Grundlagen der Programmverifikation

Vorlesung an der Fakultät für Mathematik und Informatik der Universität Leipzig [1]

Dr. Rolf Hartwig

mehrfach gehalten seit Wintersemester 1991/92, letzte Fassung vom Sommersemester 1998

[1] nach einer vom Autor korrigierten Mitschrift von Informatikstudenten

Quelle: http://www.informatik.uni-leipzig.de/~rhartwig/

Der Autor ist jederzeit für inhaltliche Hinweise zur Verbesserung der Vorlesung als auch für Hinweise zu immer noch verbliebenen Druckfehlern dankbar.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In der Praxis bedeutet Programmieren immer auch Fehler zu machen. Das „Fehlerfreimachen“ eines Programms, als „Einfahren“„Testphase“ oder „Debugging“ bekannt, erfordert zumeist wesentlich mehr Zeit als das eigentliche Erstellen des Programms. International spricht man von der Hälfte bis zwei Drittel der Kosten und der Zeit, die – bezogen auf das gesamte Projekt – hierauf entfallen.

Nun ist der gegenwärtige Zustand der, daß ein Programm üblicherweise getestet wird durch eine gewisse Anzahl von *Beispielläufen*, von denen man hofft, daß sie *alle* möglichen Fälle überdecken und alle eventuellen Fehler aufdecken. *Sicher* kann man bei dieser Verfahrensweise danach allerdings *nicht* sein, daß das Programm tatsächlich immer das leistet, was es soll, daß es also *korrekt* ist! Eher im Gegenteil! Bei einem nur einigermaßen anspruchsvollen Programm kann man sicher sein, daß es auch nach einer solchen Testphase noch Fehler enthält. Nicht umsonst spielt in den sogenannten „Software–Lebenszyklen“ die immer wiederkehrende Rückkehr zur Testphase eine entscheidende Rolle.

Man beachte in diesem Zusammenhang auch, daß Fehler in manchen Programmen verheerende Wirkungen haben können (von eingestürzten Brücken über Absturz von Flugzeugen bis zur Auslösung eines Krieges ist vieles denkbar). Vor einigen Jahren führte ein Softwarefehler zum unkorrigierbaren Abbruch der Funkverbindung mit der Weltraumsonde Phobos I, die die entfernteren Planeten unseres Sonnensystems erkunden sollte — ein Milliardenverlust!

Diese Situation ist ein Widerspruch zu der Tatsache, daß das Computerprogrammieren eigentlich als eine „exakte Wissenschaft“ angesehen werden muß. Dabei beziehe ich mich auf die etwa zu Beginn unseres Jahrhunderts übliche Einteilung der Wissenschaften in exakte, empirische und Geisteswissenschaften, wobei zu den exakten Wissenschaften traditionell die Mathematik und die Physik zu zählen sind. Die Computerprogrammierung ist nun *exakt* in dem Sinne, daß alle Eigenschaften eines Programms und alle Konsequenzen seiner Ausführung in einer gewissen Umgebung (Hardware, Compiler usw.) prinzipiell *rein deduktiv* aus dem Programm*text* abgeleitet werden können!

Das legt es nahe, ähnlich wie z.B. in der Geometrie und in anderen Teilgebieten der Mathematik die *axiomatische Methode* auf die „Theorie der Programme“ anzuwenden! Das heißt, *Axiome* und *Schlußregeln* anzugeben, die es gestatten, interessierende Programmeigenschaften zu *beweisen*.

Der Nutzer eines Programms ist in der Regel nicht interessiert an Implementationseinzelheiten, oft auch nicht an den verwendeten mathematischen Verfahren, sondern oft nur an der *Wirkung* des Programms (“wenn ein lösbares lineares Gleichungssystem eingegeben wird, kommt die Lösung heraus“; oder: „sortiert die eingegebenen Elemente“). Deshalb ist eine *Sprache* nützlich, in der man folgende oder ähnliche *Aussagen* formulieren kann:

> Wenn die Eingangsgrößen des Programms S die Eigenschaft p haben, so haben die Ausgabegrößen nach Beendigung der Ausführung von S die Eigenschaft q.

Die Formalisierung einer solchen Sprache erlaubt zugleich (mindestens) zwei Ziele zu verfolgen:

- Programmdokumentation
- Programmverifikation

Zu letzterem ist eine formalisierte Theorie für solche Aussagen wie oben zu schaffen. Dann hat man mit der *Menge aller Theoreme* einer solchen gegebenen Theorie eine sehr allgemeine (völlig maschinenunabhängige) Fassung der *Semantik* dieses Programms, genannt *axiomatische Semantik*. Solche formalisierten Theorien erlauben es prinzipiell, den Ableitungs– bzw. Beweisprozeß wiederum selbst einem Computer zu überlassen bzw. sich durch ihn unterstützen zu lassen. So gibt es inzwischen – wie die sog. Theorembeweiser für Prädikatenkalküle - international auch Programmsysteme zur *automatischen Programmverifikation* (meist allerdings noch auf Experimentalebene).

Die Forderung des **Beweisens** der Korrektheit von Programmen (ob „von Hand“ oder automatisch), **nicht** des **Testens**, beeinflußt naturgemäß auch die

- Programmiermethodologie,

d.h. die Methode des Programmentwurfs (Entwicklung „spezifizierter Programme“) und der Programmdokumentation. Entsprechende Bemerkungen dazu dann im Verlauf dieser Vorlesung.

Hier noch einige Bemerkungen zur **Geschichte**:

Die Einführung der axiomatischen Methode in die Theorie der Programmierung stammt von C.A.R. HOARE mit einer grundlegenden Arbeit von 1969, in der er sich auch auf gewisse Vorüberlegeungen von R.W. FLOYD (1967) beruft. Danach entstand eine Flut von Arbeiten zur

- Entwicklung konkreter Beweissysteme für die verschiedensten Programmkonstruktionen
- Entwicklung der Theorie der Korrektheit von Programmen
- Entwicklungder axiomatischen Semantik
- Entwicklung von automatischen Verifikationssystemen

Natürlicherweise werfen die aufgestellten Beweissysteme eine Fülle *rein logischer Fragen* auf, der sich die Mathematiker und Logiker annahmen:

es entstanden dadurch eine ganze Reihe von Arbeiten zu „Programmlogiken“, „Dynamischen Logiken“, „Algorithmischen Logiken“ und metamathematischen Grundlagenuntersuchungen, die inzwischen eigenständige Bedeutung im Grenzgebiet zwischen Logik, Mathematik und Informatik haben.

Erstmals auf eine reale Programmiersprache angewendet wurde das axiomatische Herangehen auf PASCAL durch HOARE selbst und N. WIRTH 1973, inzwischen auch auf weitere, z.B. auf EUCLID 1978. Gegenwärtig geht es um die Einbeziehung der Parallelarbeit, Nichtdeterminismus, um adäquatere Beweismethoden für rekursive Prozeduren, um nur einiges zu nennen. In dieser Vorlesung allerdings werden nur die Anfänge bzw. *Grundlagen* behandelt, dies aber so, daß der Anschluß an die gegenwärtige Forschung hergestellt und die selbständige Auswertung der Originalliteratur ermöglicht wird.

1.2 Grundlegende und Übersichtsliteratur

APT, K.R. *Ten Years of Hoare's Logic: A Survey - Part I.* ACM Transact. on Progr. Lang. and Systems 3(**1981**)4

de BAKKER, J.W. *Mathematical Theory of Program Correctness.* **1980**, Prentice Hall International, Englewood Cliffs

DIJKSTRA, E.W. *A Diszipline of Programming.* **1976**, Prentice Hall

FUTSCHEK, G. *Programmentwicklung und Verifikation.* **1989**, Springer-Verlag Wien, New York

GRIES, D. *The Science of Programming.* **1981**, Springer-Verlag

HOARE, C.A.R. *An Axiomatic Basis for Computer Programming.* Comm. ACM 12(**1969**)10, pp. 576-583

LOECKX, J. and SIEBER, K. *The Foundation of Program Verification.* **1987**, Teubner-Verlag Stuttgart

RIEDEWALD, G. u.a. *Formale Beschreibung von Programmiersprachen.* **1983**, Akademie-Verlag

1.3 Vorbemerkungen zum allgemeinen Kalkülbegriff

Die Formalisierung mathematischer u.a. Theorien ist notwendig für :

- Grundlagenuntersuchungen wie zur Widerspruchsfreiheit, Vollständigkeit usw.
- „Beherrschung“ der Theorie durch den Computer

Die historisch ersten und wohluntersuchten formalisierten Theorien sind „logische“ Kalküle, generell ist die Formalisierung aber nicht darauf beschränkt.

Notwendig zur Formalisierung einer Theorie, zum Aufbau eines formalen Kalküls ist die

1. Festlegung der *Ausdrucksmittel*, dies umfaßt die Definition gewisser Zeichenreihen als „Ausdrücke“. Z sei ein Alphabet, L die Menge der Ausdrücke: $L \subseteq Z^*$ ist die zugrundegelegte *Sprache*.

2. Festlegung der *Satzmenge*, der Menge S der *Theoreme*: $S \subseteq L$.

   Dazu existieren zwei Möglichkeiten :

   - semantisch definiert — Dies geschieht z.B. über einen Gültigkeitsbegriff („gültig“, „wahr“, „allgemeingültig“,...)
   - syntaktisch definiert — mit Hilfe einer sogenannten Ableitbarkeitsrelation

Beide definieren jeweils einen Hüllenoperator F (d.h. eine Abbildung mit bestimmten Eigenschaften), der einer Teilmeng $A \subseteq L$ wieder eine Teilmenge $B \subseteq L$ zuordnet, wobei insbesondere $F(S) \subseteq S$ sein muß.

Bei den *syntaktisch* definierten *formalen Kalkülen* wird die syntaktische Ableitungsrelation $\vdash$ im allgemeinen mit Hilfe von *Axiomen* und *Schlußregeln* definiert.

Ein *Theorem* aus S ist dann immer das letzte Glied einer endlichen Ableitungskette bzw. eines *Beweises*. In eine Ableitungskette bzw. einen Beweis darf jedes Axiom geschrieben werden, und dazu jeder Ausdruck, der durch Anwendung einer Schlußregel auf vorher hingeschriebene Ausdrücke entsteht.

Beispiel 1

Es seien A und B Ausdrücke (z.B. der Prädikatenlogik), und folgende Schlußregel sei erlaubt :

$$\left.\begin{array}{c} A \\ A \to B \end{array}\right\} \text{„Prämissen“} \qquad \text{als „Modus ponens“ bekannt}$$

$$\overline{\qquad B \qquad} \quad \text{„conclusio“(Schlußfolgerung, Behauptung)}$$

Folgende Ableitung sei schon erfolgt :

$$
\begin{array}{c}
C_1 \\
C_2 \\
\vdots \\
C_n \\
A \to B \\
C_{n+1} \\
\vdots \\
C_m \\
A \\
C_{m+1}
\end{array}
$$

Dann darf wegen der Schlußregel Modus ponens angefügt werden: B

Man schreibt dann:

$$\vdash B$$

Ist X eine Menge von Ausdrücken und $\vdash$ eine wie oben definierte Ableitungsrelation, so bedeutet $X \vdash B$, daß B aus X ableitbar (beweisbar) ist, in derselben Weise wie oben erläutert, nur daß *zusätzlich* zu den Axiomen jeder Ausdruck aus X in die Ableitungskette aufgenommen werden darf.

Manchmal wird mit $X^{\vdash}$ oder $F_{\vdash}(X)$ die Menge aller (in diesem Sinne) aus X ableitbaren Ausdrücke bezeichnet. $F_{\vdash}$ ist dann der weiter oben aufgeführte Hüllenoperator.

Im folgenden haben wir es stets mit „formalisierten Theorien 1. Stufe“ bzw. „formalisierten elementaren Theorien“ zu tun. Dies sind spezielle Kalküle, wobei

1. die Ausdrücke aus L Ausdrücke eines Prädikatenkalküls erster Stufe sind (mit den gerade benötigten speziellen Individuenvariablen, Funktionen- und Prädikatensymbolen). L ist damit eine sog. „elementare Sprache“. (Der Prädikatenkalkül erster Stufe zeichnet sich dadurch aus, daß die Quantoren nur für Individuenvariablen zugelassen sind.)
2. die Satzmenge syntaktisch durch eine Ableitungsrelation definiert ist, d.h. es genügt, die Menge A der Axiome und eine Menge R von Schlußregeln zur Festlegung der Satzmenge anzugeben.

Zusammenfassung

Zur Festlegung einer formalisierten elementaren Theorie T ist also ein Tripel anzugeben: $T = (L, A, R)$.

1.4 Verwendete grundlegende Begriffe und Bezeichnungen aus der Prädikatenlogik

Zum Verständnis der Vorlesung sind aus der „Prädikatenlogik“ einige Begriffe und Zusammenhänge ins Gedächtnis zurückzurufen (Wiederholung!) - bzw. für diejenigen, die wie die Informatikstudenten noch nicht in den Genuß einer Prädikatenlogik-Vorlesung gekommen sind, sind diese Begriffe vorab kurz einzuführen und zu erläutern. Selbstverständlich können sie im Rahmen dieser Vorlesung nicht exakt definiert, sondern nur intuitiv erklärt werden. Für die prädikatenlogische „Grundsteinlegung“ dieser Vorlesung muß auf die (im Studiengang parallel oder später liegende) einschlägige Vorlesung bzw. auf das Selbststudium der Literatur verwiesen werden.

V Menge der Individuenvariablen, kurz *Variablen* z.B.: $x, y, z, \ldots, a, b, c, \ldots, i, j, k, \ldots$

`true` , `false` logische *Konstanten* (statt W,F)

TERM Menge der *Terme*
(abgeschlossen gegenüber der üblichen Termbildung mit allen im Alphabet des Prädikatenkalküls vorkommenden Funktions- bzw. Operationssymbolen),
z.B.: $a + b$, $i \cdot j$, $f(x, y)$, $2n + 1$, $x!$, a^b

AUS Menge der *Ausdrücke* (prädikatenlogische Formeln)
des Prädikatenkalküls mit Identität

BAUS Menge der BOOLE*schen Ausdrücke* $=_{df}$ quantifikatorfreie Ausdrücke (Formeln).

Zu den in der Prädikatenlogik üblichen Termbildungsregeln kommt noch die Bildung sog. *bedingter Terme* hinzu:
Wenn $b \in$ **BAUS** und $t_1, t_2 \in$ **TERM**, so

$$\texttt{if}\ \ b\ \texttt{then}\ \ t_1\ \texttt{else}\ \ t_2\ \texttt{fi}\ \in \textbf{TERM}\ .$$

Für $b \in$ **BAUS**, $p, q \in$ **AUS** sei

$$\texttt{if}\ \ b\ \texttt{then}\ \ p\ \texttt{else}\ \ q\ \texttt{fi}$$

immer Abkürzung von

$$(b \wedge p) \vee (\neg b \wedge q).$$

Derartige *bedingte Ausdrücke* sehen wir der Einfachheit halber auch als Elemente von **AUS** an.

Beispiel 2

zu BAUS :
$a + b = b + a$
$x < y$
$n \geq 5 \wedge y \geq 0$
$x \neq 0 \rightarrow (1/x)^{-1} = x$
$x = 0 \vee 1/(1/x) = x$

zu AUS :
$\forall a \forall b(a + b = b + a)$
$\forall x \exists y(x + y = 0)$
$\forall n \forall m(n \geq 2 \wedge m \geq 1 \rightarrow m^n > m)$

Beispiel 3

der Festlegung einer elementaren Theorie
Im Alphabet sei speziell gegeben:

Individuensymbol : e
zweistelliges Funktionssymbol: $\circ$

Damit ist die Sprache L umrissen.

Zu dem Axiomensystem A sollen gehören

spezielle Axiome (Gruppenaxiome):

$$\forall x \forall y \forall z \, ((x \circ y) \circ z = x \circ (y \circ z))$$

$$\forall x (e \circ x = x)$$

$$\forall x \exists y (y \circ x = e)$$

Hinzu kommen die logischen Axiome mit (logischen) Schlußregeln.

Damit ist die elementare Gruppentheorie als syntaktisch definierte formalisierte Theorie gegeben.

Zur Semantik des Prädikatenkalküls führen wir noch folgende Bezeichnungen ein:

D Individuenbereich (*Wertebereich*),

$\Im$ *Interpretation*
ist eine eindeutige Abbildung der Funktionssymbole, einschließlich der Individuenkonstanten in die Menge der Funktionen (einschließlich Konstanten) über D, der Prädikatensymbole in die Menge der Relationen über D

σ, δ *Belegung, Zustand*
ist eine eindeutige Abbildung $\sigma : \underline{\mathbf{V}} \to D$

$\underline{\mathbf{ST}}$ Menge aller *Zustände*

Zur Auswertung der Terme und Ausdrücke sind zweistellige Funktionen

$$\text{wert}_{\Im}$$

gegeben, deren Argumente der auszuwertende Term (bzw. Ausdruck) und der (aktuelle) Zustand sind. Die Abhängigkeit von der gegebenen Interpretation ist in der Bezeichnung ebenfalls ausgedrückt; sie wird im weiteren oft weggelassen werden.

$$\text{wert}_{\Im}(t, \sigma) \in D, \quad \text{für} \quad t \in \underline{\mathbf{TERM}}, \, \sigma \in \underline{\mathbf{ST}}$$

$$\text{wert}_{\Im}(p, \sigma) \in \{W, F\} \quad \text{für} \quad p \in \underline{\mathbf{AUS}}, \, \sigma \in \underline{\mathbf{ST}}$$

$\models_{\Im} p$ p *gültig* bei der Interpretation $\Im$
$(=_{df} \text{wert}_{\Im}(p, \sigma) = W \quad \text{für alle} \quad \sigma \in \underline{\mathbf{ST}})$

Kapitel 2

Formale Theorien für Programmiersprachen

2.1 Programmspezifikationen und Beweissysteme für Programme

Es sei

$$T_0 = (L_0, A_0, R_0)$$

eine formalisierte Theorie erster Stufe mit Identität, z.B. die elementare Gruppentheorie, die elementare Arithmetik usw. Über dem Alphabet des Prädikatenkalküls, erweitert um einige Hilfssymbole wie :=, ; , `if` , `then` , `else` , `while` (Progammjunktoren) und Klammern, seien zwei weitere Sprachen festgelegt:

- die Menge **PR** der *Programme* (Programmiersprache), und mit deren Hilfe
- die Menge **SP** der *Spezifikationen* („spezifizierte Programme“, „asserted programs“).

Die konkrete Festlegung dieser Sprachen erfolgt im Detail im weiteren Verlauf der Vorlesung.

Bei gegebener Interpretation $\Im$ wird die Semantik der Programme und Spezifikationen durch weitere geeignete Fortsetzungen der Interpretation (analog wie oben zur Auswertung der Terme und Ausdrücke) bestimmt, d.h. es seien Funktionen

$$\mathrm{val}_\Im \quad \text{und} \quad \mathrm{wert}_\Im$$

gegeben mit:

$$\mathrm{val}_\Im : \mathbf{PR} \times \mathbf{ST} \dashrightarrow \mathbf{ST} \quad \text{ist eine partielle Funktion}$$

$$\mathrm{wert}_\Im : \mathbf{SP} \times \mathbf{ST} \to \{W, F\} \quad \text{ist eine volle Funktion}$$

Die bei Ausdrücken üblichen Sprechweisen werden nun auf Spezifikationen erweitert:
$\Phi \in \mathbf{SP}$ heißt *gültig* bei der Interpretation $\Im$, kurz

$$\models_\Im \Phi,$$

wenn $\mathrm{wert}_\Im(\Phi, \sigma) = W$ für alle $\sigma \in \mathbf{ST}$ gilt.
$\Im$ heißt dann ein *Modell* von Φ.

Eine *formalisierte Theorie für Programmiersprachen*

$$T = (L, A, R)$$

ist nun gegeben durch

$$L = \mathbf{AUS} \cup \mathbf{SP} \quad , \text{wobei } L_0 = \mathbf{AUS},$$

$$A = A_0 \cup A_{\mathbf{SP}},$$

$$R = R_0 \cup R_{\underline{\mathbf{SP}}},$$

d.h. eine Erweiterung von T_0 um Sezifikationen sowie um Axiome, die selbst Spezifikationen sind, und spezifische Schlußregeln für Spezifikationen (d.h. solche, deren Conclusio Spezifikation ist).

Wir haben also:

A_0 Axiome der elementaren Theorie,

$A_{\underline{\mathbf{SP}}}$ Axiome für Spezifikation ($A_{\underline{\mathbf{SP}}} \subseteq \underline{\mathbf{SP}}$),

R_0 (logische) Schlußregeln,

$R_{\underline{\mathbf{SP}}}$ Schlußregeln für Spezifikationen.

Für uns interessant sind nur die Axiome aus $A_{\underline{\mathbf{SP}}}$ und die Regeln aus $R_{\underline{\mathbf{SP}}}$, nicht die Axiome und Regeln der zugrundegelegten Theorie (die es wegen des GÖDELschen Unvollständigkeitsatzes für viele Theorien, z.B. die elementare Arithmethik, vollständig überhaupt nicht gibt).

Deshalb abstrahieren wir von diesen in folgender Weise :
Für eine feste Interpretation $\Im$ sei

$$\underline{\mathbf{TR}}_{\Im} \;=\; \{p \mid p \in \underline{\mathbf{AUS}} \wedge \models_{\Im} p\},$$

d.h. alle bei der Interpretation $\Im$ gültigen Formeln p (das ist eine semantisch definierte Satzmenge). Gegenstand des Interesses sind nunmehr formalisierte Theorien

$$T_{\Im} \;=\; (\underline{\mathbf{AUS}} \cup \underline{\mathbf{SP}},\ \underline{\mathbf{TR}}_{\Im} \cup A_{\underline{\mathbf{SP}}},\ R_{\underline{\mathbf{SP}}})$$

für beliebige Interpretationen $\Im$. Dies sind sogenannte „*durch Orakel unterstützte Theorien*".

Definition 2.1

Das Paar $B \;=\; (A_{\underline{\mathbf{SP}}},\ R_{\underline{\mathbf{SP}}})$ heißt *Beweissystem* für Programme. ϕ heißt Theorem von B bei der Interpretation $\Im$, kurz $\underline{\mathbf{TR}}_{\Im} \vdash_B \phi$ oder $\vdash_{T_{\Im}} \phi$, genau dann, wenn ϕ Spezifikation ist ($\phi \in \underline{\mathbf{SP}}$) mit ϕ ist beweisbar (ableitbar) in $T_{\Im}$.

Der Entwurf von Axiomen und Schlußregeln für eine bestimmte Programmiersprache $\underline{\mathbf{PR}}$ beruht auf einem bestimmten Verständnis der Programme und Spezifikationen, d.h. auf einer gewissen Auswahl möglicher Interpretationen (und deren Fortsetzung für Programme und Spezifikationen). Das Beweissystem darf nicht im Widerspruch zu der ausgewählten Klasse INT der Interpretationen stehen.

Definition 2.2

Ein Beweissystem B für Programme heißt *widerspruchsfrei* (*korrekt*) bezüglich einer Klasse INT von Interpretationen genau dann, wenn für alle $\Im \in$ INT und alle $\phi \in \underline{\mathbf{SP}}$ gilt : wenn $\vdash_{T_{\Im}} \phi$, so $\models_{\Im} \phi$. $(\star)$
Wenn $(\star)$ erfüllt ist, so heißt $\Im$ ein *Modell* des Beweissystems $B \;=\; (A_{\underline{\mathbf{SP}}},\ R_{\underline{\mathbf{SP}}})$.

Man nennt eine Schlußregel **korrekt** bei einer Interpretation $\Im$ genau dann, wenn aus der Gültigkeit aller Prämissen (bei $\Im$) stets die Gültigkeit der conclusio folgt.

Lemma 2.1

$\Im$ ist ein Modell von $(A_{\underline{\mathbf{SP}}},\ R_{\underline{\mathbf{SP}}})$, wenn :

1. *$\Im$ ein Modell von $A_{\underline{\mathbf{SP}}}$ ist, d.h. alle Axiome aus $A_{\underline{\mathbf{SP}}}$ sind in $\Im$ gültig, und*

2. *jede Schlußregel aus $R_{\underline{\mathbf{SP}}}$ korrekt bei $\Im$ ist.*

Beweis

Der Beweis erfolgt induktiv über die Länge der Ableitung :
Seien 1. und 2. erfüllt und es gelte für $\phi \in \underline{\mathbf{SP}}$, daß $\vdash_{T_\Im} \phi$ ist. Aus der Ableitbarkeit ergeben sich zwei Fälle :

a) ϕ ist Axiom, d.h. $\phi \in A_{\underline{\mathbf{SP}}}$. Aus 1. folgt dann $\models_\Im \phi$.

b) Es existieren bereits $\phi_1, \phi_2, \ldots, \phi_n \in \underline{\mathbf{AUS}} \cup \underline{\mathbf{SP}}$ und eine Regel $r \in R_{\underline{\mathbf{SP}}}$ der Form :

$$\frac{\phi_1, \ldots, \phi_n \quad (\leftarrow \quad \text{Prämissen})}{\phi \qquad (\leftarrow \quad \text{conclusio})} \qquad mit \quad \vdash_{T_\Im} \phi_i \quad (1 \leq i \leq n).$$

Ist $\phi_i \in \underline{\mathbf{AUS}}$, dann folgt $\phi_i \in \underline{\mathbf{TR}}_\Im$ und schließlich $\models_\Im \phi_i$.

Ist $\phi_i \in \underline{\mathbf{SP}}$, dann ist nach Induktionsvoraussetzung $\models_\Im \phi_i$

Wegen 2. (Schlußregel korrekt) gilt also, da die Prämissen gültig sind, daß auch die conclusio gültig ist, und somit $\models_\Im \phi$.

q.e.d.

Die zweite interessierende Eigenschaft für Beweissysteme ist die, ob alle durch Spezifikationen beschreibbaren gültigen Eigenschaften von Programmen auch beweisbar sind (Frage nach der Vollständigkeit):

Definition 2.3

in Beweissystem $(A_{\underline{\mathbf{SP}}}, R_{\underline{\mathbf{SP}}})$ heißt *(relativ) vollständig* (bezüglich einer Klasse INT von Interpretationen) genau dann, wenn für alle $\Im(\in \text{INT})$ und alle $\phi \in \underline{\mathbf{SP}}$ gilt :

$$\text{wenn } \models_\Im \phi, \text{ so } \vdash_{T_\Im} \phi.$$

Das Beweissystem ist also vollständig, wenn alle gültigen Spezifikationen auch stets beweisbar sind.

2.2 Beispiele für mögliche Spezifikationen und deren Bedeutung

Es werden Beispiele für die Syntax von $\underline{\mathbf{SP}}$ und mögliche Fortsetzungen der Interpretation $\Im$ auf $\phi \in \underline{\mathbf{SP}}$ angegeben.
Im folgenden seien $p, q \in \underline{\mathbf{AUS}}$, $s, s_1, s_2 \in \mathbf{PR}$, $\sigma \in \underline{\mathbf{ST}}$

a) Partielle Korrektheit

$\phi \in \underline{\mathbf{SP}}$ hat die Form

$$\{p\}S\{q\}.$$

$\text{wert}_{\Im}(\{p\}S\{q\}, \sigma) = W$ genau dann , wenn gilt :

Wenn $\text{wert}_{\Im}(p, \sigma) = W$ und $\text{val}_{\Im}(S, \sigma)$ ist definiert,
so $\text{wert}_{\Im}(q, \text{val}_{\Im}(S, \sigma)) = W$.

„ Wenn p gültig ist bei σ, so ist q gültig bei dem Zustand, der durch die Abarbeitung von S bei σ entsteht, sofern dieser definiert ist. “

Beispiel 4

Trivialerweise gilt immer:

(1) $\models_{\Im} \{\underline{\texttt{false}}\}S\{q\}$

(2) $\models_{\Im} \{p\}S\{\underline{\texttt{true}}\}$

Für das Programm S_0 :

`x:=1; i:=m; while i≠0 do x:=x+x; i:=i-1 od`

gilt bei üblicher Programminterpretation über den ganzen Zahlen z.B. :

(3) $\models_{\Im_0} \{\underline{\texttt{true}}\}S_0\{x = 2^m\}$

(4) $\models_{\Im_0} \{m \geq 0\}S_0\{x = 2^m\}$

b) Totale Korrektheit

$\phi \in$ **SP** hat dieselbe Form wie oben, aber folgende Bedeutung :
$\text{wert}_{\Im}(\{p\}S\{q\}, \sigma) = W$ genau dann, wenn gilt :

Wenn $\text{wert}_{\Im}(p, \sigma) = W$,
so ist $\text{val}_{\Im}(S, \sigma)$ definiert und es ist $\text{wert}_{\Im}(q, \text{val}_{\Im}(S, \sigma)) = W$.

Bei dieser Semantik der Spezifikation gilt bezüglich (1), (2), (3) und (4) :

(1) und (4) gelten, (2) und (3) aber nicht.

Sprechweisen für a) und b):

p ist Precondition, Vorbedingung bzw. Eingangsbedingung
q ist Postcondition, Nachbedingung bzw. Ausgangsbedingung
$\models_{\Im} \{p\}S\{q\}$ bedeutet:
„Das Programm S ist *partiell* (bzw. *total*) *korrekt* bezüglich p und q bei der Interpretation $\Im$.“ $\{p\}S\{q\}$ heißen auch *Korrektheitsformeln.*

c) Termination

$\phi \in$ **SP** hat die Form

$$\{p\}S.$$

$\text{wert}_{\Im}(\{p\}S, \sigma) = W$ genau dann, wenn gilt :

Wenn $\text{wert}_{\Im}(p, \sigma) = W$, so ist $\text{val}_{\Im}(S, \sigma)$ definiert.

Das heißt, daß das Programm S bei der Eingabe von σ terminiert.

d) Äquivalenz

$\phi \in$ **SP** hat die Form

$$S_1 = S_2.$$

$\text{wert}_{\Im}(S_1 = S_2, \sigma) = W$ genau dann, wenn gilt :

$\text{val}_{\Im}(S_1, \sigma) = \text{val}_{\Im}(S_2, \sigma)$

Dabei steht $=$ für die „ starke “ Gleichheit, d.h. wenn eine von beiden Seiten definiert ist, so ist es auch die andere und beide sind gleich.

Beispiel 5

$$\models_{\Im} \; x := y,\, y := x \;=\; x := y$$

e) Pseudoäquivalenz

$\phi \in \underline{\mathbf{SP}}$ hat die Form $S_1 \Rightarrow S_2$
$\text{wert}_{\Im}(S_1 \Rightarrow S_2, \sigma) = W$ genau dann, wenn gilt :
$\text{val}_{\Im}(S_1, \sigma)$ ist nicht definiert, oder
$\text{val}_{\Im}(S_1, \sigma) = \text{val}_{\Im}(S_2, \sigma)$

Beispiel 6

$$\models_{\Im} \; y := \frac{1}{z},\, y := x - x \;\Rightarrow\; y := 0$$

f) termweise Korrektheit

$\phi \in \underline{\mathbf{SP}}$ hat die Form $\{t_1\}S\{t_2\}$ mit $t_1, t_2 \in \underline{\mathbf{TERM}}$
$\text{wert}_{\Im}(\{t_1\}S\{t_2\}, \sigma) = W$ genau dann, wenn gilt :
$\text{val}_{\Im}(S, \sigma)$ ist nicht definiert, oder es ist
$\text{wert}_{\Im}(t_1, \sigma) = \text{wert}_{\Im}(t_1, \text{val}_{\Im}(S, \sigma))$

Zum Kommentar vergleiche (eventuell) später.

Kapitel 3

HOARE–Logik für while–Programme

Im folgenden sei die Menge **SP** , d.h. die Menge der Spezifikationen, stets wie bei den Beispielen a) festgelegt, deren Bedeutung wie bei der „partiellen Korrektheit“.

3.1 Geradeaus–Programme

Definition 3.1

$\mathbf{PR}_0$ sei die kleinste Menge von Zeichenreihen, für die gilt :

(0) `skip` $\in \mathbf{PR}_0$,
und mit $x \in \mathbf{V}$ und $t \in \mathbf{TERM}$ ist $x := t \quad \in \mathbf{PR}_0$.

(1) mit $S_1, S_2 \in \mathbf{PR}_0$ ist auch $S_1; S_2 \ \in \mathbf{PR}_0$.

(2) mit $b \in \mathbf{BAUS}$ und $S_1, S_2 \in \mathbf{PR}_0$ ist auch
`if` b `then` S_1 `else` S_2 `fi` $\in \mathbf{PR}_0$

Die Elemente von $\mathbf{PR}_0$ heißen *Geradeausprogramme.*

Beispiel 7

Beispiele für Programme aus $\mathbf{PR}_0$ (bei einer entsprechenden Bezeichnung der Funktions– und Prädikatensymbole :

```
x := x + y
```

```
if x≥ 0 then y := x else y := x fi ; z := y - x;
```

```
a := 2b; i := if k≥ n then k - i else k + i fi ; b := a^i
```

Beispiel 8

Beispiele für gültige Korrektheitsformeln (partielle Korrektheit)
$\Im$ sei eine „vernünftige“ Interpretation (d.h. Modell der zugrundegelegten Theorie) :

$\models_{\Im} \{x = 0\}$`x := x + 1`$\{x = 1\}$

$\models_{\mathfrak{Z}} \{n > 5\}\underline{\texttt{skip}}\ \{a + b = b + a\}$

$\models_{\mathfrak{Z}} \{x = 3\}\ \underline{\texttt{if}}\ \texttt{x}\geq\ \texttt{0}\ \underline{\texttt{then}}\ \texttt{x := x + y}\ \underline{\texttt{else}}\ \texttt{x := x - y}\ \underline{\texttt{fi}}\ \ \{x = y + 3\}$

$\models_{\mathfrak{Z}} \{\underline{\texttt{true}}\}\ \texttt{x := x + 1}\ \{\exists y(x = y + 1)\}$

$\models_{\mathfrak{Z}} \{x = 2 \wedge y = 1\}\texttt{y := y + x; x := x + 1}\ \{x < 5 \wedge y = x\}$

Beispiel 9

Beispiele für korrekte Schlußregeln :

1.
$$\frac{\{p\}S\{q\}, q \to r}{\{p\}S\{r\}}$$

Diese angegebene Schlußregel ist ein Schlußregelschema, steht also für eine Schar konkreter Regeln.

2.
$$\frac{\{\underline{\texttt{true}}\}x := 0, y := x + 1\{x = 0 \wedge y = 1\}}{\{true\}a := 0, b := a + 1\{a = 0 \wedge b = 1\}}$$

Dies ist im Gegensatz zum vorigen Beispiel eine konkrete Schlußregel, kein Schema. Offensichtlich ist sie aber verallgemeinerungsfähig, z. B. zu einem allgemeinen Umbenennungsregelschema.

3.
$$\frac{x = 0}{\{\underline{\texttt{true}}\}S\{x = 1\}}$$

Die hier angegebene Schlußregel ist richtig, obwohl sie verblüffend aussieht. Sie ist deshalb korrekt, weil für die Korrektheit einer Schlußregel nur verlangt ist, daß die conclusio gültig ist, falls auch die Prämisse gültig ist. Es ist aber im allgemeinen $\models_{\mathfrak{Z}} x = 0$ keine gültige Formel. Sie ist deshalb hier angeführt, um ein Gefühl für den Umfang aller gültigen Formeln zu vermitteln, der durch ein vollständiges Beweissystem erfaßt werden muß.

3.1.1 Das HOAREsche Beweissystem für Geradeausprogramme

Wir betrachten nun folgendes von HOARE eingeführtes Beweissystem:

$A_{\underline{\texttt{SP}}}$: Axiome :

(0) Das $\underline{\texttt{skip}}$ –Axiom (oder Leeraxiom) :

$$L \quad : \quad \{p\}\ \underline{\texttt{skip}}\ \{p\}$$

(1) Das HOAREsche Zuweisungsaxiom :

$$Z \quad : \quad \{p^x/_t\}\ \texttt{x := t}\ \{p\}$$

$R^0_{\underline{\mathbf{SP}}}$: Beweisregeln :

(2) Kompositionsregel :

$$R_{;} \quad : \quad \frac{\{p\}S_1\{q\}, \{q\}S_2\{r\}}{\{p\}S_1; S_2\{r\}}$$

(3) Verzweigungsregel :

$$R_{if} \quad : \quad \frac{\{p \wedge b\}S_1\{q\}, \{p \wedge \neg b\}S_2\{q\}}{\{p\}\underline{\texttt{if}}\, b\, \underline{\texttt{then}}\, S_1\, \underline{\texttt{else}}\, S_2\, \underline{\texttt{fi}}\{q\}}$$

(4) Implikationsregel :

$$R_{\rightarrow} \quad : \quad \frac{p \rightarrow p_1, \{p_1\}S\{q_1\}, q_1 \rightarrow q}{\{p\}S\{q\}}$$

Mit $H_0 = (A_{\underline{\mathbf{SP}}}, R^0_{\underline{\mathbf{SP}}})$ sei dieses Beweissystem für Geradeausprogramme bezeichnet.

Beispiel 10

Beispiele von Beweisen

a) Verifikation, daß das Programm S

$$z := x; \quad x := y; \quad y := z$$

die Werte von x und y vertauscht.

Der Beweis erfolgt von „hinten nach vorn“.

1. nach Z : $\vdash_{H_0} \{z = a \wedge x = b\}\texttt{y:=z}\ \{y = a \wedge x = b\}$
2. nach Z : $\vdash_{H_0} \{z = a \wedge y = b\}\texttt{x:=y}\ \{z = a \wedge x = b\}$
3. nach Z : $\vdash_{H_0} \{x = a \wedge y = b\}\texttt{z:=x}\ \{z = a \wedge y = b\}$
4. Anwendung von $R_{;}$ auf 2. und 3. :
   $\vdash_{H_0} \{x = a \wedge y = b\}\texttt{z:=x; x:=y}\ \{z = a \wedge x = b\}$
5. Anwendung von $R_{;}$ auf 1. und 4. :
   $\vdash_{H_0} \{x = a \wedge y = b\}\texttt{z:=x; x:=y; y:=z}\ \{y = a \wedge x = b\}$

b) Verifikation, daß nach Ausführung von S

$$x := a - b;\ \underline{\texttt{if}}\ x \geq 0\ \underline{\texttt{then}}\ \underline{\texttt{skip}}\ \underline{\texttt{else}}\ x := -x\ \underline{\texttt{fi}}$$

$x =\mid a - b \mid$ gilt.

1. $\vdash_{H_0} \{\underline{\texttt{true}}\}\texttt{x:=a-b}\ \{x = a - b\}$
   wegen Z und $R_{\rightarrow}$ und weil :
   $\vdash_{H_0} \underline{\texttt{true}} \rightarrow a - b = b - a$
   ableitbar in H_0, d.h. $\in \underline{\mathbf{TR}}_{\Im}$ folgt :

$$\frac{\underline{\texttt{true}} \rightarrow a - b = b - a, \{a - b = a - b\}\ \text{x:=a-b}\ \{x = a - b\}, x = a - b \rightarrow x = a - b}{\{true\}x := a - b\{x = a - b\}}$$

2. $\vdash_{H_0} \{x = a - b \wedge a - b \geq 0\}\underline{\texttt{skip}}\ \{x = a - b \wedge a - b \geq 0\}$
   Dies ist wegen L.
3. $\vdash_{H_0} \{x = a - b \wedge a - b \geq 0\}\texttt{x:=-x}\ \{x = b - a, a - b < 0\}$
   Es gilt wegen Z, $R_{\rightarrow}$ und weil :
   $\vdash_{H_0} x = a - b \wedge a - b < 0 \rightarrow -x = b - a \wedge a - b < 0$
   ableitbar in H_0.

4. $\vdash_{H_0} x = a - b \wedge a - b > 0 \rightarrow q$
   mit $q = (a - b \geq 0 \wedge x = a - b) \vee (a - b < 0 \wedge x = b - a)$.

5. $\vdash_{H_0} x = b - a \wedge a - b < 0 \rightarrow q$

6. $\vdash_{H_0} \{x = a - b \wedge x \geq 0\}$`skip` $\{q\}$
   $R_{\rightarrow}$ angewendet auf 2. und 4. und wegen der Umformung :
   $\vdash_{H_0} x = a - b \wedge (x \geq 0) \rightarrow x = a - b \wedge a - b \geq 0$

7. $\vdash_{H_0} \{x = a - b \wedge \neg(x = 0)\}$`x:=-x` $\{q\}$
   $R_{\rightarrow}$ angewendet auf 3. und 5. und wegen :
   $\vdash_{H_0} \{x = a - b \wedge \neg(x = 0) \rightarrow x = a - b \wedge a - b < 0$

8. $\vdash_{H_0} \{x = a - b\}$`if` x$\geq$ 0 `then` `skip` `else` x := -x `fi` $\{q\}$
   Anwendung von R_{if} auf 6. und 7.

9. $\vdash_{H_0} q \rightarrow x =\mid a - b \mid$

10. $\{$`true`$\}\ S\ \{x =\mid a - b \mid\}$ wegen Anwendung von $R_;$ auf 1 und 8 und auf 9.

Zu einer Strategie der Beweissuche :

„Rückwärts-Anwendung der Schlußregeln“

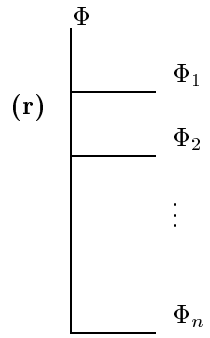
Betrachte Schlußregel

$$r : \frac{\Phi_1, \Phi_2, \ldots, \Phi_n}{\Phi}$$

mit Prämissen $\Phi_1, \ldots, \Phi_n \in$ **AUS** $\cup$ **SP** und Conclusio $\Phi \in$ **SP**.

Die Rückwärtsanwendung der Regel r auf das (Beweis-)„Ziel“ Φ ersetzt das Ziel Φ durch die n Teilziele $\Phi_1 \ldots \Phi_n$. Dann ist jedes Teilziel wiederum als Ziel anzusehen und es sind dessen Teilziele zu suchen, bis man auf Axiome (`skip` –Axiom, Zuweisungsaxiom oder Ausdrücke aus $\mathbf{TR}_{\Im}$) stößt. Hat man auf diese Weise *alle* Teilziele „erreicht“, so ist der Beweis gelungen. (Man könnte ihn dann sofort in eine Vorwärtsableitung umschreiben.)

Symbolisch:



Am Beispiel des letzten Beweises für S :

$$x := a - b;\ \texttt{if}\, x > 0\, \texttt{then}\, \texttt{skip}\, \texttt{else}\, x := -x\, \texttt{fi}$$

Ziel : $\{$`true`$\}\ S\ \{x = |a - b|\ \}$

$(R_;)$

(1) $\underbrace{\{\, a-b = a-b \,\}}_{\underline{\texttt{true}}}\; x := a-b \,\{\, x = a-b \,\}$ (wegen (Z))

(2) $\{\, x = a-b \,\}\, \underline{\texttt{if}}\; x \geq 0 \;\underline{\texttt{then}}\; \underline{\texttt{skip}}\; \underline{\texttt{else}}\; x := -x \;\underline{\texttt{fi}}\, \{\, x = |a-b| \,\}$

(2)

$(R_{\underline{\texttt{if}}})$

(3) $\{\, x = a-b \wedge x \geq 0 \,\}\, \underline{\texttt{skip}}\, \{\, x = |a-b| \,\}$

(4) $\{\, x = a-b \wedge x < 0 \,\}\, x := -x \,\{\, x = |a-b| \,\}$

(3)

$(R_{\rightarrow})$

(5) $x = a-b \;\wedge\; x \geq 0 \;\rightarrow\; x = a-b \;\wedge\; x \geq 0$ $(\in \underline{Tr_{\Im}})$

(6) $\{\, x = a-b \wedge x \geq 0 \,\}\, \underline{\texttt{skip}}\, \{\, x = a-b \wedge\; x \geq 0 \,\}$ (wegen (L))

(7) $x = a-b \;\wedge\; x \geq 0 \;\rightarrow\; x = |a-b|$ $(\in \underline{Tr_{\Im}})$

(4)

$(R_{\rightarrow})$

(8) $x = a-b \;\wedge\; x < 0 \;\rightarrow\; -x = |a-b|$ $(\in \underline{Tr_{\Im}})$

(9) $\{\, -x = |a-b| \,\}\, x := -x \,\{\, x = |a-b| \,\}$ (wegen (Z))

(10) $x = |a-b| \;\rightarrow\; x = |a-b|$ $(\in \underline{Tr_{\Im}})$

Damit ist der Beweis erbracht.

3.1.2 Semantik der Geradeausprogramme

Die Frage der *Korrektheit* und der *Vollständigkeit* des Beweissystems H_0 bleibt zu untersuchen. Voraussetzung dafür ist eine exakte Beschreibung der Semantik der Programme, also der Semantik von $\underline{\mathbf{PR}}_0$. Für jedes Programm $S \in \underline{\mathbf{PR}}_0$ ist $\mathrm{val}_{\Im}(S, \sigma)$ bei $\sigma \in \underline{\mathbf{ST}}$ festzulegen.

$$\mathrm{val}_{\Im} : \underline{\mathbf{PR}}_0 \;\times\; \underline{\mathbf{ST}} \;\longmapsto\; \underline{\mathbf{ST}}$$

wird induktiv über den Aufbau von S definiert.

Definition 3.2

Es seien $x \in \underline{\mathbf{V}}$, $t \in \underline{\mathbf{TERM}}$, $b \in \underline{\mathbf{BAUS}}$, $S_1, S_2 \in \underline{\mathbf{PR}}_0$ und $\sigma, \sigma' \in \underline{\mathbf{ST}}$ und $\Im$ eine Interpretation.

0.) $\text{val}_{\Im}(\underline{\texttt{skip}}, \sigma) = \sigma$

$\text{val}_{\Im}(x := t, \sigma) = \sigma'$ mit $\sigma'(y) = \begin{cases} \sigma(y) & \forall\, y \in \underline{\mathbf{V}} \quad mit \quad y \neq x \\ \text{wert}_{\Im}(t, \sigma) & für \quad y = x \end{cases}$

1.) $\text{val}_{\Im}(S_1; S_2, \sigma) = \text{val}_{\Im}(S_2, \text{val}_{\Im}(S_1, \sigma))$

2.) $\text{val}_{\Im}(\underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S_1\ \underline{\texttt{else}}\ S_2 \underline{\texttt{fi}}, \sigma) = \begin{cases} \text{val}_{\Im}(S_1, \sigma), \text{ falls } \text{wert}_{\Im}(b, \sigma) = W \\ \text{val}_{\Im}(S_2, \sigma), \text{ falls } \text{wert}_{\Im}(b, \sigma) = F \end{cases}$

3.1.3 Zur Widerspruchsfreiheit von H_0

Satz 3.1

H_0 ist korrekt bezüglich jeder Interpretation $\Im$, d.h. für beliebige Spezifikationen Φ gilt :
Wenn $\underline{\mathbf{TR}}_{\Im} \vdash_{H_0} \Phi$ *, so* $\models_{\Im} \Phi$.

Beweis

Wird entsprechend Lemma 2.1 auf Seite 8 geführt. Sei $\Im$ eine beliebige Interpretation.

1.) $\Im$ ist Modell von $A_{\underline{\mathbf{SP}}}$:

a) Sei $\sigma \in \underline{\mathbf{ST}}$ beliebig und sei $\text{wert}_{\Im}(p, \sigma) = W$, dann folgt $\text{wert}_{\Im}(p, \text{val}_{\Im}(\underline{\texttt{skip}}, \sigma)) = \text{wert}_{\Im}(p, \sigma) = W$, also $\models \{p\}\ \underline{\texttt{skip}}\ \{p\}$.

b) Sei $\text{wert}_{\Im}(p^x/_t, \sigma) = \text{wert}_{\Im}(p, \sigma')$ mit $\sigma(y) = \begin{cases} \sigma(y)\ für\ y \neq x \\ \text{wert}_{\Im}(t, \sigma)\ für\ y = x \end{cases}$

Also gilt : $\text{wert}_{\Im}(p^x/_t, \sigma) = \text{wert}_{\Im}(p, \overbrace{\text{val}_{\Im}(x := t, \sigma)}^{\sigma'}) = W$. Damit haben wir
$\models \{p^x/_t\}\ x := t\ \{p\}$.

2.) Jede Schlußregel aus $R_{\underline{\mathbf{SP}}}$ ist korrekt:

c) Sei $\models \{p\}\ S_1\ \{q\}$ **(*)** und $\models \{q\}\ S_2\ \{r\}$ **(**)**. Sei $\sigma \in \underline{\mathbf{ST}}$ beliebig und $\text{wert}_{\Im}(p, \sigma) = W$
$\rightarrow$ wegen **(*)** $\text{wert}_{\Im}(q, \underbrace{(S_1, \sigma)}_{\sigma_1}) = W$.

$\rightarrow$ wegen **(**)** $\text{wert}_{\Im}(r, \text{val}_{\Im}(S_2, \sigma_1)) = W$.
Also $\text{wert}_{\Im}(r, \text{val}_{\Im}(S_2, \text{val}_{\Im}(S_1, \sigma))) = W$
(Wegen Semantikdefinition) $\rightarrow$ $\text{wert}_{\Im}(r, \text{val}_{\Im}(S_1; S_2, \sigma)) = W$. Damit gilt $\models \{p\}\ S_1; S_2\ \{r\}$.

3.) Sei $\models \{p \wedge b\}\ S_1\ \{q\}$ **(+)** und $\models \{p \wedge \neg\, b\}\ S_2\ \{q\}$ **(++)**. Sei $\sigma \in \underline{\mathbf{ST}}$ mit $\text{wert}_{\Im}(p, \sigma) = W$.

Fall a) $\text{wert}_{\Im}(b, \sigma) = W$, dann $\text{wert}_{\Im}(p \wedge b, \sigma) = W$ wegen **(+)** $\text{wert}_{\Im}(q, \text{val}_{\Im}(S_1, \sigma) = W$ und $\text{wert}_{\Im}(q, \text{val}_{\Im}(\underline{\texttt{if}} \ldots, \sigma)) = W$

Fall b) $\text{wert}_{\Im}(b, \sigma) = F$, dann $\text{wert}_{\Im}(p \wedge \neg\, b, \sigma) = W$ wegen **(++)** $\text{wert}_{\Im}(q, \text{val}_{\Im}(S_2, \sigma)) = W$
$\rightarrow$ $\text{wert}_{\Im}(q, \text{val}_{\Im}(\underline{\texttt{if}} \ldots, \sigma)) = W \Longrightarrow \models_{\Im} \{p\}\ \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S_1 \underline{\texttt{else}}\ S_2\ \underline{\texttt{fi}}\ \{q\}$

4.) Sei $\models p \rightarrow p_1$ **(*)** , $\models q_1 \rightarrow q$ **(**)** und $\models \{p_1\}\ S\ \{q_1\}$ **(***)**. Sei $\sigma \in \underline{\mathbf{ST}}$ mit $\text{wert}_{\Im}(p, \sigma) = W$. Wegen **(*)** gilt $\text{wert}_{\Im}(p_1, \sigma) = W$ und wegen **(***)** gilt
$\text{wert}_{\Im}(q_1, \text{val}_{\Im}(s, \sigma)) = W$, damit folgt aus **(**)** $\text{wert}_{\Im}(q, \text{val}_{\Im}(S, \sigma)) = W$. Also $\models \{p\}\ S\ \{q\}$.

q.e.d.

3.1.4 Zur Vollständigkeit von H_0

Satz 3.2

Das Beweissystem H_0 ist vollständig.
(Das heißt für alle $\Im$ und alle $\Phi \in \underline{\mathbf{SP}}$ gilt : Wenn $\models_\Im \Phi$, so $\underline{\mathbf{TR}}_\Im \vdash_\Im \Phi$).

Beweis

Φ sei beliebige Spezifikation : $\{p\}\ S\ \{q\}$ mit $p, q \in \underline{\mathbf{AUS}}$, $S \in \underline{\mathbf{PR}}_0$. Es sei $\models_\Im \{p\}\ S\ \{q\}$. Beweis induktiv über Aufbau von S.

0.) Induktionsanfang

0.1) $S \equiv \underline{\texttt{skip}}$. $\models_\Im \{p\}\ \underline{\texttt{skip}}\ \{q\}$. D.h., wenn für $\sigma \in \underline{\mathbf{ST}}$ $\text{wert}_\Im(p,\sigma) = W$, so auch $\text{wert}_\Im(q, \text{val}_\Im(\underline{\texttt{skip}}, \sigma)) = W$, also $\text{wert}_\Im(q,\sigma) = W$. Damit ist $\models_\Im p \to q$, d.h. $p \to q \in \underline{\mathbf{TR}}_\Im$. Nach Axiom (L) : $\vdash_\Im \{p\}\ \underline{\texttt{skip}}\ \{p\}$, deshalb folgt wegen $R_\to$ und $p \to p \in \underline{\mathbf{TR}}_\Im$ auch $\vdash_\Im \{p\}\ \underline{\texttt{skip}}\ \{q\}$.

0.2) $S \equiv x := t$, $\models_\Im \{p\}\ x := t\ \{q\}$ d.h., wenn $\text{wert}_\Im(p,\sigma) = W$, so $\text{wert}_\Im(t,\sigma') = W$
mit $\sigma'(y) = \begin{cases} \text{wert}_\Im(t,\sigma)\ \textit{für}\ \ y = x \\ \sigma(y)\ \textit{sonst} \end{cases}$
Nach oben zitiertem allgemeinen Satz ist $\text{wert}_\Im(q,\sigma') = W = \text{wert}_\Im(q^x/_t, \sigma)$. Damit $\models_\Im p \to q^x/_t$. Also $p \to q^x/_t \in \underline{\mathbf{TR}}_\Im$. Nach Axiom (Z) gilt : $\vdash_{H_0} \{q^x/_t\}\ x := t\ \{q\}$. Damit haben wir mit $R_\to$ $\vdash_{H_0} \{p\}\ x := t\ \{q\}$

Bemerkung

Wir verwenden hier, daß mit $q \in \underline{\mathbf{AUS}}$ auch $q^x/_t \in \underline{\mathbf{AUS}}$ ist, d.h. eine Forderung an die „Ausdrucksfähigkeit“ der Sprache $\underline{\mathbf{AUS}}$, der sogenannten „assertion language“.

1.) Induktionsschluß : für S_1, S_2 sei die Behauptung erfüllt (Induktionsvoraussetzung).
$S \equiv S_1; S_2$ Sei $\models_\Im \{p\}\ S_1; S_2\ \{q\}$

! Angenommen, es gibt ein $r \in \underline{\mathbf{AUS}}$ mit $\models_\Im \{p\}\ S_1\ \{r\}$ und $\models_\Im \{r\}\ S_2\ \{q\}$.
- Existenz einer Zwischenbehauptung („intermittent assertion“)!

Dann nach Induktionsvoraussetzung $\vdash_\Im \{p\}\ S_1\ \{r\}$ *und* $\vdash_\Im \{r\}\ S_2\ \{q\}$ wegen $R_;$
$\vdash_\Im \{p\}\ S_1; S_2\ \{q\}$.

2.) $S \equiv \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S_1\ \underline{\texttt{else}}\ S_2\ \underline{\texttt{fi}}$
Sei $\vdash_\Im \{p\}\ S\ \{q\}$. D.h., wenn $\text{wert}_\Im(p,\sigma) = W$, so $\text{wert}_\Im(q,\sigma') = W$,
wobei $\sigma' = \text{val}_\Im(S,\sigma) = \begin{cases} \text{val}_\Im(S_1,\sigma),\ \textit{falls}\ \text{wert}_\Im(b,\sigma) = W \\ \text{val}_\Im(S_2,\sigma),\ \textit{falls}\ \text{wert}_\Im(b,\sigma) = F \end{cases}$
Also, wenn $\text{wert}_\Im(p \wedge b, \sigma) = W$, so $\text{wert}_\Im(q,\sigma') = W$ und $\text{wert}_\Im(q, \text{val}_\Im(S_1,\sigma)) = W$,
d.h. $\models_\Im \{p \wedge b\}\ S_1\ \{q\}$. Wenn aber $\text{wert}_\Im(p \wedge \neg b, \sigma) = W$, so $\text{wert}_\Im(q, \text{val}_\Im(S_2,\sigma) = W$, also $\models_\Im \{p \wedge \neg b\}\ S_2\ \{q\}$. Nach Induktionsvoraussetzung gilt also :
$\vdash \{p \wedge b\}\ S_1\ \{q\}$ und $\vdash \{p \wedge \neg b\}\ S_2\ \{q\}$. Nach R_{if} gilt $\vdash \{p\}\ S\ \{q\}$. Damit ist der Beweis erbracht, falls die mit ! gekennzeichnete Aussage der Existenz einer Zwischenbehauptung bewiesen werden kann. (Siehe dazu im folgenden Abschnitt Lemma 3.8 und Folgerung).

q.e.d.

3.2 Schwächste Vorbedingung, stärkste Nachbedingung, Ausdrucksfähigkeit

$$\models_{\Im} \quad \{p\}\ S\ \{q\}$$

Nach Definition der partiellen Korrektheit kann für jedes Programm S stets `true` als richtige Nachbedingung, `false` als richtige Vorbedingung gewählt werden (das ist eine stets korrekte Spezifikation). Zur Beschreibung der Wirkung eines Programms S sind aber schwächere Vorbedingungen und stärkere Nachbedingungen nötig.

Beispiel 11

$\models_{\Im} \quad \{x = 0\}\ x := x+1\ \{x = 1\}$ Hier ist ohne Zweifel zur Vorbedingung die bestmögliche Nachbedingung angegeben und umgekehrt. Im Gegensatz dazu :
$\models_{\Im} \quad \{x = 0\}\ x := x+1\ \{x > 0\}$ ist nicht bestmögliche Nachbedingung zu gegebener Vorbedingung. Es ist nicht offensichtlich, was beste Nachbedingung zu gegebener Vorbedingung ist :

$$\{\texttt{true}\}\ x := x+1\ \{?\}$$

Vergleiche : $\models_{\Im} \quad \{\texttt{true}\}\ x := 1\ \{x = 1\}$, aber $\{\texttt{true}\}\ x := x+1\ \{x = x+1\}$.

Definition 3.3

Bei gegebener Interpretation $\Im$ heißt $p \in$ **AUS** eine *schwächste Vorbedingung*, (q eine *stärkste Nachbedingung*) zu dem Programm S bezüglich der Nachbedingung q (der Vorbedingung p), kurz $p = \mathrm{wp}_{\Im}(S,q)$ bzw. $q = \mathrm{sp}_{\Im}(S,p)$, genau dann, wenn

a) $\models_{\Im} \{p\}\ S\ \{q\}$ und

b) Für alle $r \in$ **AUS** gilt:

- wenn $\models_{\Im} \{r\}\ S\ \{q\}$, *so* $\models_{\Im} r \rightarrow p$
- (wenn $\models_{\Im} \{p\}\ S\ \{r\}$, *so* $\models_{\Im} q \rightarrow r$)

$\left(\begin{array}{l} \mathrm{wp}_{\Im} \ldots \textit{ weakest precondition} \\ \mathrm{sp}_{\Im} \ldots \textit{ strongest postcondition} \end{array}\right)$.

Bemerkung

Zu festem gegebenen S stellen wp und sp Funktionen dar, die Ausdrücken (der Prädikatenlogik) wieder Ausdrücke zuordnen. – (DIJKSTRA'sche Prädikattransformer)
Die Menge aller Paare (p,q) beschreibt die Wirkung des Programms S . – „axiomatische Semantik“

Satz 3.3

Es seien $\Im$ *eine Interpretation* $p, q \in$ **AUS**, $S \in$ **PR**, *so daß* $\mathrm{wp}_{\Im}(S,q) \in$ **AUS** *und* $\mathrm{sp}_{\Im}(S,p) \in$ **AUS**.

- $\models_{\Im} \{p\}\ S\ \{q\}$ *genau dann, wenn*
- $\models_{\Im} p \rightarrow \mathrm{wp}_{\Im}(S,q)$ *genau dann, wenn*

- $\models_{\Im}\ \ \mathrm{sp}_{\Im}(S,p)\ \rightarrow\ q$

Der **Beweis** ist eine gute Übung ! (Umsetzung der Definitionen)

Definition 3.4

Man sagt, ein Ausdruck p *stellt* eine Menge **B** von Zuständen *dar* bei der Interpretation $\Im$ genau dann, wenn

$$\mathrm{wert}_{\Im}(p,\sigma)\ =\ W \qquad \textit{genaudann, wenn} \qquad \sigma\ \in\ \mathbf{B}.$$

Bemerkung

Oft werden Bedingungen an die Ein-(Ausgangs-)variablen eines Programmes direkt durch eine Zustandsmenge B beschrieben. Deshalb werden Zustandsmengen oft direkt als „Prädikat“ bezeichnet. Im allgemeinen können aber nur gewisse Zustandsmengen durch Ausdrücke einer vorgegebenen Sprache **AUS** beschrieben werden, das hängt im besonderen auch von der Interpretation ab!

Speziell wird definiert:

Definition 3.5

$$\begin{aligned}\underline{\mathrm{pre}}_{\Im}(S,q)\ &=\ \{\sigma\ |\forall\ \rho(\mathrm{val}_{\Im}(S,\sigma)\ =\ \rho\ \rightarrow\ \mathrm{wert}_{\Im}(q,\rho)\ =\ W\}\\ \underline{\mathrm{post}}_{\Im}(p,S)\ &=\ \{\sigma\ |\exists\ \rho(\mathrm{wert}_{\Im}(p,\rho)\ =\ W\ \wedge\ \mathrm{val}_{\Im}(S,\rho)\ =\ \sigma)\}\end{aligned}$$

Das ist die semantische Beschreibung der oben eingeführten Begriffe „schwächste Vorbedingung“ bzw. „stärkste Nachbedingung“.

Satz 3.4

$\models_{\Im}\ \ \{p\}\ S\ \{q\}$ *genau dann, wenn*

$$\textit{a)}\ \ \{\sigma\ |\,\mathrm{wert}_{\Im}(p,\sigma)\ =\ W\}\ \subseteq\ \underline{\mathrm{pre}}_{\Im}(S,q)\ .$$

$$\textit{b)}\ \ \underline{\mathrm{post}}_{\Im}(p,S)\ \subseteq\ \{\sigma|\,\mathrm{wert}_{\Im}(q,\sigma)\ =\ W\}\ .$$

Beweis

a) Es sei $\mathbf{P}\ =\ \{\sigma\ |\,\mathrm{wert}_{\Im}(\mathbf{p},\sigma)\ =\ \mathbf{W}\}$

($\longleftarrow$) $\mathbf{P}\ \subseteq\ \underline{\mathrm{pre}}_{\Im}(\mathbf{S},\mathbf{q})$. Sei $\mathrm{wert}_{\Im}(p,\sigma)\ =\ W\ \rightarrow\ \sigma\ \in\ \mathbf{P}\ \rightarrow\ \sigma\ \in\ \underline{\mathrm{pre}}_{\Im}(\mathbf{S},\mathbf{q})$. Nach Definition von $\underline{\mathrm{pre}}_{\Im}$ gilt, wenn $\mathrm{val}_{\Im}(S,\sigma)\ =\ \rho$, so $\mathrm{wert}_{\Im}(q,\rho)\ =\ W$, d.h. $\models_{\Im}\ \ \{p\}\ S\ \{q\}$.

($\longrightarrow$) Sei $\models_{\Im}\ \ \{p\}\ S\ \{q\}$. Sei $\sigma\ \in\ \mathbf{P}$, d.h. $\mathrm{wert}_{\Im}(p,\sigma)\ =\ W$, wegen Spezifikation ist dann $\mathrm{wert}_{\Im}(q,\underbrace{\mathrm{val}_{\Im}(S,\sigma)}_{\rho})\ =\ W\ \rightarrow\ \sigma\ \in\ \underline{\mathrm{pre}}_{\Im}(S,q)$. Also $\mathbf{P}\ \subseteq\ \underline{\mathrm{pre}}_{\Im}(\mathbf{S},\mathbf{q})$.

b) analog zu a) (Übung)

q.e.d.

Definition 3.6

Eine prädikatenlogische Sprache **AUS** heißt *ausdrucksfähig* (expressiv) bezüglich einer Interpretation $\Im$ und einer Menge von Programmen **PR** , kurz

$$\Im \in EXP(\mathbf{AUS}, \mathbf{PR}),$$

genau dann, wenn für alle Ausdrücke q und Programme S mit $q \in$ **AUS** und $S \in$ **PR** ein $p \in$ **AUS** existiert, so daß p die Menge $\underline{\text{pre}}_{\Im}(S, q)$ darstellt.

Bemerkung

1980 bewies E.R. Olderog, daß ein äquivalenter Begriff der Ausdrucksfähigkeit entsteht, wenn man in obiger Definition $\underline{\text{pre}}_{\Im}$ durch $\underline{\text{post}}_{\Im}(q, S)$ ersetzt. Die Äquivalenz gilt für alle im folgenden betrachteten Programmklassen **PR** .

Satz 3.5

Es sei $\Im \in EXP(\mathbf{AUS}, \mathbf{PR})$, $p, q \in$ **AUS** *und* $S \in$ **PR**.

- $\underline{\text{pre}}_{\Im}(S, q)$ *wird durch* $\text{wp}_{\Im}(S, q)$ *und*
- $\underline{\text{post}}_{\Im}(p, S)$ *wird durch* $\text{sp}_{\Im}(S, p)$ *dargestellt.*

Beweis

Man beachte die Voraussetzung, daß $\underline{\text{pre}}_{\Im}(S, q)$ *bzw.* $\underline{\text{post}}_{\Im}(p, S)$ überhaupt dargestellt werden können.

a) $p = \text{wp}_{\Im}(S, q)$. $\quad \underline{\text{pre}}_{\Im}(S, q) = \{\sigma \mid \forall\ \rho\ \text{val}_{\Im}(S, \sigma) = \rho \rightarrow \text{wert}_{\Im}(q, \rho) = W\}$
($\longleftarrow$) $\sigma \in \underline{\text{pre}}_{\Im}(S, q)$. Es sei r ein Ausdruck, der $\underline{\text{pre}}_{\Im}(S, q)$darstellt. $\rightarrow$ $\text{wert}_{\Im}(r, \sigma) = W$. Nach Definition von $\underline{\text{pre}}_{\Im}(S, q)$ und der Definition der partiellen Korrektheit gilt : $\models_{\Im} \{r\}\ S\ \{q\}$ (vgl. Satz oben)
Damit ist nach Voraussetzung (Definition von „$\text{wp}_{\Im}$“ (2)) $\models r \rightarrow p \rightarrow \text{wert}_{\Im}(p, \sigma) = W$.
($\longrightarrow$) Sei nun $\text{wert}_{\Im}(p, \sigma) = W$. Wegen Voraussetzung („$\text{wp}_{\Im}$“ (1)) $\models_{\Im} \{p\}\ S\ \{q\}$
D.h., wenn $\text{val}_{\Im}(S, \sigma) = \rho$, so $\text{wert}_{\Im}(q, \rho) = W$. Also $\sigma \in \underline{\text{pre}}_{\Im}(S, q)$.

b) für $q = \text{sp}_{\Im}(S, p)$ analog.

q.e.d.

Bemerkung

Allein aus der Existenz (in **AUS**) von $\text{wp}_{\Im}(S, q)$ zu beliebigen $S \in$ **PR** und beliebigen $q \in$ **AUS** läßt sich *nicht* schließen, daß $\Im \in EXP(\mathbf{AUS}, \mathbf{PR})$. Dazu sind zusätzliche Voraussetzungen an $\Im$ zu machen, die letztlich gerade die Ausdrucksfähigkeit von **AUS** bezüglich $\Im$ ausmachen. (vgl. später)

Satz 3.6

Für jede Interpretation $\Im$ *gilt :* $\Im \in EXP(\mathbf{AUS}, \mathbf{PR}_0)$,
d.h. die Sprache des Prädikatenkalküls ist für Geradeausprogramme bei jeder Interpretation ausdrucksfähig.

Beweis

Zu zeigen : für alle $S \in \mathbf{PR}_0$, alle $q \in \mathbf{AUS}$ existiert ein $p \in \mathbf{AUS}$, das $\underline{\text{pre}}_{\Im}(S,q)$ darstellt.
(Beweis induktiv über den Aufbau von S)
Im folgenden sei zu beliebigem $q \in \mathbf{AUS}$ stets $\mathbf{B_q} = \{\sigma \mid \text{wert}_{\Im}(\mathbf{q},\sigma) = \mathbf{W}\}$.

Induktionsanfang:

0.1) $S \equiv \underline{\texttt{skip}}$

$$\begin{aligned}\underline{\text{pre}}_{\Im}(S,q) &= \{\sigma \mid \forall\ \rho\ \text{val}_{\Im}(S,\sigma) = \rho \rightarrow \text{wert}_{\Im}(q,\rho) = W\ )\}\\ &= \{\sigma \mid \forall\ \rho\ \text{val}_{\Im}(\underline{\texttt{skip}},\rho) = \rho \rightarrow \text{wert}_{\Im}(q,\rho) = W\ )\}\\ &= \{\sigma \mid \forall\ \rho\ (\sigma = \rho \rightarrow \text{wert}_{\Im}(q,\rho) = W\ )\}\\ &= \{\sigma \mid \text{wert}_{\Im}(q,\sigma) = W\}\\ &= \mathbf{B_q}\end{aligned}$$

Also wird $\underline{\text{pre}}_{\Im}(S,q)$ durch q dargestellt. Nach vorigem Satz gilt also $q = \text{wp}_{\Im}(\underline{\texttt{skip}},q)$.

0.2) $S \equiv x := t$

$$\begin{aligned}\underline{\text{pre}}_{\Im}(S,q) &= \{\sigma \mid \forall\ \rho\ (\sigma' = \rho \rightarrow \text{wert}_{\Im}(q,\rho) = W\ )\}\\ &= \{\sigma \mid \text{wert}_{\Im}(q,\sigma') = W\}\ ,\end{aligned}$$

wobei $\sigma' = \text{val}_{\Im}(x := t,\sigma)$, d.h. $\sigma'(y) = \begin{cases}\text{wert}_{\Im}(t,\sigma) & \text{für } y = x\\ \sigma(y) & \text{für} y \neq x\end{cases}$

Nach dem schon früher zitierten Satz über Wertänderungen bei Substitution ist
$\{\sigma \mid \text{wert}_{\Im}(q,\sigma') = W\} = \{\sigma \mid \text{wert}_{\Im}(q^x/_t,\sigma) = W\} = \mathbf{B_q}^{x}/_{t}$.
Das heißt, $\underline{\text{pre}}_{\Im}(S,q)$ wird durch $q^x/_t$ dargestellt. Weiter ist $\text{wp}_{\Im}(x := t\ ,\ q) = q^x/_t$.

Induktionsvoraussetzung:
$\underline{\text{pre}}_{\Im}(S_i,q)$ sei durch $\text{wp}_{\Im}(S_i,q)$ ($\in \mathbf{AUS}$) darstellbar für $i = 1, 2$ und beliebige $q \in \mathbf{AUS}$, d.h.

$\sigma \in \underline{\text{pre}}_{\Im}(S_i,q)$ genau dann, wenn $\text{wert}_{\Im}(\text{wp}_{\Im}(S_i,q),\sigma) = W$ genau dann, wenn $\text{wert}_{\Im}(q,\text{val}_{\Im}(S_i,\sigma)) = W$

also für beliebige $q \in \mathbf{AUS}$ und $\sigma \in \mathbf{ST}$ gilt nach Induktionsvoraussetzung:

$$\text{wert}_{\Im}(q,\text{val}_{\Im}(S_i,\sigma)) = \text{wert}_{\Im}(\text{wp}_{\Im}(S_i,q),\sigma).$$

Induktionsschluß:

1.) $S \equiv S_1;S_2$
$\underline{\text{pre}}_{\Im}(S,q)$ wird dann durch $\text{wp}_{\Im}(S_1,\text{wp}_{\Im}(S_2,q))$ ($\in \mathbf{AUS}$) dargestellt, **denn:**

$$\begin{aligned}\underline{\text{pre}}_{\Im}(S_1;S_2,q) &= \{\sigma \mid \text{wert}_{\Im}(q,\text{val}_{\Im}(S_2,\text{val}_{\Im}(S_1,\sigma))) = W\ )\}\\ &= \{\sigma \mid \text{wert}_{\Im}(q,\text{val}_{\Im}(S_2,\sigma')) = W\ \} \quad \text{mit } \sigma' = \text{val}_{\Im}(S_1,\sigma)\\ &= \{\sigma \mid \text{wert}_{\Im}(\text{wp}_{\Im}(S_2,q),\sigma') = W\ \} \quad \text{nach Ind.vor.}\\ &= \{\sigma \mid \text{wert}_{\Im}(\text{wp}_{\Im}(S_2,q),\text{val}_{\Im}(S_1,\sigma)) = W\}\\ &= \{\sigma \mid \text{wert}_{\Im}(\text{wp}_{\Im}(S_1,\text{wp}_{\Im}(S_2,q)),\sigma) = W\} \quad \text{nach Ind.vor.}\end{aligned}$$

2.) $S \equiv \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S_1\ \underline{\texttt{else}}\ S_2\ \underline{\texttt{fi}}$
$\underline{\text{pre}}_{\Im}(S,q)$ wird dann durch

$$p \equiv \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ \text{wp}_{\Im}(S_1,q)\ \underline{\texttt{else}}\ \text{wp}_{\Im}(S_2,q)\ \underline{\texttt{fi}}$$

dargestellt, was als Abkürzung für den wertverlaufsgleichen Ausdruck

$$(b \wedge \text{wp}_{\Im}(S_1,q)) \vee (\neg b \wedge \text{wp}_{\Im}(S_2,q)) \quad \in \mathbf{AUS} \text{ (wegen } \text{wp}_{\Im}(S_i,q) \in \mathbf{AUS} \text{ nach Ind.vor.)}$$

steht. Denn es ist:

$$\underline{\mathrm{pre}}_{\Im}(S,q) = \{\sigma \mid \mathrm{wert}_{\Im}(q,\sigma') = W\} \qquad \text{mit } \sigma' = \mathrm{val}_{\Im}(S,\sigma),$$

$$\text{d.h. } \sigma' = \begin{cases} \mathrm{val}_{\Im}(S_1,\sigma), & \text{falls } \mathrm{wert}_{\Im}(b,\sigma) = W \\ \mathrm{val}_{\Im}(S_2,\sigma), & \text{falls } \mathrm{wert}_{\Im}(b,\sigma) = F \end{cases},$$

$$= \{\sigma \mid (\mathrm{wert}_{\Im}(b,\sigma) = W \wedge \mathrm{wert}_{\Im}(q,\mathrm{val}_{\Im}(S_1,\sigma)) = W) \\ \vee \ (\mathrm{wert}_{\Im}(\neg b,\sigma) = W \wedge \mathrm{wert}_{\Im}(q,\mathrm{val}_{\Im}(S_2,\sigma)) = W)\}.$$

$$= \{\sigma \mid (\mathrm{wert}_{\Im}(b,\sigma) = W \wedge \mathrm{wert}_{\Im}(\mathrm{wp}_{\Im}(S_1,q),\sigma) = W) \\ \vee \ (\mathrm{wert}_{\Im}(\neg b,\sigma) = W \wedge \mathrm{wert}_{\Im}(\mathrm{wp}_{\Im}(S_2,q),\sigma) = W)\}$$

$$= \{\sigma \mid (\mathrm{wert}_{\Im}((b \wedge \mathrm{wp}_{\Im}(S_1,q)) \vee (\neg b \wedge \mathrm{wp}_{\Im}(S_2,q)),\sigma) = W\}$$

q.e.d.

Aus den beiden vorangegangenen Sätzen ergibt sich die

Folgerung 3.7

- $\mathrm{wp}_{\Im}(\underline{\texttt{skip}}, q) = q$
- $\mathrm{wp}_{\Im}(x := t, q) = q^x/_t$
- $\mathrm{wp}_{\Im}(S_1; S_2, q) = \mathrm{wp}_{\Im}(S_1, \mathrm{wp}_{\Im}(S_2, q))$
- $\mathrm{wp}_{\Im}(\underline{\texttt{if}}\, b\, \underline{\texttt{then}} S_1\, \underline{\texttt{else}} S_2\, \underline{\texttt{fi}},\ q) = \underline{\texttt{if}}\, b\, \underline{\texttt{then}}\, \mathrm{wp}_{\Im}(S_1,q)\, \underline{\texttt{else}}\, \mathrm{wp}_{\Im}(S_2,q)\, \underline{\texttt{fi}}$

Lemma 3.8

(zur Existenz der Zwischenbehauptung)

Aus $\models_{\Im} \{p\}\, S_1; S_2\, \{q\}$ *folgt: es gibt ein* $r \in \underline{\mathbf{AUS}}$ *mit*

$$\models_{\Im} \{p\}\, S_1\, \{r\} \quad \textit{und} \quad \models_{\Im} \{r\}\, S_2\, \{q\}.$$

(Das ist die im Vollständigkeitsbeweis im vorigen Abschnitt gemachte und durch **!** gekennzeichnete Hilfsbehauptung.)

Beweis

Setze $r = \mathrm{wp}_{\Im}(S_2, q)$. Nach vorigem Satz ist dann $r \in \underline{\mathbf{AUS}}$. Entsprechend der Definition von $\mathrm{wp}_{\Im}$ gilt: $\models_{\Im} \{r\}\, S_2\, \{q\}$. Weiter gilt nach einem Satz weiter oben:

$\models_{\Im} \{p\}\, S_1; S_2\, \{q\}$	genau dann, wenn	$\models_{\Im} p \rightarrow \mathrm{wp}_{\Im}(S_1; S_2, q)$
genau dann, wenn (wegen obiger Folgerung)		$\models_{\Im} p \rightarrow \mathrm{wp}_{\Im}(S_1, \mathrm{wp}_{\Im}(S_2, q))$
genau dann, wenn (wegen selbem obigen Satz)		$\models_{\Im} \{p\}\, S_1\, \{\mathrm{wp}_{\Im}(S_2, q)\}$,

also haben wir auch $\models_{\Im} \{p\}\, S_1\, \{r\}$. q.e.d.

Bemerkung

Damit ist der Beweis des Vollständigkeitssatzes für H_0 lückenlos.

3.3 Das FLOYDsche Vorwärts–Zuweisungsaxiom

Das HOARE–Axiom (Z) für Ergibtanweisungen wirkt „rückwärts“: zu gegebener Nachbedingung gibt es die bestmögliche, d.h. schwächste Vorbedingung an. Häufig aber interessiert die umgekehrte Frage: gesucht ist die bestmögliche Nachbedingung zu gegebener Vorbedingung. Man möchte auch „vorwärts“ schließen können.

Satz 3.9

Es sei $p \in \underline{\mathbf{AUS}}$, $x \in \underline{\mathbf{V}}$, $t \in \underline{\mathbf{TERM}}$ *und* $\Im$ *eine Interpretation. Dann gilt:*

$$\mathrm{sp}_{\Im}(x := t,\ p) = \exists y(p^x/_y \wedge x = t^x/_y),$$

wobei $y \in \underline{\mathbf{V}}$ *mit* $y \not\equiv x$ *und* $y \notin var(p,t)$[1].

Beweis

Der Beweis wird entsprechend der Definition von $\mathrm{sp}_{\Im}$ geführt:

1. Zu zeigen:

$$\models_{\Im} \{p\}\, x := t\, \{\exists y(p^x/_y \wedge x = t^x/_y)\} \qquad \text{(F)}$$

- wird auch als FLOYD–Axiom bezeichnet.

Sei $\mathrm{wert}_{\Im}(p,\sigma) = W$ und $\sigma' = \mathrm{val}_{\Im}(x := t, \sigma)$.
Es ist $\mathrm{wert}_{\Im}(\exists y(p^x/_y \wedge x = t^x/_y),\ \sigma') = W$ genau dann, wenn es eine y–Variation σ'_y von σ' gibt mit $\mathrm{wert}_{\Im}(\, p^x/_y \wedge x = t^x/_y,\ \sigma') = W$.
Betrachte nun $\sigma'_y(z) = \begin{cases} \sigma(x) & \text{für } z = y \\ \sigma'(z) & \text{für } z \neq y \end{cases}$,
das heißt nach Definition von σ':

$$\sigma'_y(z) = \begin{cases} \mathrm{wert}_{\Im}(t,\sigma) & \text{für } z = x \\ \sigma(x) & \text{für } z = y \\ \sigma(z) & \text{für } z \neq x \text{ und } z \neq y \end{cases}.$$

Damit ergibt sich

$$\begin{array}{lll} \mathrm{wert}_{\Im}(\, p^x/_y & \wedge & x = t^x/_y,\ \sigma'_y) \\ = et(\mathrm{wert}_{\Im}(p^x/_y, \sigma'_y) & , & \mathrm{wert}_{\Im}(x = t^x/_y, \sigma'_y)) \\ = et(\mathrm{wert}_{\Im}(p, \sigma'^{*}_y) & , & [\,\sigma'_y(x) = \mathrm{wert}_{\Im}(t, \sigma'^{*}_y)\,]\,), \end{array}$$

wobei nach dem schon öfter gebrauchten Satz über Wertänderung bei Substitution

$$\sigma'^{*}_y(z) = \begin{cases} \sigma'_y(y) & \text{für } z = x \\ \sigma'_y(z) & \text{für } z \neq x \end{cases},$$

[1] $var(p,t)$ bezeichnet die Menge aller Variablen, die in p oder t vorkommen

das heißt

$$\sigma'^{*}_{y}(z) \;=\; \begin{cases} \sigma(x) & \text{für } z = x \\ \sigma(z) & \text{für } z \neq x \text{ und } z \neq y \\ \sigma(x) & \text{für } z = y \end{cases},$$

d.h. σ'^{*}_{y} stimmt mit σ überall - mit Ausnahme von y - überein. Da aber $y \notin var(p,t)$ ist, folgt

$$\begin{array}{lll} \mathrm{wert}_{\Im}(\, p^x/_y & \wedge & x = t^x/_y,\, \sigma'_y) \\ = et(\, \mathrm{wert}_{\Im}(p,\sigma) & , & [\,\mathrm{wert}_{\Im}(t,\sigma) = \mathrm{wert}_{\Im}(t,\sigma)\,]\,) \\ = et(\, W & , & W) \;=\; W. \end{array}$$

Damit ist (F) gezeigt.

2. Zu zeigen:
Für alle $r \in$ **AUS** : wenn $\models_{\Im} \{p\}\, x := t\, \{r\}$, so $\models_{\Im} \exists y(p^x/_y \wedge x = t^x/_y) \rightarrow r$.
Sei für alle $\sigma \in$ **ST**:
$(*)$ wenn $\mathrm{wert}_{\Im}(p,\sigma) = W$, so $\mathrm{wert}_{\Im}(r,\sigma') = W$ mit $\sigma' = \mathrm{val}_{\Im}(x := t, \sigma)$.
Weiter sei $\mathrm{wert}_{\Im}(\exists y(p^x/_y \wedge x = t^x/_y), \sigma) = W$, d.h. es existiert eine y–Variation σ_y von σ mit $\mathrm{wert}_{\Im}(p^x/_y \wedge x = t^x/_y, \sigma_y) = W$, d.h. $\mathrm{wert}_{\Im}(p^x/_y, \sigma_y) = W$ und $\sigma_y(x) = \mathrm{wert}_{\Im}(t^x/_y, \sigma_y)$.
Ist σ^*_y die entsprechend der Einsetzung aus σ_y hervorgegangene abgeänderte Belegung, so folgt $\mathrm{wert}_{\Im}(p,\sigma^*_y) = W$ und $\sigma_y(x) = \mathrm{wert}_{\Im}(t,\sigma^*_y)$.
Nach Definition der Abänderung durch die Einsetzung ist $\sigma^*_y(x) = \sigma_y(y)$, und $\sigma^*_y(z) = \sigma_y(z)$ für $z \neq x$.
Damit ergibt sich für σ^* mit $\sigma^*(z) = \sigma^*_y(z)$ für $z \neq y$ und $\sigma^*(y) = \sigma(y)$ wegen der Voraussetzungen $y \notin var(p)$ und $y \notin var(t)$

$$a) \quad \mathrm{wert}_{\Im}(p,\sigma^*) = W \qquad \text{und} \qquad b) \quad \sigma_y(x) = \mathrm{wert}_{\Im}(t,\sigma^*).$$

Nach $(*)$ ist wegen $a)$ damit $\mathrm{wert}_{\Im}(r,\sigma'^*) = W$ mit $\sigma'^* = \mathrm{val}_{\Im}(x := t, \sigma^*)$.
Nun ist aber wegen $b)$ und der Definition einer y–Variation

$$\sigma'^*(x) = \mathrm{wert}_{\Im}(t,\sigma^*) = \sigma_y(x) = \sigma(x),$$

und für $z \neq x$ ist

$$\sigma'^*(z) = \sigma^*(z) = \begin{cases} \sigma^*_y(z) = \sigma_y(z) = \sigma(z) & \text{für } z \neq y \\ \sigma^*(y) = \sigma(y) & \text{für } z = y \end{cases},$$

das heißt $\sigma'^* = \sigma$, woraus $\mathrm{wert}_{\Im}(r,\sigma) = W$ folgt.
Damit ist die Gültigkeit der Implikation bewiesen.

q.e.d.

Beispiel 12

(für Anwendungen des HOARE- und des FLOYD–Axioms bei Ergibtanweisungen)

1. a)

$$\{\ ?\ \}\, x := x+1\, \{\, x = 1\,\}$$
$$\mathrm{wp}_{\Im}(x := x+1\,,\, x = 1) \;=\; (x=1)^x/_{x+1} \;=\; (x+1=1) \;=\; (x=0)$$
$$\{\, x = 0\,\}\, x := x+1\, \{\, x = 1\,\}.$$

Umgekehrt: b)

$$\{\, x = 0\,\}\, x := x+1\, \{\ ?\ \}$$
$$\begin{array}{rl} \mathrm{sp}_{\Im}(x := x+1\,,\, x = 0) & = \exists y\,(\,(x=0)^x/_y \wedge x = (x+1)^x/_y\,) \\ & = \exists y\,(\,y = 0 \wedge x = y+1\,) \\ & = \exists y\,(\,y = 0 \wedge x = 1\,) \\ & = \exists y\,(\,y = 0\,) \wedge (\,x = 1\,) \\ & = \underline{\underline{(\,x = 1\,)}}. \end{array}$$

2.

$$\{ \quad \underline{\texttt{true}} \quad \} \, x := x+1 \, \{ \quad ? \quad \}$$

(vgl. Motivierung zum Begriff der stärksten Nachbedingung $sp_{\Im}$ am Anfang des vorigen Abschnitts)

$sp_{\Im}(x := x+1 \,,\, \underline{\texttt{true}}) \;\; = \; \exists y \, (\, \underline{\texttt{true}} \, \wedge \, x \, = \, (x+1)^{x}/_{y} \,)$
$= \; \exists y \, (\, x \, = \, y+1 \,)$
bei Interpretationen $\Im_N$ über natürlichen Zahlen (Standardinterpretationen) gleichbedeutend mit
$= \; (\, x \, > \, 0 \,)$
bei Interpretationen $\Im$ in anderen Zahlbereichen:
$= \; \exists z \, (\, x \, = \, z \,) \; = \; \underline{\texttt{true}}.$

Damit haben wir entweder

$$\models_{\Im_N} \{ \quad \underline{\texttt{true}} \quad \} \, x := x+1 \, \{ \quad x > 0 \quad \}$$

als in diesem Fall bestmögliche Nachbedingung, oder im allgemeinen Fall

$$\models_{\Im} \{ \quad \underline{\texttt{true}} \quad \} \, x := x+1 \, \{ \quad \underline{\texttt{true}} \quad \},$$

das heißt die „bestmöglich“ anzugebende Nachbedingung ist eben die, die nichts einschränkt, umgangssprachlich also eigentlich „keine“ (einschränkende) Bedingung.

3.4 `while` –Programme (Iteration)

Definition 3.7

PR sei die kleinste Menge von Zeichenreihen, für die gilt :
0), 1) und 2) wie in der Definition von $\mathbf{PR}_0$, nur jetzt mit **PR** statt $\mathbf{PR}_0$, und dazu

3) mit $b \in$ **BAUS** und $S \in$ **PR** ist auch

$$\underline{\texttt{while}} \, b \, \underline{\texttt{do}} \, S \, \underline{\texttt{od}} \quad \in \mathbf{PR}.$$

Die Elemente von **PR** heißen *iterative* oder `while` *–Programme.*

Beachte: $\mathbf{PR}_0 \subseteq \mathbf{PR}$.

Beispiel 13

Beispielprogramme und zugehörige gültige Korrektheitsformeln
Die zugrunde gelegte Theorie T_0 möge die Arithmetik der ganzen Zahlen (mit $+$, $\cdot$, -, $\div$, *mod*) sein, $\Im_0$ sei Standardmodell von T_0.

1.) S_1 : `a := 0; b := x;` `while` `b ≥ y` `do` `b := b - y; a := a + 1` `od`

$$\models_{\Im_0} \{ x \geq 0 \, \wedge \, y \geq 0 \} \quad S_1 \quad \{ a \; = \; x \; \div \; y \, \wedge \, b \; = \; x \, mod \, y \}$$

2.) S_2 : `while x > 0 do x := x - 1 od`

$$\models_{\Im_0} \{ x = 5 \} \quad S_2 \quad \{ x \; = \; 0 \}$$

3.) S_3 : `a := x; b := y; z := 1; while b ≠ 0 do b := b - 1; z := z · a od`

$$\models_{\Im_0} \{ x \geq 0 \wedge y \geq 0 \} \quad S_3 \quad \{ z = x^y \}$$

4.) S_4 : `while x > 0 do x := x + 1 od`

$$\models_{\Im_0} \{ x = 5 \} \quad S_4 \quad \{ y = 3 \}$$

5.) für beliebige $S \in$ **PR**, $p, q \in$ **AUS** und beliebige Interpretationen $\Im$:

$$\models_{\Im} \{ p \} \quad \underline{\texttt{while}}\ \texttt{true}\ \underline{\texttt{do}}\ S\ \underline{\texttt{od}} \quad \{ q \}$$

Unter der Eingangsbedingung p betrachten wir die folgende `while` -Anweisung mit der üblichen intuitiven Bedeutung. An den verschiedenen Stellen der Anweisung fügen wir die dort zutreffenden Bedingungen („assertions“) ein, um uns auf diese Weise eine Beweisregel für die `while` -Anweisung plausibel zu machen.

$\{ p \}$
`while` b
`do`
$\{ p \wedge b \}$
S
$\{ p \}$
`od`
$\{ p \wedge \neg b \}$

Damit erhalten wir folgende

Beweisregel für Iterationen

$$R_{wh} : \qquad \frac{\{ p \wedge b \}\ S\ \{ p \}}{\{ p \}\ \underline{\texttt{while}}\ b\ \underline{\texttt{do}}\ S\ \underline{\texttt{od}}\ \{ p \wedge \neg b \}}$$

Die Bedingung p heißt dabei **Schleifeninvariante**.

Bezeichnungen

$R_{SP} = R^0_{SP} \cup \{ R_{wh} \}$ Menge der Schlußregeln für `while` -Programme.

$H = (A_{SP}, R_{SP})$ Beweissystem für **PR**

Beispiel 14

Beispiel für einen Beweis in H (vgl. das 1. Beispielprogramm im obigen Beispiel 13) :

$$S_1 : a := 0;\ b := x;\ \underline{\texttt{while}}\ b \geq y\ \underline{\texttt{do}}\ b := b - y;\ a := a + 1\ \underline{\texttt{od}}$$

Zu zeigen:

$$\vdash_H \{ x \geq 0 \wedge y \geq 0 \}\, S_1 \,\{ a = x \div y \wedge b = x \,\underline{mod}\, y \}$$

ist äquivalent zu

$$\vdash_H \{ x \geq 0 \wedge y \geq 0 \}\, S_1 \,\{ a \cdot y + b = x \wedge 0 \leq b \leq y \} \qquad (*)$$

(*)

$(R_;)$

(1) $\{\, a \cdot y + b = x \wedge 0 \leq b \,\}$ $\underline{\texttt{while}}\; b \geq y \;\underline{\texttt{do}}\; b := b - y;\; a := a + 1 \;\underline{\texttt{od}}$ $\{\, a \cdot y + b = x \wedge 0 \leq b \leq y \,\}$

(2) $\{\, x \geq 0 \wedge y \geq 0 \,\}\, a := 0;\, b := x \,\{\, a \cdot y + b = x \wedge 0 \leq b \,\}$

(1)

(R_{wh})

(3) $\{\, a \cdot y + b = x \wedge 0 \leq b \wedge b \geq y \,\}$ $b := b - y;\; a := a + 1$ $\{\, a \cdot y + b = x \wedge 0 \leq b \,\}$

(2)

$(R_;)$

(4) $\underbrace{\{\, a \cdot y + x = x \wedge x \geq 0 \,\}}_{a = 0 \wedge x \geq 0}$ $b := x \,\{\, a \cdot y + b = x \wedge 0 \leq b \,\}$ (wegen (Z))

(5) $\{\, x \geq 0 \wedge y \geq 0 \,\}\, a := 0 \,\{\, a = 0 \wedge x \geq 0 \,\}$

(5): es gilt $\{\, 0 = 0 \wedge x \geq 0 \,\}\, a := 0 \,\{\, a = 0 \wedge x \geq 0 \,\}$ (wegen (Z))
und (5) dann wegen $x \geq 0 \wedge y \geq 0 \rightarrow x \geq 0$ und $R_\rightarrow$.

(3)

$(R_;)$

(6) $\{\, (a+1) \cdot y + b = x \wedge b \geq 0 \,\}$ $a := a + 1 \,\{\, a \cdot y + b = x \wedge b \geq 0 \,\}$ (wegen (Z))

(7) $\{\, a \cdot y + b = x \wedge b \geq 0 \wedge b \geq y \,\}\, b := b - y$ $\{\, (a+1) \cdot y + b = x \wedge b \geq 0 \,\}$

wegen (Z) gilt

$$\{\, (a+1) \cdot y + b - y = x \wedge b - y \geq 0 \,\}\, b := b - y \,\{\, (a+1) \cdot y + b = x \wedge b \geq 0 \,\}$$

und es gilt

$$a \cdot y + b = x \wedge b \geq 0 \wedge b \geq y \rightarrow \underbrace{(a+1) \cdot y + b - y = x \wedge b - y \geq 0}_{a \cdot y + b = x \wedge b \geq 0}$$

und damit wegen $R_\rightarrow$ auch (7).

Bemerkung

a) Dies ist ein Beweis in formalisierten Theorien $T_{\Im}$, für die die verwendeten Implikationen zu $\underline{\mathbf{TR}}_{\Im}$ gehören. $\Im$ kann also irgendeine Standardinterpretation der Arithmetik der ganzen oder auch der reellen Zahlen sein.

b) Die Termination von S_1 ist damit **nicht** gezeigt. (Bei der Betrachtung der partiellen Korrektheit generell nicht!) Beachte zum Beispiel: Für $y = 0$ stimmt die Nachbedingung nicht, S_1 terminiert aber auch nicht.

c) Die verwendete Schleifeninvariante p war hier

$$a \cdot y + b = x \wedge b \geq 0.$$

Es bedarf einer gewissen Übung, diese Invariante aufzustellen. Dazu später mehr.

d) Mit der `while` -Konstruktion (Iteration) kommen neuartige Probleme in die Sprache:

- Frage der Termination, hier erstmalig

$$\mathrm{val}_{\Im} : \underline{\mathbf{PR}} \times \underline{\mathbf{ST}} \rightharpoonup \underline{\mathbf{ST}}$$

echt partiell zu definieren!

- erst für `while` -Programme unterscheiden sich die partielle von der totalen Korrektheit.

3.4.1 Zur Semantik der `while` -Konstruktion

Wir gehen aus von der intuitiven Vorstellung der Bedeutung von

$$P \equiv \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}.$$

Für einen gegebenen Zustand σ sind zwei Fälle möglich:

a) entweder nach n-maliger Iteration ($n \geq 0$), das heißt wiederholter Ausführung von S wird ein σ' erreicht, so daß $\mathrm{wert}_{\Im}(b, \sigma') = F$ ist: dann ist σ' der „Nachfolgezustand“ von P: $\mathrm{val}_{\Im}(P, \sigma) = \sigma'$

b) oder: es gibt kein solches n, dann ist $\mathrm{val}_{\Im}(P, \sigma)$ nicht definiert.

Es bezeichne `loop` ein solches Programm, das für kein σ einen Folgezustand liefert:

$$\mathrm{val}_{\Im}(\texttt{loop}, \sigma) \quad \text{nicht definiert}$$

für alle $\sigma \in \underline{\mathbf{ST}}$ bei allen Interpretationen $\Im$.

z.B.: `loop` Abkürzung für:

`while true do skip od`

.

Nun betrachte die folgende Folge von Programmen:

$$P_0 \equiv \texttt{loop}$$

$$P_{n+1} \equiv \texttt{if}\ b\ \texttt{then}\ S; P_n\ \texttt{else skip fi} \quad \text{für} \qquad n \geq 0$$

Beachte: P semantisch äquivalent zu:

$$\texttt{if}\ b\ \texttt{then}\ S;\ P\ \texttt{else skip fi}$$

Beispiel 15

$$P_3 \equiv \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S;\ \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S;\ \underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S;\ \underline{\texttt{loop}}\ \underline{\texttt{else}}\ \underline{\texttt{skip}}\ \underline{\texttt{fi}}\ \underline{\texttt{else}}\ \underline{\texttt{skip}}\ \underline{\texttt{fi}}\ \underline{\texttt{else}}\ \underline{\texttt{skip}}\ \underline{\texttt{fi}}$$

betrachten beliebige $\sigma \in \underline{\mathbf{ST}}$: wenn P bei σ höchstens 2 Iterationen ausführt, dann kann P auch durch P_3 beschrieben werden, d. h. bei 0,1 oder 2 Iterationen bzw. bei Wertefolgen für b :
$\text{wert}_{\mathfrak{I}}(b, \sigma)$, $\text{wert}_{\mathfrak{I}}(b, \text{val}_{\mathfrak{I}}(S, \sigma)), \ldots$

	F
entsprechend:	W,F
	W,W,F

allgemein:
P_{n+1} ist mit P äquivalent für solche Zustände, bei denen P höchstens n Iterationen ausführt. (Für Zustände σ, für die P mehr als n-mal iteriert, ist $\text{val}_{\mathfrak{I}}(P_n, \sigma)$ nicht definiert.)
P_0 ist mit P äquivalent für solche Zustände, für die P nicht terminiert.

Statt einer **Definition** (der Semantik) formulieren wir nun folgenden

Fakt
Es sei P das Programm

$$\underline{\texttt{while}}\ b\ \underline{\texttt{do}}\ S\ \underline{\texttt{od}},$$

P_0 das Programm $\underline{\texttt{loop}}$ und P_{n+1} das Programm $\underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S; P_n\ \underline{\texttt{else}}\ \underline{\texttt{skip}}\ \underline{\texttt{fi}}$ für $n \geq 0$.
Dann gilt für alle $\sigma \in \underline{\mathbf{ST}}$:

a) Es gibt ein $n \geq 0$, so daß $\text{val}_{\mathfrak{I}}(P, \sigma) = \text{val}_{\mathfrak{I}}(P_n, \sigma)$.

b) Ist $\text{wert}_{\mathfrak{I}}(b, \sigma) = F$, so ist $\text{val}_{\mathfrak{I}}(P, \sigma) = \sigma$.

c) $\text{val}_{\mathfrak{I}}(P, \sigma) = \text{val}_{\mathfrak{I}}(\underline{\texttt{if}}\ b\ \underline{\texttt{then}}\ S; P\ \underline{\texttt{else}}\ \underline{\texttt{skip}}\ \underline{\texttt{fi}}, \sigma)$.

Bemerkung

a) $\text{val}_{\mathfrak{I}}$ ist eine partielle Funktion (vgl. Fakt 1, für $n = 0$).
Fakt 3 ist zu lesen: Die rechte Seite ist genau dann definiert, wenn die linke Seite definiert ist, und im Falle der Definiertheit haben wir eine Fixpunktgleichung für $\text{val}_{\mathfrak{I}}(P, \sigma)$.

b) Das gemäß Fakt 1 existierende n ist nicht eindeutig bestimmt, aber es gilt:
Für alle σ, für die $\text{val}_{\mathfrak{I}}(P_n, \sigma)$ definiert ist, gilt:

$$\text{val}_{\mathfrak{I}}(P_n, \sigma) = \text{val}_{\mathfrak{I}}(P_{n+1}, \sigma).$$

Der **Beweis** sei als Übungsaufgabe empfohlen.

Zur Semantik von $\underline{\mathbf{PR}}$

$\text{val}_{\mathfrak{I}}(S, \sigma)$ wird für $S \in \underline{\mathbf{PR}}$ über den Aufbau von S erklärt, wobei die bisherige Definition für $S \in \underline{\mathbf{PR}}_0$ zu übertragen ist.
Im Induktionsschritt 1. und 2. ist $\text{val}_{\mathfrak{I}}(S, \sigma)$ genauso erklärt wie bisher, und damit definiert, wenn $\text{val}_{\mathfrak{I}}(S_i, \sigma)$ für die Teilprogramme S_i definiert ist, sonst nicht.
Der Induktionsschritt 3. ist durch obigen Fakt zu ersetzen.
Damit ist $\text{val}_{\mathfrak{I}}(S, \sigma)$ im allgemeinen nicht explizit festgelegt, aber es sind die von uns benötigten Eigenschaften bekannt.

3.4.2 Zur Widerspruchsfreiheit von H

Satz 3.10

Die Beweisregel R_{wh} für iterative Programme aus **PR** *ist korrekt bei jeder Interpretation $\Im$.*

Beweis

Es sei $\Im$ eine beliebige Interpretation, $p \in$ **AUS**, $b \in$ **BAUS**, $S \in$ **PR**.
Sei $\models_\Im \{ p \wedge b \} S \{ p \}$, d.h.
(V) wenn $\mathrm{val}_\Im(S,\sigma)$ definiert ist und $\mathrm{val}_\Im(S,\sigma) = \rho$ und $\mathrm{wert}_\Im(p \wedge b, \sigma) = W$,
so ist $\mathrm{wert}_\Im(p,\rho) = W$ für beliebige $\sigma, \rho \in$ **ST**.

Die Bezeichnungen P, P_k $(k = 0, 1, \ldots)$ seien wie in Fakt gewählt.
Es gilt: Es gibt ein $k \geq 0$, so daß $\mathrm{val}_\Im(P,\sigma) = \mathrm{val}_\Im(P_k,\sigma)$ ist. Wir zeigen

$$(*) \qquad \models_\Im \{ p \} P_k \{ p \wedge \neg b \} \quad \text{für jedes } k \geq 0,$$

so daß daraus die Behauptung

$$\models_\Im \{ p \} P \{ p \wedge \neg b \}$$

folgt.

Sei nun $\mathrm{wert}_\Im(p,\sigma) = W$.

(Induktionsanfang)

$k = 0 : \mathrm{val}_\Im(P_0,\sigma) = \mathrm{val}_\Im(\underline{\texttt{loop}},\sigma)$ ist nicht definiert, damit $(*)$ trivialerweise gültig.

(Induktionsschritt)

$k = n + 1$

$$\sigma' = \mathrm{val}_\Im(P_{n+1},\sigma) = \begin{cases} \mathrm{val}_\Im(P_n, \mathrm{val}_\Im(S,\sigma)), & \text{wenn } \mathrm{wert}_\Im(b,\sigma) = W \\ \sigma & \text{sonst} \end{cases}$$

2 Fälle:

a) $\mathrm{wert}_\Im(b,\sigma) = W$.
$\sigma' = \mathrm{val}_\Im(P_n, \mathrm{val}_\Im(S,\sigma))$.
Nach **(V)** folgt $\mathrm{wert}_\Im(p,\rho) = W$. Für beliebige $\rho \in$ **ST** ist nach (Induktionsvoraussetzung):
wenn $\mathrm{wert}_\Im(p,\rho) = W$, so $\mathrm{wert}_\Im(p \wedge \neg b, \mathrm{val}_\Im(P_n,\rho)) = W$.

Also haben wir auch für $\rho = \mathrm{val}_\Im(P_n,\sigma)$
$\mathrm{wert}_\Im(p \wedge \neg b, \mathrm{val}_\Im(P_n,\rho)) = W$, d.h. $\mathrm{wert}_\Im(p \wedge \neg b\,,\, \sigma') = \mathrm{wert}_\Im(p \wedge \neg b, \mathrm{val}_\Im(P_{n+1},\sigma)) = W$.

b) $\mathrm{wert}_\Im(b,\sigma) = F$.
Dann ist $\sigma' = \sigma$. Daraus folgt $\mathrm{wert}_\Im(p \wedge \neg b, \mathrm{val}_\Im(P_{n+1},\sigma)) =$
$\mathrm{wert}_\Im(p \wedge \neg b, \sigma') = \mathrm{wert}_\Im(p \wedge \neg b, \sigma) =$
$\texttt{et}(\mathrm{wert}_\Im(p,\sigma), \texttt{non}(\mathrm{wert}_\Im(b,\sigma))) = \texttt{et}(W,W) = W$.

q.e.d.

Korollar 3.11

Das Beweissystem H ist korrekt bezüglich jeder Interpretation.

Beweis

Der Beweis für die Widerspruchsfreiheit von H_0 läßt sich auch für $S \in \underline{\mathbf{PR}}$ entsprechend der Erweiterung der Semantik übernehmen und ist durch obigen Beweis für die Korrektheit von R_{wh} zu ergänzen. q.e.d.

3.4.3 Zur Rolle der Schleifeninvarianten

Um eine Verifikation wie im Beispiel zu führen, ist eine Schleifeninvariante zu finden.

Da das Aufstellen der Schleifeninvariante sehr schwierig ist, fordern moderne Programmentwurfsmehtoden das Aufstellen der Invariante gleichzeitig mit dem Entwurf der Iterationsschleife.

Diese Invariante (und andere Bedingungen) werden zusammen mit dem Programm notiert („annotated program"). Damit wird die Verifikation des Programms unterstützt, eventuell sogar der automatische Beweis der Korrektheit des Programms, und gleichzeitig wird das Programm dokumentiert (in einer gewissen standardisierten Form).

Die Invariante p muß immer erfüllt sein, wenn die Testbedingung b ausgewertet wird. Daraus folgt: p muß Nachbedingung des Teilprogramms vor der Schleife sein und muß ebenfalls nach dem Durchlaufen des Schleifenrumpfs erfüllt sein.

Aus einer gegebenen Invarianten läßt sich oft das Programm entwickeln.

Beispiel 16

Beispiel für diese Methode

Berechnung der Potenz:

Spezifikation

$$\{\ y \geq 0\ \}\ S\ \{\ z = x^y\ \}\ .$$

Als Invariante p wird gewählt:

$$za^b = x^y$$

a, b neue Programmvariablen, in z wird ein Teil der Potenz von x^y berechnet, Korrektur durch Multiplikation mit a^b.

Diese Invariante p ist eine Verallgemeinerung der postcondition! p läßt sich leicht initialisieren:

$$a := x; b := y; z := 1$$

Also ergibt sich ein erster Entwurf eines „annotated program":

$$\{\ y \geq 0\ \}$$

$$a := x; b := y; z := 1$$

$$\{\ za^b = x^y\ \}$$

$$\underline{\texttt{while}}\ b \neq 0$$

$$\underline{\texttt{do}}\ \{\ za^b = x^y \wedge b \neq 0\ \}$$

$$S_1$$

$$\underline{\texttt{od}}\ \{\ za^b = x^y \wedge b = 0\ \} \longrightarrow \{\ z = x^y\ \}$$

In S_1 ist also b zu verringern, um von $b \neq 0$ auf $b = 0$ zu kommen, beim Verringern um 1, muß z mit a multipliziert werden, um die Invariante zu erhalten. Also S_1:

$$b := b - 1; z := z * a$$

Damit ist S:

$$a := x; b := y; z := 1; \underline{\texttt{while}}\ b \neq 0\ \underline{\texttt{do}}\ b := b - 1; z := z * a\ \underline{\texttt{od}}$$

Hier sind y Schleifendurchläufe nötig. Kann man b „in größeren Schritten“ verringern, um ein effektiveres Programm zu erhalten?

p: $\quad z \cdot a^b = x^y$

$$a := x;\ b := y;\ z := 1;$$
$$\underline{\texttt{while}}\ b \neq 0\ \underline{\texttt{do}}\ b := b - 1;\ z := z * a\ \underline{\texttt{od}}$$

Hierbei sind y Schleifendurchläufe nötig. Wenn b in größeren Schritten verringert werden könnte, so wäre ein effektiveres Programm möglich:

b halbieren, wenn b gerade; sonst um 1 vermindern.

Bei Halbierung : $\quad b := b/2$
Invariante p wiederherstellen:

$$z \cdot a^b = x^y$$
$$z \cdot a^{\frac{b}{2}} \cdot a^{\frac{b}{2}} = x^y$$
$$= z \cdot (a \cdot a)^{\frac{b}{2}}$$

Also ergibt sich als **neues Programm**

$$\{ x \geq 0 \wedge y \geq 0 \}$$
$$a := x;\ b := y;\ z := 1$$
$$\underline{\texttt{while}}\ b \neq 0\ \underline{\texttt{do}}\ \{ z \cdot a^b = x^y \}$$
$$\underline{\texttt{while}}\ even(b)\ \underline{\texttt{do}}\ b := b/2;\ a := a * a\ \{ z \cdot a^b = x^y \}$$
$$\underline{\texttt{od}}\ ;$$
$$b := b - 1;\ z := z * a$$
$$\underline{\texttt{od}}\{ z \cdot a^b = x^y \wedge b = 0 \}$$
$$\{ z = x^y \}$$

Zusammenfassung des Entwickelns einer Schleife aus einer Invarianten p und gegebener Spezifikation $\{ q \}S\{ r \}$

1. Finde Teilprogramm zur Initialisierung von p.

2. Finde Bedingung b so, daß nach der Schleife gewünschte Postcondition r gilt :
$$\models p \wedge \neg b \rightarrow r$$

3. Finde do –Programm (Schleifenrumpf) so, daß p invariant bleibt, aber in endlich vielen Schritten von Bedingung b zu Bedingung $\neg b$ führt :
   - oft 2 Teile:
     a) ein „Schritt“ von b in Richtung $\neg b$
     b) „Wiederherstellen“ von p

Umgekehrt gibt es heuristische Methoden, die es in vielen Fällen erlauben, aus gegebenen Spezifikationen Invarianten zu entwickeln. Dies ist gut erläutert in :
FUTSCHEK, G., *Programmentwicklung und Verifikation.* Springerverlag Wien, New York 1989
Bevor wir zu diesen Methoden kommen, soll hier eine kleine Aufgabe eingeblendet werden, die uns zeigt, wie zweckmäßig manchmal das „Denken in Invarianten“ sein kann.

Aufgabe: „Kugelproblem“[2]
In einem Behälter befinden sich schwarze und und weiße Kugeln. Folgendes wird wiederholt, so lange es geht:

2 Kugeln werden herausgenommen (ohne hinzusehen).
Sind die Kugeln gleichfarbig, werden diese weggelegt und eine schwarze Kugel wird wieder hineingetan.
(Dafür sei ein hinreichend großer Vorrat an schwarzen Kugeln vorhanden.)

Sind die Kugeln verschiedenfarbig, so wird die schwarze weggelegt und die weiße kommt wieder in das Behältnis.

Offenbar endet dieser Prozeß, wenn nur noch 1 Kugel im Behältnis zurückgeblieben ist.

Die *Frage* ist nun, ob man auf Grund der Farbe der letzten übrigbleibenden Kugel irgendetwas über die Zahl der anfänglich vorhandenen schwarzen und weißen Kugeln aussagen kann?

Versuchen Sie, das Problem in 10 Minuten zu lösen!
Die Lösung finden Sie am Ende dieses Abschnitts.

Entwickeln von Invarianten aus gegebenen Spezifikationen

Im vorigen Abschnitt haben wir am Beispiel erläutert, daß es oft möglich ist in einfachen Teilschritten von einer gegebenen Invarianten zur Entwicklung einer `while` –Schleife zu kommen.
Damit stellt sich als eine *Kernfrage der Programmentwicklung*:

> Wie kommt man von einer gegebenen Spezifikation
> (Pre- und Postcondition) zu einer geeigneten Invarianten ?

Es gibt heuristische Standardmethoden zur Entwicklung von Invarianten. Hier sollen (wieder *am Beispiel*) **zwei Methoden** dargestellt werden, die beide davon ausgehen, daß

die Invariante eine Verallgemeinerung der Postcondition

sein muß.

Betrachte Spezifikation

$$\{\ q\ \}\ P\ \{\ r\ \}$$

P sei zu realisieren mit Hilfe einer `while` –Schleife

$$P\ \equiv\ S_{init};\ \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od}$$

Dabei bezeichne: S_{init} : Initialisierungsschritte

- Suche Invariante p:

$$\begin{array}{l}\{\ q\ \}\\ S_{init}\\ \{\ q\ \}\\ \texttt{while}\ \ b\\ \texttt{do}\ \ \{\ p\ \wedge\ b\ \}\\ S\\ \{\ p\ \}\\ \texttt{od}\\ \{\ p\ \wedge\ \neg b\ \}\\ \{\ r\ \}\end{array}$$

Es muß also gelten :

a) $\models\ p\ \wedge\ \ \neg b \rightarrow\ r$

[2] Aufgabe entnommen aus: GRIES,D., The Science of Programming, Springer-Verlag 1981

b) Es gibt eine (einfache) Initialisierung, so daß p vor der Schleife gilt.

- Bedingung p also so *schwach*, daß sie leicht initialisiert werden kann.
- Zum anderen p so stark, daß am Ende die Postcondition gesichert ist.

Zwei „Standardmethoden“ zur Abschwächung der Postcondition :

A. Methode „Weglassen einer Bedingung“:

Prinzip: Voraussetzung: r ist Konjunktion (von *mindestens* zwei Konjunktionsgliedern)

$$r = r_1 \wedge r_2$$

wähle: $p =_{df} r_1$ („Weglassen der Bedingung“ r_2)
$b =_{df} \neg r_2$

Kriterien für Wahl :

- lasse die Bedingung weg, die sich gut zum Bilden der Schleifenbedingung eignet (speziell keine Quantoren $\forall, \exists$; $b \in$ **BAUS** !) *Termination zu sichern !*
- die Bedingung(en) in p aufnehmen, die sich leicht initialisieren lassen

a) Betrachte Beispiel von früher (ganzzahlige Division mit Rest)[3]

$$\{ x \geq 0 \wedge y \geq 0 \} \quad P \quad \{ a * y + c = x \wedge 0 \leq c < y \}$$

Also Postcondition $r : a * y + c = x \wedge 0 \leq c \wedge c < y$.
Damit zunächst drei Möglichkeiten zum Weglassen :

(a) Weglassen von $0 \leq c$:
Dann Schleifenbedingung $b : c < 0$.
Zur Termination müßte c „vergrößert“ werden; Initialisierung von c mit irgendeiner negativen Zahl (aber welcher ??)

(b) Weglassen von $a * y + c = x$
Schleifenbedingung $b : a * y + c \neq x$.
Invariante $p : 0 \leq c < y$.
Damit ist wieder die Initialisierung unklar, Erreichen der Termination mit Hilfe b noch unklarer

(c) Weglassen von $c < y$
Damit Schleifenbedingung $b : c \geq y$
Invariante $p : a * y + c = x \wedge c \geq 0$.
Hier Initialisierung klar : $a := 0; c := x$
Terminationsverhalten auch klar : von $c \geq y$ zu $c < y$ zu kommen ist durch „Verringern“ von c möglich.

Also ist hier c) einzig sinnvolle Möglichkeit; Schleife nun leicht zu entwickeln aus p; vgl. auch Beispiel vorn.

b) Berechnung der ganzzahligen Näherung der Quadratwurzel

$$\{ x \geq 0 \} \quad P \quad \{ y \geq 0 \wedge y^2 \leq x < (y+1)^2 \}$$

$r : y \geq 0 \wedge y^2 \leq x \wedge x < (y+1)^2$

[3] wegen Bezeichnungskonfusion wurde hier eine Variable mit c statt b bezeichnet

a) Weglassen von $x < (y+1)^2$
Damit $b \;:\; x \geq (y+1)^2$ (sehr gut als Schleifenbedingung geeignet)
$p \;:\; y \geq 0 \wedge y^2 \leq x$
Initialisierung: $y := 0$;
zur Schleifenentwicklung :

$$\begin{array}{l} \underline{\texttt{while}}\; x \geq (y+1)^2 \\ \underline{\texttt{do}}\; \{\; p \;\wedge\; b\} \\ \quad \{\; y \geq 0 \wedge x \geq (y+1)^2 \;\} \qquad \text{(damit folgt auch } x \geq y^2\text{)} \\ \qquad\qquad S \\ \quad \{\; y \geq 0 \wedge x \geq y^2 \;\} \qquad \text{(das ist } p\text{)} \\ \underline{\texttt{od}} \\ \quad \{\; y \geq 0 \wedge y^2 \leq x < (y+1)^2 \;\} \end{array}$$

In S muß man y vergrößern (*Termination*). Bei zu großen Schritten läuft man Gefahr, die Invariante zu verletzen.
Wähle deshalb $S \;:\; y := y+1$.
Dann haben wir wegen Axiom (Z)

$$\{\; y+1 \geq 0 \wedge x \geq (y+1)^2 \;\} \quad y := y+1 \quad \{\; y \geq 0 \wedge x \geq y^2 \;\}$$

und wegen der Initialisierung $y := 0$ und der Schleifenanweisung $y := y+1$ generell $y \geq 0$, so daß dies genügt.

b) Weglassen von $y^2 \leq x$
Dies führt zu einer anderen Invarianten und damit dann auch zu einem anderen Programm.
Die Ausführung sei als **Übungsaufgabe** empfohlen!

B. Methode „Konstante durch Variable ersetzen“:

Prinzip: Voraussetzung: in r kommt Konstante k vor : $r = r(k)$
wähle Variable x mit Wertebereich Wb, so daß $k \in Wb$ und setze :
$p =_{df} r(x) \wedge x \in Wb$ ($x \in Wb \;\hat{=}$ Angabe eines Wertebereiches für x)
$b =_{df} x \neq k$
Dann klar : $\models p \wedge \neg b \rightarrow r(k)$

Beispiel 17

1.) $r :\; m = max\,\{a_i \mid 0 \leq i \leq 10\}$
Konstanten: 0, 10;
ersetze z.B. 10 durch n mit $0 \leq n \leq 10$ und bilde :

$$p :\; m = max\,\{a_i \mid 0 \leq i \leq n\} \wedge 0 \leq n \leq 10$$

$$b :\; n \neq 10$$

2.) Programm zur Berechnung des Skalarproduktes von $a = (a_1, a_2, \ldots, a_N)$ und $c = (c_1, c_2, \ldots, c_N)$

$$\{N \geq 0\}\; P\; \{\underbrace{s = \sum_{i=1}^{N} a_i \cdot c_i}_{r}\}$$

(Konstanten : 1 und N)
ersetze Konstante N durch Variable $n \;:\; 1 \leq n \leq N$

$p:\ s\ = \sum_{i=1}^{n} a_i \cdot c_i\ \wedge\ 1 \leq\ n\ \leq\ N$
$b:\ n\ \neq\ N$
Initialisierung: $n := 1;\ s := a_1 * c_1$
oder noch besser wähle:
$p:\ s\ = \sum_{i=1}^{n} a_i \cdot c_i\ \wedge\ 0 \leq\ n\ \leq\ N$
$b:\ n\ \neq\ N.$
Dann einfachere Initialisierung: $n := 0;\ s := 0$
wegen $N \geq 0$ und Initialisierung $n := 0$ muß zur Termination der Schleife ($n = N$) die Variable n erhöht werden:
$n := n + 1;$
dann aber zur Wiederherstellung der Invariante
$s := s + a_n * c_n$
zu setzen! Damit ergibt sich das Programm P:
$n := 0;\ s := 0;$ **while** $n \neq N\ n := n + 1;\ s := s\ +\ a_n * c_n$ **od**

Zur Lösung der Aufgabe „Kugelproblem“
Zunächst wird deutlich, daß ein Herangehen mit Fallunterscheidung kaum zum Ziel führt. Mancher ist vielleicht versucht, etwa zu probieren, was passiert, wenn anfangs eine schwarze und eine weiße Kugel vorhanden sind, dann den Fall 2 schwarze, 1 weiße Kugel usw.
Dies entspräche einem Vorgehen wie beim Einfahren eines Programms durch endlich viele Testläufe mit mehr oder weniger geeigneten Beispielen.
Besser ist ein Herangehen wie bei einem modernen Programmierstil: Entwurf und Beweis der Programmeigenschaften gleichzeitig. Dabei ist das „Denken in Invarianten“ gefragt.
Bei unserer Aufgabe geht es um die Frage nach einer einfachen Eigenschaft der Kugeln, die *invariant* bleibt. Angenommen, eine schwarze Kugel ist übriggeblieben. Bei der Frage nach der Invarianten muß man nach einer Eigenschaft suchen, die bis zum Schluß wahr ist, die als Invariante verallgemeinert werden könnte.
1 ist eine ungerade Zahl. Ist die Anzahl der schwarzen Kugeln etwa immer ungerade? Nein, sie wechselt bei jedem Schritt von gerade zu ungerade und umgekehrt. Andererseits bedeutet 1 schwarze Kugel am Ende, daß 0 weiße Kugeln übrig sind. 0 ist gerade Zahl. Ist die Anzahl der weißen Kugeln vielleicht immer gerade? Ja, die Zahl der weißen Kugeln wird - wenn überhaupt - stets um 2 verringert!
Damit haben wir die **Lösung:** Ist die Ausgangszahl der weißen Kugeln gerade, so bleibt eine schwarze übrig. Ist sie ungerade, so bleibt eine weiße Kugel übrig.

3.4.4 Zur Frage der Vollständigkeit von H

Diese Frage ist nicht nur von theoretischem Interesse. Während die Frage der Widerspruchsfreiheit die Korrektheit der Beweismethode betrifft, entspricht die Vollständigkeitsfrage der nach der Anwendbarkeitsbreite der Methode. Sie ist also auch von höchst praktischem Interesse!
Es interessiert, ob die Umkehrung des Widerspruchsfreiheitssatzes gilt :
Für alle Interpretationen $\Im$ *und alle Spezifikationen* Φ *:*
wenn $\models\ \Phi$, so $\mathbf{TR}_{\Im}\ \vdash_H\ \Phi$ **?**
(„Vollständigkeit“)

Leider gilt aber :

Satz 3.12

Das Beweissytem H für iterative Programme ist nicht *vollständig.*

Dazu hat M. WAND 1978 bewiesen, daß es eine bestimmte Sprache **AUS** (Sprache einer festen elementaren Theorie) gibt, und eine feste Interpretation $\Im$ sowie eine Spezifikation Φ, so daß zwar $\models_{\Im} \Phi$ ist, aber nicht $\mathbf{TR}_{\Im} \vdash_H \Phi$. Diese Unvollständigkeit rührt daher, daß die notwendigen Invarianten und Zwischenbehauptungen in der Sprache bei dieser Interpretation nicht ausdrückbar sind!

Dieser Satz läßt sich noch verschärfen :

Satz 3.13 (BERGSTRA, J.A./TUCKER, J.V.1981)

Kein *Beweissystem G für iterative Programme kann vollständig sein.*

> Ein Argument dafür ist die sogenannte Unentscheidbarkeit des Halteproblems für **PR** bei der Standardinterpretation $\Im_N$ der natürlichen Zahlen :
> $\models_{\Im_N} \{\texttt{true}\}\ S\ \{\texttt{false}\}$ genau dann, wenn S *nicht* terminiert.
> Gäbe es ein vollständiges Beweissystem G, so hätte man mit $\models_{\Im_N} \{\texttt{true}\}\ S\ \{\texttt{false}\}$ auch $\vdash_G \{\texttt{true}\}\ S\ \{\texttt{false}\}$, und damit wäre die Menge der nicht terminierenden Programme S rekursiv aufzählbar. Aus der Unentscheidbarkeit des obengenannten Halteproblems ergibt sich aber, daß sie eben nicht rekursiv aufzählbar sein kann.

Diese Unvollständigkeitsresultate dürfen nicht entmutigen, denn *in praxi* interessieren nicht beliebige Interpretationen, sondern z.B. nur die Standardinterpretationen der Arithmetik der ganzen, rationalen oder reellen Zahlen, Interpretationen über endlichen Bereichen usw. (Es interessieren *nicht* die „Nichtstandardinterpretationen", die für die Logiker und Grundlagenfragen von Interesse sind.)

Wichtig ist also die Herausarbeitung von Klassen von Interpretationen, für die *relative* Vollständigkeit erhältlich sind. Der bei WAND gegebene Hinweis gibt Anlaß für die Betrachtung solcher Interpretationen, für die $\underline{\text{pre}}_{\Im}(S, q)$ stets ausdrückbar ist. (S.A. COOK 1978 und E.M. CLARKE JR. 1979).

Definition 3.8

Ein Beweissystem G für eine Programmiersprache **PR** ist *vollständig im Sinne von* COOK , kurz C–*vollständig*, wenn G relativ vollständig bezüglich $INT = \{\Im \mid \Im \in EXP(\mathbf{AUS}, \mathbf{PR})\}$ ist.
Das heißt, G ist C–vollständig genau dann, wenn für jedes $\Im \in EXP(\mathbf{AUS}, \mathbf{PR})$, jedes $\Phi \in \mathbf{SP}$ gilt:

$$\models_{\Im} \Phi \qquad \text{impliziert} \qquad \mathbf{TR}_{\Im} \vdash_G \Phi.$$

Satz 3.14

Das Beweissystem H für `while` *–Programme ist C–vollständig.*

Beweis

Es sei $\Im \in EXP(\mathbf{AUS}, \mathbf{PR})$ und $\models_{\Im} \{p\}S\{q\}$. Der Beweis von $\mathbf{TR}_{\Im} \vdash_H \{p\}S\{q\}$ wird induktiv über den Aufbau von S geführt. Dabei kann der Induktionsanfang sowie der Induktionsschritt Fälle 1.) und 2.) direkt vom Beweis des Vollständigkeitssatzes für H_0 übernommen werden. Es bleibt der Fall
3.) $S \equiv$ `while` b `do` S_1 `od` , wobei für S_1 die Behauptung vorausgesetzt werden kann (entspricht der Induktionsvoraussetzung).
Es sei $r = wp_{\Im}(S, q)$. $r \in \mathbf{AUS}$ nach Voraussetzung der Ausdrucksfähigkeit und entsprechendem Satz

aus Abschnitt „Vorbedingung, Nachbedingung, ...“)
Wir zeigen :

a.) $\models_{\Im} \; p \; \rightarrow \; r$, d.h. $p \rightarrow r \; \in \; \underline{\mathbf{TR}}_{\Im}$

b.) $\models_{\Im} \; r \; \wedge \; \neg b \; \rightarrow \; q$

c.) $\models_{\Im} \; \{r \; \wedge \; b\} S_1 \{r\}$

Dann folgt mit (IV): $\underline{\mathbf{TR}}_{\Im} \; \vdash_H \; \{r \; \wedge \; b\} S_1 \{r\}$, und mit R_{wh} $\underline{\mathbf{TR}}_{\Im} \; \vdash_H \; \{r\} S \{r \; \wedge \; \neg b\}$ (mit Invariante r), und mit Implikationsregel $R_{\rightarrow}$ dann die Behauptung $\underline{\mathbf{TR}}_{\Im} \; \vdash \; \{p\} S \{q\}$.

zu a.) $r \; = \; wp_{\Im}(S, q)$
Nach Definition von wp (2. Bedingung) folgt aus $\models_{\Im} \; \{p\} S \{q\}$ sofort $\models_{\Im} \; p \; \rightarrow \; r$

zu b.) Sei $\mathrm{wert}_{\Im}(r \; \wedge \; \neg b, \sigma) = W$, d.h. $\mathrm{wert}_{\Im}(r, \sigma) = W, \quad \mathrm{wert}_{\Im}(b, \sigma) = F$.
Nach Fakt(2) folgt : $\mathrm{val}_{\Im}(S, \sigma) \; = \; \sigma$ und wegen $r \; = \; wp_{\Im}(S, q) \; \models_{\Im} \; \{r\} S \{q\}$ und damit auch $\mathrm{wert}_{\Im}(q, \mathrm{val}_{\Im}(S, \sigma)) = \mathrm{wert}_{\Im}(q, \sigma) = W$, also $\models_{\Im} \; r \; \wedge \; \neg b \; \rightarrow \; q$

zu c.) Sei $\mathrm{wert}_{\Im}(r \wedge b, \sigma) = W$ und sei $\mathrm{val}_{\Im}(S_1, \sigma) = \sigma'$ definiert. Nach Fakt 3.) ist $\mathrm{val}_{\Im}(S, \sigma) = \mathrm{val}_{\Im}(\underline{\texttt{if}}\; b \; \underline{\texttt{then}}\; S_1; S \; \underline{\texttt{else}}\; \underline{\texttt{skip}}\; \underline{\texttt{fi}}$,
d.h. wegen $\mathrm{wert}_{\Im}(b, \sigma) = W$

$$\begin{aligned} \mathrm{val}_{\Im}(S, \sigma) \; &= \; \mathrm{val}_{\Im}(S_1; S, \sigma) \\ &= \; \mathrm{val}_{\Im}(S, \mathrm{val}_{\Im}(S_1, \sigma)) \\ &= \; \mathrm{val}_{\Im}(S, \sigma'). \end{aligned}$$

Nach Voraussetzung über $\Im$ ist $\underline{\mathrm{pre}}_{\Im}(S, q)$ darstellbar, und zwar wird es durch $wp_{\Im}(S, q)$ dargestellt, d.h. durch r. Damit ist $\sigma \; \in \; \underline{\mathrm{pre}}_{\Im}(S, q)$ genau dann, wenn $\mathrm{wert}_{\Im}(r, \sigma) = W$ genau dann, wenn $\mathrm{val}_{\Im}(S, q)$ nicht definiert ist oder (nach Definition von $\underline{\mathrm{pre}}_{\Im}$) $\mathrm{wert}_{\Im}(q, \mathrm{val}_{\Im}(S, \sigma)) = W$ genau dann, wenn $\mathrm{val}_{\Im}(S, \sigma')$ undefiniert oder $wert(q, \mathrm{val}_{\Im}(S, q')) = W$ genau dann, wenn (nach Definition von $\underline{\mathrm{pre}}_{\Im}$) $\sigma' \; \in \; \underline{\mathrm{pre}}_{\Im}(S, q)$ genau dann, wenn $\mathrm{wert}_{\Im}(r, \sigma') = W$.
Also folgt : Falls $\mathrm{val}_{\Im}(S_1, \sigma)$ definiert, so $\mathrm{wert}_{\Im}(r, \mathrm{val}_{\Im}(S_1, \sigma)) = W$. Damit $\models_{\Im} \; \{r \; \wedge \; b\} S_1 \{r\}$. q.e.d.

Um die Tragfähigkeit des Vollständigkeitssatzes abzuschätzen, muß untersucht werden, wie einschneidend die Voraussetzung bezüglich der Ausdrucksfähigkeit ist. Glücklicherweise ist bekannt :

1. Für die Sprache L_N der elementaren Arithmetik $(+, \cdot)$ und deren Standardinterpretation $\Im_N$ im Bereich der natürlichen Zahlen gilt :

   $$\Im_N \; \in \; EXP(L_N, \underline{\mathbf{PR}})$$

   .

   Zum Beweis sind Mittel aus der Theorie der rekursiven Funktionen nötig. Ein Beweis findet sich in [DE BAKKER, J.W.] im Anhang.

   Damit kann jede gültige Spezifikation von Programmen der Arithmetik natürlicher (und dann auch ganzer) Zahlen in H bewiesen werden (unter der Voraussetzung der Kenntnis aller gültigen Formeln der Arithmetik der natürlichen Zahlen – „Orakel“).
   Wegen des GÖDELschen Unvollständigkeitssatzes kann man diese Voraussetzung allerdings nicht ebenfalls auf eine formale Ableitbarkeit reduzieren).

   Mit diesem Resultat ergibt sich, daß H in der Informatik brauchbar ist.

Es gilt auch (CLARKE 1979):

2. Wenn der Individuenbereich von $\Im$ *endlich* ist, dann ist

   $$\Im \; \in \; EXP(\underline{\mathbf{AUS}}, \underline{\mathbf{PR}}).$$

Es stellt sich heraus, daß dies prinzipiell die einzigen 2 Möglichkeiten für die Ausdrucksfähigkeit sind.
Es gilt („grob“ formuliert):

Satz 3.15

Wenn $\Im \in EXP(\underline{\mathbf{AUS}}, \underline{\mathbf{PR}})$ *, so ist*

1. *ein Standardmodell der elementaren Arithmetik natürlicher Zahlen in* $\Im$ *definierbar. Oder*
2. *für jedes* $S \in \underline{\mathbf{PR}}$ *gibt es ein n, so daß bei der Ausführung von S höchstens n Zustände über dem Individuenbereich von* $\Im$ *durchlaufen werden bei beliebigem Eingangszustand.*

Kapitel 4

Ausblick : Korrektheitsregeln für weitere Sprachkonstruktionen

4.1 Indizierte Variablen

Wir behandeln die indizierten Variablen aus Zeitgründen hier nur im Spezialfall („eindimensionale Felder"; allgemein kann man unter den Begriff der indizierten Variablen neben beliebig-dimensionalen Feldern auch Zugriffe zu „records" u.ä. fassen) und ohne exakte Beschreibung ihrer Semantik.[1]

AV sei im folgenden eine vorgegebene Menge sogenannter *Feldvariablen* (array variables), disjunkt zum bisherigen Alphabet und disjunkt zu **TERM** , **AUS** und **PR** . Die Klammern „[" und „]" mögen neu zum Alphabet hinzukommen.

Definition 4.1

In die Termdefinition ist aufzunehmen :
mit $a \in$ **AV** und $Z \in$ **TERM** ist auch $a[t] \in$ **TERM** und heißt *indizierte Variable*.

indV bezeichne die Menge aller indizierten Variablen. Es wird gesetzt :

VAR $=_{df}$ **V** $\cup$ **indV** – die Elemente von **VAR** heißen nun *Variablen*, und die Elemente von **V** nunmehr *einfache Variablen* .

Die **Definition** der Menge $\mathbf{PR}_{iV}$ der *Programme mit indizierten Variablen* ist genauso wie die Definition von **PR** , nur mit $\mathbf{PR}_{iV}$ statt **PR** und im Definitionsschritt 0. steht statt $x \in$ **V** nunmehr $x \in$ **VAR** .

Bemerkung

Damit sind Zuweisungen von indizierten Variablen in der Sprache enthalten. Sowohl in den Indexausdrücken wie auch in den rechten Seiten solcher Zuweisungen können wiederum indizierte Variablen auftreten.

[1] Die Problematik wurde u.a. vom Lesenden ausführlich behandelt unter Einbeziehung simultaner Zuweisungen und bei iterierter Mehrfachindizierung. Vgl. dazu und für weitergehende Literaturangaben: HARTWIG, R. *Elektron. Informationsverarb.Kybernet.* **17** (1981) 1, pp. 39-60, und HARTWIG, R. *Periodica Math. Hung.* **15** (1984) 1, pp. 61-71.

Beispiel 18

(„Curiosa“ mit indizierten Variablen)

1. Es zweifelt kaum jemand an der Gültigkeit von

   $$\{\underline{\texttt{true}}\}\ v := 1\ \{v = 1\},$$

   jedoch für indizierte Variable v kann dies falsch sein, wie das Beispiel :

   $$v \ \equiv \ a[a[2]]$$

   mit den Anfangswerten $a[1] = 2,\ a[2] = 2$ zeigt. Nach $v := 1$ gilt nämlich $v = 2$.

2. Ähnlich verhält es sich bei

   $$\{v = 1\}\ v := v + 1\ \{v = 2\}.$$

   Mit $v = a[a[1]]$ ist folgende Aussage erfüllbar:

   $$\{v = 1\}\ v := v + 1\ \{v = 0\}$$

   (Wähle als Anfangswerte $a[1] = 1$ und $a[2] = 0$.)

3. Ebenso falsch ist die Annahme, daß das Programm

   $$S :\ n := x;\ x := y;\ y := n$$

   die Werte von x und y vertauscht, daß also

   $$\{x = X\ \wedge\ y = Y\}\ S\ \{x = Y\ \wedge\ y = X\}$$

   für beliebige X, Y gilt.

   Als Gegenbeispiel haben wir erfüllbar :

   $$\{a[a[1]] = 1\ \wedge\ a[a[2]] = 2\}\ h := a[a[2]];\ a[a[1]] := a[a[2]];\ a[a[2]] := h\ \{a[a[1]] = 1\ \wedge\ a[a[2]] = 2\}$$

   Wähle nämlich $\sigma\ \in\ \underline{\mathbf{ST}}$ so, daß $a[1] = 2$ und $a[2] = 1$ gilt.

Die Beispiele zeigen die Notwendigkeit der *Verallgemeinerung der Zuweisungsaxiome* an indizierte Variablen.

Der Schlüssel zur Lösung des Problems liegt in der genauen Analyse und Beschreibung der Semantik (des „Auswertungsmechanismus“) indizierter Variablen und auf dieser Grundlage der Definition der Substitution für indizierte Variablen. Für skizzenhafte Ausführungen dazu sei auf die Vorlesungen des Lesenden zu den „Algebraischen Grundlagen der Informatik“ verwiesen.
Grundsätzlich haben Variablen zwei verschiedene Bedeutungen: *Name* (bzw. Adresse) und *Wert* genannt. Je nachdem, ob eine Variable in einem Ausdruck (etwa auf der rechten Seite einer Wertzuweisung) steht, oder ob sie die linke Seite einer Wertzuweisung bildet, ist als deren Bedeutung eben ihr Name oder ihr Wert zu berechnen.

Das (frühere) HOARE–Axiom Z hat die Gestalt :

$$\{wp_{\mathfrak{Z}}(S, p)\}\ S\ \{p\},$$

wobei S eine Zuweisung ist (vgl. dazu die Folgerung 3.7 auf Seite 24).

$$\{p^x/t\}\ x := t\ \{p\}$$

Nun gilt der

Satz 4.1

S sei ein stets terminierendes Programm (d.h. $val_{\Im}(S,\sigma)$ ist immer definiert). Dann gilt:
Wenn für alle $\sigma \in \underline{\mathbf{ST}}$

$$\text{wert}_{\Im}(q,\sigma) = \text{wert}_{\Im}(p, \text{val}_{\Im}(S, sigma))$$

ist, so gilt :

$$\models \; q = wp_{\Im}(S,p).$$

Beweis

1. Aus der Voraussetzung folgt : $\models \{q\}S\ \{p\}$.
   Sei nämlich $\sigma \in \underline{\mathbf{ST}}$ *mit* $\text{val}_{\Im}(S,\sigma)$ definiert. Ist dann $\text{wert}_{\Im}(q,\sigma) = W$, so wegen der Gleichung auch $\text{wert}_{\Im}(p, \text{val}_{\Im}(S,\sigma)) = W$.
2. Sei auch $\models \{r\}\ S\ \{p\}$, d.h. für alle $\sigma \in \underline{\mathbf{ST}}$: wenn $\text{wert}_{\Im}(r,\sigma) = W$, *so* $\text{wert}_{\Im}(p, \text{val}_{\Im}(S,\sigma)) = W$.
   Also nach Voraussetzung auch $\text{wert}_{\Im}(q,\sigma) = W$. Damit $\models r \rightarrow q$.

q.e.d.

Bemerkung

Es gilt sogar die Umkehrung des Satzes (also der Satz mit „genau dann, wenn“).
Er wird hier aber nicht benötigt.

In Anlehnung an den (schon früher zitierten) Satz über Wertänderung bei Substitution (vgl. Vorlesung über algebraische Grundlagen der Informatik), der aussagt, daß

$$\text{wert}_{\Im}(sub(p,s),\sigma) = \text{wert}_{\Im}(p,\sigma')$$

für den durch die Einsetzungsabbildung s geänderten Zustand σ'
kann man fragen nach der Existenz einer Einsetzungsabbildung s ($=_{df}$ eindeutige Abbildung der Menge der Variablen in die Menge der Terme)

$$s : \underline{\mathbf{VAR}} \rightarrow \underline{\mathbf{TERM}},$$

die *adäquat* zu der durch das Programm S bewirkten Zustandsänderung von σ nach $\sigma' = \text{val}_{\Im}(S,\sigma)$ ist, d.h. die Gleichung

$$\text{wert}_{\Im}(sub(p,s),\sigma) = \text{wert}_{\Im}(p, \text{val}_{\Im}(S,\sigma))$$

erfüllt. Denn dann hat man nach obigem Satz

(Z') $\boxed{\{\ sub(p,s)\ \}\ S\ \{\ p\ \}}$

(mit s als adäquate Einsetzung zu S) als *allgemeines Zuweisungsaxiom.*
Solche für Zuweisungen an i.a. indizierte Variablen adäquate Einsetzungen gibt es. Man kann „ausrechnen“, daß die in der folgenden Definition festgelegte Einsetzung s adäquat zu der Wertzuweisung $v := t$ ist.

Definition 4.2

Zu der Anweisung $v := t$ mit $v \in \underline{\mathbf{VAR}}$, $t \in \underline{\mathbf{TERM}}$ wird rekursiv folgende Einsetzungsabbildung

$$s : \underline{\mathbf{VAR}} \rightarrow \underline{\mathbf{TERM}}$$

als *adäquat* festgelegt:

1. für $u \in \underline{\mathbf{V}}$ gilt:
   - $s(u) = u$ für $u \not\equiv v$
   - $s(u) = t$ für $u \equiv v$
2. für $u \equiv a[e] \in \underline{\mathbf{indV}}$
   - $s(u) = a[sub(e,s)]$ für $v \in \underline{\mathbf{V}}$ oder $v \equiv b[e']$ und $a \not\equiv b$
   - $s(u) = \underline{\texttt{if}}\ e = e'\ \underline{\texttt{then}}\ t\ \underline{\texttt{else}}\ a[e]\ \underline{\texttt{fi}}$ für $v \equiv a[e']$.

Man beachte, daß gilt:

$$sub(a[r], s) = s(a[sub(r, s)]).$$

(Hier ohne nähere Begründung.)

Bemerkung

Um den Nachweis zu führen, daß die hier definierte Einsetzung s die geforderte Bedingung erfüllt (also tatsächlich „adäquat“ ist), erfordert die genaue Festlegung der Semantik indizierter Variablen.

Beispiel 19

der Anwendung des allgemeinen Zuweisungsaxioms (Z'):

1. Betrachte wie oben im Beispiel 1. die Anweisung $a[a[2]] := 1$ und als Nachbedingung $a[a[2] = 1$. Damit erhält man als *schwächste Vorbedingung* q gemäß (Z')

   $$q \doteq sub(a[a[2]] = 1, s),$$

   wobei s eine zu obiger Anweisung adäquate Einsetzung ist.
   $\doteq$ bezeichne im folgenden die Wertverlaufsgleichheit logischer Ausdrücke. Nun gilt:

   $$\begin{aligned} q &\doteq sub(a[a[2]] = 1, s) \\ &\doteq sub(a[a[2]], s) = sub(1, s) \\ &\doteq s(a[sub(a[2], s)]) = 1 \\ &\doteq s(a[s(a[sub(2, s)])]) = 1 \\ &\doteq s(a[s(a[2])]) = 1 \end{aligned}$$

   Nach Definition von s erhalten wir :

   $$s(a[2]) = \underline{\texttt{if}}\ 2 = a[2]\ \underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[2]\ \underline{\texttt{fi}},$$

   d.h. wir haben

   $$q \doteq s(a[\ \underline{\texttt{if}}\ a[2] = 2\ \underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[2]\ \underline{\texttt{fi}}) = 1$$

   und damit nach Definition von s:

   $$q \doteq \underline{\texttt{if}}\ \underline{\texttt{if}}\ a[2] = 2\ \underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[2]\ \underline{\texttt{fi}} = a[2]$$

   $$\underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[\underline{\texttt{if}}\ a[2] = 2\ \underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[2]\ \underline{\texttt{fi}}\ ]\underline{\texttt{fi}} = 1$$

$$
\begin{aligned}
\doteq\ & (\underline{\texttt{if}}\ a[2] = 2\ \underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[2]\ \underline{\texttt{fi}}\ = a[2]\ )\ \wedge\ 1 = 1 \\
& \vee\ (\underline{\texttt{if}}\ a[2] = 2\ \underline{\texttt{then}}\ 1\ \underline{\texttt{else}}\ a[2]\ \underline{\texttt{fi}}\ \neq\ a[2]) \wedge\ a[\underline{\texttt{if}}\ \ldots\ \underline{\texttt{fi}}]\ =\ 1 \\
\doteq\ & (a[2] = 2\ \wedge\ 1 = a[2])\ \vee\ (a[2] \neq 2\ \wedge a[2] = a[2]) \\
& \vee\ (a[2] = 2\ \wedge\ 1\ \neq a[2])\ \vee\ (a[2] \neq 2\ \wedge\ a[2] \neq a[2])) \\
& \wedge\ a[\underline{\texttt{if}} \ldots \underline{\texttt{fi}}]\ =\ 1
\end{aligned}
$$

$$\doteq\ \boldsymbol{a[2]\ 6{=}\ 2\ \dot{}\ (a[2] = 2\ \hat{}\ a[1] = 1)}$$

$v = 1$ wie erwartet im Eingangsbeispiel kommt also nur unter diesen Bedingungen heraus. Diese hatten wir oben verletzt:
dort war $a[2] = 2$, aber auch $a[1] = 2$ gewählt, was eben $v = 2$ ergab.

2. Betrachte Anweisung $v := v + 1$ (wie im Beispiel oben), d.h.

$$a[a[1]] := a[a[1]] + 1,$$

und als Nachbedingung $v = k$ (k ganze Zahl).
Berechne die schwächste Vorbedingung (als Übungsaufgabe !)

Bemerkung

Mit dem Begriff der adäquaten Einsetzung gelingt es auch, zu vorgegebenen Wertzuweisungen $a[t] := e$ und gegebener Vorbedingung die *stärkste Nachbedingung* zu bestimmen (**verallgemeinertes FLOYD–Axiom**).
Es gilt:
für $a\ \in\ \underline{\textbf{AV}}$, t, $e \in\ \underline{\textbf{TERM}}$, $p \in\ \underline{\textbf{AUS}}$:

$$sp(a[t] := e, p)\ \doteq\ \exists\ x \exists\ z\ (sub(p, s)\ \wedge\ a[z] = sub(e, s)\ \wedge\ z = sub(t, s)),$$

wobei $x, z\ \notin\ var\{p, t, e\}$ gilt, und s die zu $a[z] := x$ adäquate Einsetzung ist.

Damit haben wir das

FLOYD–Axiom für Zuweisungen an indizierte Variablen:

$$\{p\}\ a[t] := e\ \{\exists\ x\ \exists\ z(sub(p, s)\ \wedge\ a[z] = sub(e, s)\ \wedge\ z = sub(t, s))\},$$

wobei $x, z\ \notin\ var\{p, t, e\}$ und s die zu der Anweisung $a[z] := x$ adäquate Einsetzung ist.

Beispiel 20

Betrachte wieder Beispiel 1 von vorhin:

$$\underbrace{a[a[2]] := 1}_{v:=1}$$

Es sei „keine Vorbedingung“ gestellt, d.h. wir setzen $p\ =\ \underline{\texttt{true}}$. Damit ist eine vollständige Analyse der Wirkung dieser Anweisung möglich. Bestimme also $sp(a[a[2]] := 1, \underline{\texttt{true}})$. Es ergibt sich $sub(p, s) \equiv\ \underline{\texttt{true}}$, entfällt damit als Konjunktionsglied. Weiter ist $sub(e, s) = sub(1, s) = 1$. Wesentlich ist folglich nur $sub(t, s)$:

$$sub(t, s) = sub(a[2], s) = s(a[2])$$

s adäquat zu $a[2] := x$:

$$s(a[2]) \;=\; \underline{\texttt{if}}\; z \;=\; 2\; \underline{\texttt{then}}\; x\; \underline{\texttt{else}}\; a[2]\; \underline{\texttt{fi}}$$

Als *stärkste Nachbedingung* ergibt sich:

$$\doteq\; \exists\, x\, \exists\, z(a[z] = 1 \;\wedge\; z = \underline{\texttt{if}}\; z = 2\; \underline{\texttt{then}}\; x\; \underline{\texttt{else}}\; a[2]\; \underline{\texttt{fi}})$$

$$\doteq\; \exists\, x\, \exists\, z(a[z] = 1 \;\wedge\; ((z = 2 \;\wedge\; z = x) \vee\; (z \neq 2 \wedge\; z = a[2])))$$

$$\doteq\; \exists\, x\, \exists\, z((a[z] = 1 \;\wedge\; z = x = 2) \;\vee\; (a[z] = 1 \;\wedge\; z = a[2] \;\wedge\; z \neq 2))$$

$$\doteq\; \exists\, x\, \exists\, z(z = x = 2 \;\vee\; z \neq 2) \;\wedge\; (a[2] = 1 \;\vee\; (a[a[2]] = 1 \;\wedge\; a[2] \;\neq 2))$$

$$\doteq\; a[2] = 1 \;\vee\; a[a[2]] = 1$$

In unserem Anfangsbeispiel ist die 1. Alternative wahr geworden (aus ihr ergab sich dann v=2)!

4.2 Blockstruktur

Definition 4.3

$\underline{\mathbf{PR}}_B$ sei die kleinste Menge von Zeichenreihen, für die gilt:
0., 1., 2. und 3. der Definition von $\underline{\mathbf{PR}}$, nur mit $\underline{\mathbf{PR}}_B$ statt $\underline{\mathbf{PR}}$, und

4. mit $x \in \underline{\mathbf{V}}$ und $S \in \underline{\mathbf{PR}}_B$ ist auch

$$\underbrace{\underline{\texttt{begin}}\; \underline{\texttt{new}}\; x;\; S\; \underline{\texttt{end}}}_{\text{„}Block\text{“}} \in \underline{\mathbf{PR}}_B.$$

Sprechweisen:

- x gültig im Block
- x lokale Variable
- x gebundene Variable
- lokale Variablen entsprechen den gebundenen Variablen in logischen Ausdrücken (dort gebunden durch $\exists$, $\forall$)
- Eine weitere Bedeutung als die Einführung einer neuen Variablen in einen „Gültigkeitsbereich“ (etwa die Zuordnung zu einer „Wertsorte“) interessiert uns hier bei der *Variablendeklaration* $\underline{\texttt{new}}$ x nicht !

Bemerkung zur Semantik:

- Eine einfache Beschreibung der Semantik ist möglich bei Änderung des Zustandsbegriffes:

Zustand $=_{df}$ Abbildung einer *endlichen* Menge von Variablen in Werte

- Mit $\underline{\texttt{new}}$ x ist nun x in den Definitionsbereich $dom(\sigma')$ des aktuellen Zustandes σ' mit aufzunehmen, d.h. $dom(\sigma')$ zu erweitern. Als Initialwert von x bei Eintritt in den Block könnte ein ausgezeichneter Wert $\perp$ („undefiniert“) speziell in die Wertmenge aufgenommen werden – oder z.B. **Null** bei Zahlvariablen verwendet werden.

$$\sigma'(x) = \perp \qquad \text{bzw.} \qquad \sigma'(x) = 0$$

Bei Austritt aus dem Block ist x wieder aus dem Definitionsbereich des aktuellen Zustands zu streichen (beschrieben durch **DROP** (σ', x))!

$$\mathrm{val}_{\Im}(\underline{\texttt{begin}}\ \underline{\texttt{new}}\ x;\ S\ \underline{\texttt{end}},\ \sigma) = \underline{\mathbf{DROP}}(\mathrm{val}_{\Im}(S^x/_y,\ \sigma\ \cup\ \{(y, \bot)\})\ ,\ y)$$

Mit y ist dabei die (irgendwie festgelegte) erste Variable $\notin dom(\sigma)$ bezeichnet.

Damit ist klar:
Die Bedeutung von

$$\underline{\texttt{begin}}\ \underline{\texttt{new}}\ x;\ S\underline{\texttt{end}}$$

ist unabhängig von der Bezeichnung der lokalen Variablen x. Dies muß in einer Beweisregel für Blöcke ausgedrückt werden.

Beweisregel für Blöcke

$$R_B\ :\qquad \frac{\{p\}\ S^x/_y\ \{q\}}{\{p\}\ \underline{\texttt{begin}}\ \underline{\texttt{new}}\ x;\ S\ \underline{\texttt{end}}\ \{q\}},$$

wobei $y \notin\ frei\{p, S, q\}$ (nicht gebunden durch $\exists$, $\forall$, $\underline{\texttt{new}}$).

Beispiel 21

Es gilt sicher:

$$\{x = -1\ \wedge\ z > 0\}\ \underline{\texttt{begin}}\ \underline{\texttt{new}}\ y;\ y := 5\ \underline{\texttt{end}}\ \{x < 0\}, \qquad (1)$$

aber auch

$$\{x = -1\ \wedge\ z > 0\}\ \underline{\texttt{beginnew}}\ x;\ x := 5\ \underline{\texttt{end}}\ \{x < 0\}. \qquad (2)$$

(1) und (2) lassen sich beweisen wegen:

$$\vdash\ \{x = -1\ \wedge\ z > 0\}\ u := 5\ \{x < 0\}\ ,$$

(man beachte, daß $(x = -1\ \rightarrow\ x < 0)\ \in\ \underline{\mathbf{TR}_{\Im}}$)
mit der Regel R_B, wobei für (1) die Einsetzung $^y/_u$ und für (2) die Einsetzung $^x/_u$ verwendet wird.

- Läßt man aber die Voraussetzung $y\ \notin\ frei\{p, S, q\}$ in R_B fallen, ist die Regel nicht mehr korrekt:

  a) $y\ \in\ frei\{p, q\}$
  z.B. gilt

  $$\{y = -1\}\ \overbrace{y := 5}^{S^x/_y}\ \{y > 0\},$$

  aber nicht

  $$\{y := -1\}\ \underline{\texttt{beginnew}}\ x; \underbrace{\underline{x := 5}}_{S}\ \underline{\texttt{end}}\ \{y > 0\}.$$

  b) $y\ \in\ frei(S)$
  z.B. gilt

  $$\{y = 0\}\ \overbrace{y := y + 1}^{S^x/_y}\ \{y = 1\},$$

  aber natürlich nicht

  $$\{y = 0\}\ \underline{\texttt{begin}}\ \underline{\texttt{new}}\ x; \underbrace{\underline{x := y + 1}}_{S}\ \underline{\texttt{end}}\ \{y = 1\}$$

Resultate:

a) Regel R_B ist korrekt.
(Der Beweis ist leicht möglich nach genauer Ausführung der Semantik.)

b) Das Beweissystem $H \cup \{R_B\}$ ist für iterative Programme mit Blockstruktur zwar korrekt, aber *nicht vollständig* (auch nicht C–vollständig).
Denn, betrachte die Anweisungen:

$S_1 \equiv$ **begin** **new** z; $x := z$ **end**

$S_2 \equiv$ **begin** **new** u; $y := u$ **end**.

Bei der oben angedeuteten Semantik (neue Variable zu initialisieren mit $\perp$ bzw. 0) hat man

$$\models \{\mathtt{true}\} S_1;\ S_2\ \{x = y\},$$

aber es gibt keinen Weg, dies zu beweisen!

c) Die obigen "Schwierigkeiten" bilden aber den einzigen Grund für die Unvollständigkeit des Beweissystems.
Sie ist also nicht in der „Schwäche" der Regeln, sondern in der übergenauen Beschreibung der Semantik (Belastung mit Implementationseinzelheiten!) begründet. Derartige, durch eine spezielle Implementation hervorgerufene Eigenschaften zu verifizieren, ist i.a. nicht einmal erwünscht, sie sollen für den Nutzer „verborgen" bleiben. Das heißt, hier ist die *Unvollstandigkeit des Beweissystems zweckmäßig* (vgl. als Analogon z.B. die Unvollständigkeit der Gruppentheorie in der Algebra).

d) *Vollständigkeit des Beweissystems* wäre auf verschiedene Weise „künstlich" erreichbar:

   a) Beschränkung der Programme durch Verbot der Verwendung uninitialisierter lokaler Variablen in den rechten Seiten von Ergibtanweisungen.

   b) Aufnahme einer Konstanten $\perp$ in die Sprache **AUS** und Abänderung der Regel R_B in

$$R'_B: \quad \frac{\{p \wedge y = \perp\}\ S^x/_y\ \{q\}}{\{p\}\ \mathtt{begin}\ \mathtt{new}\ x;\ S\ \mathtt{end}\ \{q\}},$$

   $y \notin frei\{p, S, q\}$.

Index