, übertragen auf die Zusicherungen) ohne frische Variable auskommt.

Beim Induktionschluß sind die Fälle der Alternative und sequentielle Komposition so einfach, daß wir p gleich als While-Schleife `while` b `do` s `od` voraussetzen. Wir zeigen in einer Nebeninduktion für alle natürlichen Zahlen $i \in \mathbb{N}$ und alle Formeln φ_i aus 3.3 die Inklusion

$$var(\varphi_i) \subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi)\,, \tag{4}$$

denn diese impliziert

$$var(wp(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}, \varphi)) = \bigcup_{i\in\mathbb{N}} var(\varphi_i) \subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi)\,.$$

Der Induktionsbeginn $i = 0$ folgt aus

$$var(\varphi_0) = var(not(b) \wedge \varphi) \subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi)\,.$$

Den Induktionsschritt $i \mapsto i+1$ zeigt man wie nachstehend:

$$\begin{aligned} var(\varphi_{i+1}) &= var(b) \cup var(wp(s, \varphi_i)) \cup var(\varphi_i) \\ &\subseteq var(b) \cup var(s) \cup var(\varphi_i) && \text{Ind. Hyp. 3.4} \\ &\subseteq var(\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}) \cup var(\varphi) && \text{Ind. Hyp. (4)} \end{aligned}$$ ■

4 *wp*-Funktion und denotationelle Semantik

Wegen der Eindeutigkeit der Substitution wird durch die induktive Definition von $wp(s, \varphi)$ in 3.3 einem While–Programm $s \in \mathcal{PRG}$ und einer Nachbedingung $\varphi \in \mathcal{ASS}$ genau eine Zusicherung aus $\mathcal{ASS}$ zugeordnet. Für diese gilt die beabsichtigte Eigenschaft; dies ist das Hauptresultat dieses Abschnitts. Wir beginnen mit einer Aussage, welche die Gültigkeit einer *wp*-Zusicherung bezüglich eines definierten Speichers beschreibt.

4.1 Lemma. Für alle While-Programme $p \in \mathcal{PRG}$, Zusicherungen $\varphi \in \mathcal{ASS}$ und definierte Speicher $\sigma \in \mathsf{Store}$ gilt die Äquivalenz

$$\models wp(p, \varphi)[\sigma] \iff [\![p]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![p]\!](\sigma)]\,.$$

Beweis. Wir führen wiederum eine Induktion über den Aufbau von p.

Der Induktionsbeginn besteht aus drei Fällen: Ist p die leere Anweisung `skip`, so bekommen wir die Äquivalenz

$$\begin{aligned} & \models wp(\texttt{skip}, \varphi)[\sigma] \\ \iff & \models \varphi[\sigma] && \text{Def. } wp \\ \iff & \sigma \neq \bot \text{ und } \models \varphi[\sigma] && \text{da } \sigma \in \mathsf{Store} \\ \iff & [\![\texttt{skip}]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![\texttt{skip}]\!](\sigma)] && \text{Def. Semantik.} \end{aligned}$$

Falls p die undefinierte Anweisung `abort` ist, so sind die beiden Seiten der Behauptung wegen $\not\models false[\sigma]$ und $[\![\texttt{abort}]\!](\sigma) = \bot$ falsch, also gilt die Behauptung. Wenn p die Form einer Zuweisung $x := t$ hat, so erhalten wir schließlich

$$\begin{aligned} & \models wp(x := t, \varphi)[\sigma] \\ \iff & \models def[t][\sigma] \text{ und } \models \varphi[t/x][\sigma] && \text{Def. } wp \\ \iff & [\![t]\!](\sigma) \neq \bot \text{ und } \models \varphi[\sigma[x \leftarrow [\![t]\!](\sigma)]] && \text{Gültigkeit } def\text{, Satz 3.2} \\ \iff & [\![x := t]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![x := t]\!](\sigma)] && \text{Def. Semantik.} \end{aligned}$$

Beim Induktionsschluß nehmen wir p zuerst als Alternative $\texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2\ \texttt{fi}$ an und unterscheiden drei Fälle: Es gelte $[\![b]\!](\sigma) = tt$. Dann bekommen wir, indem wir die Äquivalenz dieser Gleichung mit $\models b[\sigma]$ verwenden,

$$\begin{array}{lll} & \models wp(\texttt{if}\ldots\texttt{fi}, \varphi)[\sigma] & \\ \iff & \models wp(s_1, \varphi)[\sigma] & \text{Def. } wp \\ \iff & [\![s_1]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s_1]\!](\sigma)] & \text{Ind. Hyp. 4.1} \\ \iff & [\![\texttt{if}\ldots\texttt{fi}]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![\texttt{if}\ldots\texttt{fi}]\!](\sigma)] & \text{Def. Semantik.} \end{array}$$

Gilt hingegen $[\![b]\!](\sigma) = ff$, so ist dies gleichwertig zu $\models not(b)[\sigma]$ und daraus folgt, analog zu eben, die Äquivalenz

$$\begin{array}{lll} & \models wp(\texttt{if}\ldots\texttt{fi}, \varphi)[\sigma] & \\ \iff & \models wp(s_2, \varphi)[\sigma] & \text{Def. } wp \\ \iff & [\![s_2]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s_2]\!](\sigma)] & \text{Ind. Hyp. 4.1} \\ \iff & [\![\texttt{if}\ldots\texttt{fi}]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![\texttt{if}\ldots\texttt{fi}]\!](\sigma)] & \text{Def. Semantik.} \end{array}$$

Der dritte Fall ist schließlich noch $[\![b]\!](\sigma) = \bot_{bool}$. Auch hier gilt die Behauptung weil, wegen $\not\models b[\sigma]$ und $\not\models not(b)[\sigma]$ bzw. $[\![\texttt{if}\ldots\texttt{fi}]\!](\sigma) = \bot$, ihre beiden Seiten falsch sind.

Nun setzen wir voraus, daß p eine While-Schleife $\texttt{while}\ b\ \texttt{do}\ s\ \texttt{od}$ sei. Wir zeigen für diesen Fall zuerst durch eine Nebeninduktion (analog zu Satz 3.4) für alle definierten Speicher $\rho \in \mathsf{Store}$ und alle natürlichen Zahlen $i \in \mathbb{N}$ die Äquivalenz

$$\models \varphi_i[\rho] \iff \tau^{i+1}[\Omega](\rho) \neq \bot \text{ und } \models \varphi[\tau^{i+1}[\Omega](\rho)], \tag{5}$$

mit den Zusicherungen φ_i aus der Definition 3.3 der wp-Funktion und dem Funktional τ aus der Semantikdefinition 2.2 der While-Schleife.

Beim Induktionsbeginn $i = 0$ reduziert sich die Behauptung (5) auf

$$\models not(b)[\rho] \text{ und } \models \varphi[\rho] \iff \tau[\Omega](\rho) \neq \bot \text{ und } \models \varphi[\tau[\Omega](\rho)]\,.$$

Diese Äquivalenz gilt. Aus $[\![b]\!](\rho) = tt$ oder $[\![b]\!](\rho) = \bot_{bool}$ folgt nämlich, daß ihre beiden Seiten falsch sind, und im verbleibenden Fall $[\![b]\!](\rho) = ff$ sind beide Seiten offensichtlich äquivalent zur Beziehung $\models \varphi[\rho]$.

Beim Induktionsschluß $i \mapsto i + 1$ unterscheiden wir ebenfalls die drei Fälle des Induktionsbeginns. Zuerst sei $[\![b]\!](\rho) = tt$. Dann bekommen wir, indem wir – wie bei der Alternative – die Äquivalenz dieser Gleichung mit $\models b[\sigma]$ verwenden,

$$\begin{array}{lll} & \models \varphi_{i+1}[\rho] & \\ \iff & \models wp(s, \varphi_i)[\rho] & \text{Def. } wp \\ \iff & [\![s]\!](\rho) \neq \bot \text{ und } \models \varphi_i[[\![s]\!](\rho)] & \text{Ind. Hyp. 4.1} \\ \iff & [\![s]\!](\rho) \neq \bot \text{ und } \tau^{i+1}[\Omega]([\![s]\!](\rho)) \neq \bot \text{ und } \models \varphi[\tau^{i+1}[\Omega]([\![s]\!](\rho))] & \text{Ind. Hyp. (5)} \\ \iff & \tau^{i+1}[\Omega]([\![s]\!](\rho)) \neq \bot \text{ und } \models \varphi[\tau^{i+1}[\Omega]([\![s]\!](\rho))] & \tau[h] \text{ strikt} \\ \iff & \tau^{i+2}[\Omega](\rho) \neq \bot \text{ und } \models \varphi[\tau^{i+2}[\Omega](\rho)] & \text{Def. } \tau. \end{array}$$

Im zweiten Fall $[\![b]\!](\rho) = ff$ erhalten wir

$$\begin{array}{lll} & \models \varphi_{i+1}[\rho] & \\ \iff & \models \varphi_i[\rho] & \text{Def. } wp \\ \iff & \tau^{i+1}[\Omega](\rho) \neq \bot \text{ und } \models \varphi[\tau^{i+1}[\Omega](\rho)] & \text{Ind. Hyp. (5)} \\ \iff & \tau^{i+2}[\Omega](\rho) \neq \bot \text{ und } \models \varphi[\tau^{i+2}[\Omega](\rho)] & \tau^{i+2}[\Omega](\rho) = \rho = \tau^{i+1}[\Omega](\rho) \end{array}$$

wegen der Äquivalenz der vorausgesetzten Gleichung zur Beziehung $\models not(b)[\sigma]$. Gilt schließlich $[\![b]\!](\rho) = \bot_{bool}$, so sind wiederum die beiden Seiten der zu zeigenden Äquivalenz (5) nach der Definition der *wp*-Funktion und des Schleifenfunktionals τ falsch. Damit ist der Beweis von (5) beendet.

Nun betrachten wir das Supremum $\bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega](\sigma)$ der Funktionsiteration des Fixpunktsatzes für stetige Funktionen auf Cpos. Ist es ungleich $\bot$, so gibt es offensichtlich ein Kettenglied mit $\tau^{i_0}[\Omega](\sigma) \neq \bot$. Da die Cpo $\mathsf{Store}^\bot$ flach geordnet ist, gilt sogar $\bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega](\sigma) = \tau^{i_0}[\Omega](\sigma) \neq \bot$. Wegen der Flachheit von $\mathsf{Store}^\bot$ trifft auch die Umkehrung dieser Aussage zu: Gibt es also ein $i_0 \in \mathbb{N}$ mit $\tau^{i_0}[\Omega](\sigma) \neq \bot$, so stimmt dieses Glied der Kette mit dem Supremum $\bigsqcup_{i\in\mathbb{N}} \tau^i[\Omega](\sigma)$ überein[1].

Mit Hilfe der eben bewiesenen Äquivalenz sind wir nun in der Lage, den Fall der While-Schleife abzuschließen, wobei diese Eigenschaft bei der nachfolgenden Kette von Äquivalenzumformung im vierten Schritt Verwendung findet:

$$
\begin{array}{lll}
& \models wp(\texttt{while}\ldots\texttt{od}, \varphi)[\sigma] & \\
\Longleftrightarrow & \models (\bigvee\{\varphi_i : i \in \mathbb{N}\})[\sigma] & \text{Def. } wp \\
\Longleftrightarrow & \text{Es gibt ein } i \in \mathbb{N} \text{ mit } \models \varphi_i[\sigma] & \\
\Longleftrightarrow & \text{Es gibt ein } i \in \mathbb{N} \text{ mit } \tau^{i+1}[\Omega](\sigma) \neq \bot \text{ und } \models \varphi[\tau^{i+1}[\Omega](\sigma)] & (5) \\
\Longleftrightarrow & \bigsqcup_{i\in\mathbb{N}} \tau^{i+1}[\Omega](\sigma) \neq \bot \text{ und } \models \varphi[\bigsqcup_{i\in\mathbb{N}} \tau^{i+1}[\Omega](\sigma)] & \\
\Longleftrightarrow & \bigsqcup_{i\in\mathbb{N}} \tau^{i}[\Omega](\sigma) \neq \bot \text{ und } \models \varphi[\bigsqcup_{i\in\mathbb{N}} \tau^{i}[\Omega](\sigma)] & \text{Indextransform.} \\
\Longleftrightarrow & \mu_\tau(\sigma) \neq \bot \text{ und } \models \varphi[\mu_\tau(\sigma)] & \tau \text{ stetig} \\
\Longleftrightarrow & [\![\texttt{while}\ldots\texttt{od}]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![\texttt{while}\ldots\texttt{od}]\!](\sigma)] & \text{Def. Semantik}
\end{array}
$$

Die Indextransformation im fünften Schritt ist korrekt, da $\tau^0[\Omega](\sigma) = \bot$ das kleinste Element der Cpo $\mathsf{Store}^\bot$ ist.

Es bleibt noch der Fall, daß p eine sequentiellen Komposition $s_1; s_2$ ist. Hier erhalten wir die zu zeigende Äquivalenz durch

$$
\begin{array}{lll}
& \models wp(s_1; s_2, \varphi)[\sigma] & \\
\Longleftrightarrow & \models wp(s_1, wp(s_2, \varphi))[\sigma] & \text{Def. } wp \\
\Longleftrightarrow & [\![s_1]\!](\sigma) \neq \bot \text{ und } \models wp(s_2, \varphi)[[\![s_1]\!](\sigma)] & \text{Ind. Hyp. 4.1} \\
\Longleftrightarrow & [\![s_1]\!](\sigma) \neq \bot \text{ und } [\![s_2]\!]([\![s_1]\!](\sigma)) \neq \bot \text{ und } \models \varphi[[\![s_2]\!]([\![s_1]\!](\sigma))] & \text{Ind. Hyp. 4.1} \\
\Longleftrightarrow & [\![s_2]\!]([\![s_1]\!](\sigma)) \neq \bot \text{ und } \models \varphi[[\![s_2]\!]([\![s_1]\!](\sigma))] & \text{Semantik strikt} \\
\Longleftrightarrow & [\![s_1; s_2]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s_1; s_2]\!](\sigma)] & \text{Def. Semantik.} \quad \blacksquare
\end{array}
$$

Insbesondere erhalten wir mit Hilfe dieses Lemmas: Ist $\sigma \in \mathsf{Store}$ ein definierter Speicher, so ist $\models wp(s, \mathit{true})[\sigma]$ äquivalent zur Terminierungsbedingung $[\![s]\!](\sigma) \neq \bot$.

Wir kommen nun zum am Anfang dieses Abschnitts angekündigten Hauptresultat, dem Korrektheitssatz, welcher besagt, daß $wp(s, \varphi)$ tatsächlich eine schwächste Vorbedingung für totale Korrektheit ist. Ein Beweis dieser Aussage ist nicht mehr allzu schwierig, da der technische Teil schon im vorhergehenden Lemma 4.1 bewerkstelligt wurde.

4.2 Satz (Korrektheit). Die Zusicherung $wp(s, \varphi) \in \mathcal{ASS}$ ist eine schwächste Vorbedingung für totale Korrektheit zum Programm $s \in \mathcal{PRG}$ und zur Nachbedingung $\varphi \in \mathcal{ASS}$.

[1] Die Eigenschaft, daß in der Kette $\Omega(\sigma) \leq \tau[\Omega](\sigma) \leq \tau^2[\Omega](\sigma) \leq \ldots$ höchstens ein Wertwechsel auftritt, gilt auch, wenn $\mathsf{Store}^\bot$ nicht flach geordnet ist. Durch Induktion [8] kann nämlich für alle $i \in \mathbb{N}$ und $\sigma \in \mathsf{Store}^\bot$ leicht gezeigt werden, daß $\tau^i[\Omega](\sigma) = \bot$ oder $\tau^i[\Omega](\sigma) = \tau^{i+1}[\Omega](\sigma)$.

Beweis. Mit Hilfe von Lemma 4.1, Richtung „⇒“, erhalten wir für jeden definierten Speicher $\sigma \in \mathsf{Store}$ die Implikation

$$\models wp(s,\varphi)[\sigma] \implies [\![s]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s]\!](\sigma)]\,.$$

Diese zeigt $[wp(s,\varphi)]\,s\,[\varphi]$, also Bedingung (i) von 2.4.b. Nun sei $\psi \in \mathcal{ASS}$ eine weitere Zusicherung mit $[\psi]\,s\,[\varphi]$. Dann gilt für jeden definierten Speicher $\sigma \in \mathsf{Store}$ die Implikation

$$\begin{array}{ll} & \models \psi[\sigma] \\ \implies & [\![s]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s]\!](\sigma)] \qquad \text{wegen } [\psi]\,s\,[\varphi] \\ \implies & \models wp(s,\varphi)[\sigma] \qquad \text{Lemma 4.1, Richtung „⇐“.} \end{array}$$

Somit haben wir die Beziehung $\models \psi \to wp(s,\varphi)$ gezeigt, also genau die noch fehlende Eigenschaft (ii) von 2.4.b. ■

Dieser Satz besagt, daß die Menge $\mathcal{L}_{\omega_1\omega}$ zur Axiomatisierung von schwächsten Vorbedingungen geeignet ist. Statt $\mathcal{L}_{\omega_1\omega}$ kann man auch andere Erweiterungen der Logik erster Stufe verwenden, wie dies beispielsweise in [15, 3, 4] mittels einer schwachen Logik zweiter Stufe oder in [2] mittels einer polymorphen Logik höherer Stufe geschieht.

Die finitären Formeln $\mathcal{L}$ erster Stufe sind jedoch zu schwach zu einer solchen Axiomatisierung; es gibt Formeln $\varphi \in \mathcal{L}$ und While-Programme $s \in \mathcal{PRG}$, so daß keine Formel $\psi \in \mathcal{L}$ eine schwächste Vorbedingung zu s und der Nachbedingung φ im Sinne von 2.4.b sein kann. Dies wird im nächsten Beispiel gezeigt.

4.3 Beispiel (nach R. Back, [1]). Wir betrachten die finitäre Formel $\mathit{true} \in \mathcal{L}$ und das While-Programm

$$\begin{array}{l} y := x; \\ \texttt{while } y \neq e \texttt{ do} \\ \quad y \;:=\; y \circ x \texttt{ od} \end{array}$$

mit Variablen x, y und einer Konstanten e, alle von gleicher Sorte $g \in S$, sowie einer entsprechend typisierten Infix-Operationen $\circ$. Weiterhin nehmen wir an, daß das Tripel $(g^A, \circ^A, e^A)$ in der Σ-Algebra A eine Gruppe $(G, \cdot, 1)$ bildet und die Ungleichheitsoperation $\neq$ bzw. die Gleichheitsoperation $=$ der Signatur strikt standardinterpretiert sind.

Es bezeichne s das obige While-Programm. Aus der Definition der wp-Funktion in 3.3 folgt dann unmittelbar die Gleichheit

$$wp(s,\mathit{true}) = \mathit{def}[x] \wedge (\bigvee\{\varphi_i : i \in \mathbb{N}\})[x/y]\,.$$

Durch einige elementare Umformungen[2] kann man für alle Zusicherungen φ_i, $i \in \mathbb{N}$, aus dem Schleifenfall von 3.3 und alle definierten Speicher $\sigma \in \mathsf{Store}$ die Äquivalenz

$$\models \varphi_i[\sigma] \iff \models (y = e \vee y \circ x = e \vee \ldots \vee y \circ x \circ \ldots \circ x = e)[\sigma]$$

beweisen, mit $i+1$ Disjunktionsgliedern in der Zusicherung der rechten Seite, wobei jeweils die Variable x im j-ten Glied genau $j-1$-mal vorkommt $(1 \leq j \leq i+1)$. Daraus folgt unmittelbar für $\sigma \in \mathsf{Store}$ die Äquivalenz

$$\begin{array}{ll} & \models wp(s,\mathit{true})[\sigma] \\ \iff & \models \mathit{def}[x][\sigma] \text{ und } \models (\bigvee\{x = e, x = e \vee x \circ x = e, \ldots\})[\sigma] \\ \iff & \sigma(x) \neq \bot_g \text{ und es gibt ein } i \in \mathbb{N} \setminus \{0\} \text{ mit } \sigma(x)^i = 1 \\ \iff & \sigma(x) \text{ ist ein Torsionselement in } (G, \cdot, 1). \end{array}$$

[2] Insbesondere hat man zu verwenden, daß aus der Gültigkeit $\models (t_1 = t_2)[\sigma]$ sowohl $\models \mathit{def}[t_1][\sigma]$ als auch $\models \mathit{def}[t_2][\sigma]$ folgt.

Nun nehmen wir an, daß $\psi \in \mathcal{L}$ eine schwächste Vorbedingung zu s und der Nachbedingung *true* sei. Da offensichtlich alle schwächsten Vorbedingungen äquivalent sind, gilt nach der obigen Rechnung für $\sigma \in \mathsf{Store}$, daß

$$\models \psi[\sigma] \iff \sigma(x) \text{ ist ein Torsionselement in } (G, \cdot, 1)\,.$$

Ist also $\mathcal{G} \subseteq \mathcal{L}$ die Menge der Gruppenaxiome, so axiomatisiert die Menge $\mathcal{G} \cup \{\forall\, x : \psi\}$ von finitären Formeln erster Stufe gerade alle Torsionsgruppen. Aus der Prädikatenlogik ist aber als eine Konsequenz des Kompaktheitssatzes bekannt (siehe beispielsweise [7]), daß dies nicht möglich ist. ■

In der Einleitung haben wir erwähnt, daß aus Lemma 4.1 und dem Korrektheitssatz 4.2 sofort die Äquivalenz von $\models \psi \to wp(s, \varphi)$ und $[\psi]\, s\, [\varphi]$ folgt, die, in Verbindung mit weiteren beweisbaren Eigenschaften, wie den „Healthiness conditions“ und dem „Loop invariant theorem“, üblicherweise verwendet wird, um totale Korrektheit zu verifizieren. Wir beenden diesen Abschnitt mit dem Beweis dieser Äquivalenz.

4.4 Satz. Für alle While-Programme $s \in \mathcal{PRG}$ und alle Zusicherungen $\psi, \varphi \in \mathcal{ASS}$ gilt die Äquivalenz

$$\models \psi \to wp(s, \varphi) \iff [\psi]\, s\, [\varphi]\,.$$

Beweis. Zum Beweis der Richtung „$\Rightarrow$“ sei $\sigma \in \mathsf{Store}$ ein definierter Speicher. Dann haben wir die Implikation

$$\begin{array}{lll} & \models \psi[\sigma] & \\ \Longrightarrow & \models wp(s, \varphi)[\sigma] & \text{wegen } \models \psi \to wp(s, \varphi) \\ \Longrightarrow & [\![s]\!](\sigma) \neq \bot \text{ und } \models \varphi[[\![s]\!](\sigma)] & \text{Lemma 4.1, Richtung „}\Rightarrow\text{“,} \end{array}$$

also die Korrektheitsaussage $[\psi]\, s\, [\varphi]$. Bei der umgekehrten Richtung „$\Leftarrow$“ verwenden wir, daß $wp(s, \varphi)$ nach dem Korrektheitssatzes 4.2 eine schwächste Vorbedingung zu s und der Nachbedingung φ ist. Aus der Forderung (ii) von 2.4.b folgt dann $\models \psi \to wp(s, \varphi)$. ■

5 Abschließende Bemerkungen

Aufbauend auf den Ansatz von R. Back haben wir in diesem Artikel eine rein syntaktische Formulierung des *wp*-Kalküls angegeben und die Korrektheit der entsprechenden *wp*-Funktion bezüglich des mittels denotationeller Semantik definierten Begriffs „schwächste Vorbedingung für totale Korrektheit“ bewiesen. Diese dadurch formal hergestellte Verbindung zwischen axiomatischer und denotationeller Semantik ermöglicht es, die beiden Begriffe wechselseitig zu verwenden. Insbesondere kann man bei einer vorliegenden Problemstellung die jeweils geeignetere Semantik zur Lösung wählen.

Ein Beipiel hierzu ist der Nachweis von Terminierung, wie er üblicherweise bei *wp*-Berechnungen vorkommt, d.h. der Beweis von

$$\models (\psi \to wp(\texttt{while b do s od}, \mathit{true}))[\sigma]\,. \tag{6}$$

Unserer Erfahrung nach ist (6) *formal* am einfachsten zu beweisen, indem man auf der Menge $\{\sigma \in \mathsf{Store} : \models \psi[\sigma]\}$ eine geeignete noethersche Relation definiert und $\mu_\tau(\sigma) \neq \bot$ (mit τ als dem Funktional der Schleifensemantik in 2.2) durch noethersche Induktion unter Verwendung von denotationeller Semantik zeigt. Diese Vorgehensweise orientiert sich natürlich am bekannten Terminierungsverfahren (siehe etwa [9]) mit einer sogenannten „bound function“, vermeidet jedoch

dessen Nachteil, daß die noethersche Relation in der Programmiersprache ausgedrückt werden muß. Weiterhin berücksichtigt es auch die sogenannten „endlichen“ Fehler bei der Ausführung der Schleifenbedingung und des Schleifenrumpfes.

Der Einfachheit halber beschränkten wir uns in diesem Artikel auf die deterministische Sprache der While-Programme. Ursprünglich wurde der *wp*-Kalkül von E.W. Dijkstra jedoch für die nichtdeterministische Programmiersprache der „guarded commands“ formuliert; siehe [5, 6]. Es bietet sich deshalb an, die Untersuchungen auch auf diese Programmiersprache auszudehnen und dabei, neben der von E.W. Dijkstra betrachteten dämonischen Version des Nichtdeterminismus, auch die erratische Variante miteinzubeziehen. Im Fall eines beschränkten Nichtdeterminismus dürften bei diesen Verallgemeinerungen eigentlich keine größeren technischen Probleme auftreten, wenn man sich am deterministischen Fall, d.h. insbesondere am Beweis des Lemmas 4.1, orientiert. Läßt man hingegen auch unbeschränkten Nichtdeterminismus zu, wie etwa in [1] mittels einer speziellen Zuweisung, so scheint eine Verallgemeinerung nicht mehr so einfach zu sein. Einerseits verliert man nämlich die Stetigkeit des Funktionals der While-Schleife und damit die Möglichkeit, wie im Beweis von Lemma 4.1 mittels der Äquivalenz (5) vorzugehen, andererseits wird in [1] demonstriert, daß für eine solche Sprache selbst $\mathcal{L}_{\omega_1\omega}$ nicht ausreicht, die *wp*-Funktion zu axiomatisieren.

Danksagung. Ich bedanke mich bei meinen beiden Kieler Kollegen Kai Engelhardt und Martin Fränzle und bei Markus Müller-Olm aus Passau für die Anregungen und Diskussionen während der Abfassung dieses Artikels und auch dafür, daß sie eine Vorfassung kritisch durchgesehen haben.

Literatur

[1] Back, R.-J.: Proving total correctness of nondeterministic programs in infinitary logic. Acta Informatica 15, 233-249 (1981)

[2] Back, R.-J., von Wright J.: Predicate transformers and higher order logic. In: de Bakker J.W., de Roever W.-P., Rozenberg G. (eds.): Proc. REX Workshop „Semantics: Foundations and Applications“, Beekbergen, Niederlande, 1.6.-4.6. 1992, Lecture Notes in Computer Science 666, Springer-Verlag, 1-20 (1993)

[3] Berghammer R., Elbl B., Schmerl U.: Proving correctness of programs in weak second-order logic. In: de Bakker J.W., de Roever W.-P., Rozenberg G. (eds.): Proc. REX Workshop „Semantics: Foundations and Applications“, Beekbergen, Niederlande, 1.6.-4.6. 1992, Lecture Notes in Computer Science 666, Springer-Verlag, 51-72 (1993)

[4] Berghammer R., Elbl B., Schmerl U.: Formalizing Dijkstra's predicate transformer *wp* in weak second-order logic. Theoretical Computer Science 146, 185-197 (1995)

[5] Dijkstra E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Comm. ACM 18 (8), 453-457 (1975)

[6] Dijkstra E.W.: A discipline of programming. Prentice Hall (1976)

[7] Ebbinghaus H.-D., Flum J., Thomas W.: Einführung in die mathematische Logik. BI Wissenschaftsverlag (1992)

[8] Fränzle M.: Private Mitteilung

[9] Gries D.: The science of computer programming. Springer-Verlag (1981)

[10] Harel D.: On folk theorems. Comm. ACM 23 (7), 379-389 (1980)

[11] Hoare C.A.R.: An axiomatic basis for computer programming. Comm. ACM 12 (10), 576-580, 583 (1969)

[12] Karp C.R.: Languages with expressions of infinite length. North-Holland (1964)

[13] Keisler H.J.: Model theory of infinitary logic. North-Holland (1971)

[14] Nielson H.R., Nielson F.: Semantics with applications. A formal introduction. Wiley (1992)

[15] Tucker J.V, Zucker J.L.: Program correctness over abstract data types with error-state semantics. CWI Monographs 6, North-Holland (1988)

[16] Winskel G.: The formal semantics of programming languages. An introduction. Reihe „Foundations of Computing“, The MIT Press (1993)

[17] Wirsing M.: Algebraic Specification. In: van Leeuwen J. (ed.): Handbook of Theoretical Computer Science, Vol. B, Elsevier Science Publishers, 675-788 (1990)