Der Hoare-Kalkül

Sabine Laszakovits

13. September 2012

Inhaltsverzeichnis

Vorwort

Die vorliegende Arbeit soll eine Einführung in den Hoare-Kalkül geben. Sie richtet sich gezielt an interessierte Studenten der Lehrveranstaltung "Theoretische Informatik und Logik" (TIL) an der Technischen Universität Wien, die einen größeren Überblick erlangen und sich tiefergehend mit der Materie beschäftigen möchten. Darum legt diese Arbeit Wert darauf, nicht nur Ableitungsregeln einzuführen, sondern diese auch zu motivieren, sowie sich mit totaler Korrektheit, Korrektheitsbeweisen und Vollständigkeitsaussagen auseinanderzusetzen. Am Ende werden fünf kurze Programme durchbesprochen und in Bezug auf ihre Korrektheit im Detail vorgerechnet.

Der Besuch der oben genannten Lehrveranstaltung soll aber keine Voraussetzung für die Lektüre dieser Arbeit sein. Sie ist als eigenständiges Werk konzipiert und weicht in Notation und Beispiel-Programmiersprache von den Unterrichtsmaterialien geringfügig ab. Dies sollte für interessierte Leser kein Hindernis darstellen, sondern einen ersten Einstieg in die Literatur bilden, wo auch andere Notationen vorherrschen.

1 Einleitung

1.1 Der Hoare-Kalkül

Eines der Kerngebiete der Informatik ist das Schreiben von Programmen nach Spezifikationen. Programme, die sich falsch verhalten, können großen Schaden anrichten. Aus diesem Grund ist eine systematische Überprüfung unumgänglich. Die häufigste Methode, das zu erreichen, ist durch Testen, aber Testen kann immer nur Fehler aufzeigen, nicht die Korrektheit des Programms beweisen: Das Nicht-Finden von Fehler lässt nicht auf deren Abwesenheit schließen. Außerdem müssen gefundene Fehler ihre Ursache nicht in der Programmlogik haben, was die Behebung schwierig gestalten kann.

Der Hoare-Kalkül — wie andere Verfahren zur Verifikation von Programmen — wählt eine mathematisch-logische Herangehensweise. Dadurch kann er die Korrektheit eines Programmes tatsächlich beweisen und geht somit über die Möglichkeiten des Testens hinaus. Er überprüft nur die eigentliche Programmlogik und erlaubt so ein leichtes Lokalisieren von Fehlern.

1.2 Gliederung der Arbeit

Diese Arbeit teilt sich in zwei große Teile: Theorie und Beispiele. Im theoretischen Teil werden zunächst eine Prädikatenlogik und eine Programmiersprache definiert, anhand derer wir uns den Regeln des Hoare-Kalküls und der Korrektheit zuwenden. Die Vollständigkeit wird nur gestreift.

Im zweiten Teil werden fünf kurze Beispiele mit Integer-Operationen vorgestellt und im Detail durchgerechnet. Dabei wird auf Verständlichkeit und Überblick Wert gelegt, aber es stehen auch alle Ableitungsbäume zur Verfügung.

1.3 Literaturverweise

Der Hoare-Kalkül wurde von C. A. R. Hoare in [Hoa69] erstmals veröffentlicht. Auf 6 Seiten motiviert er das neue Kalkül, stellt das Zuweisungsaxiom und vier Regeln vor und gibt einen Ausblick, was dadurch alles erreicht werden könnte.

[Hoa69] kann als eine Fortführung des Gedankens in [Flo67] verstanden werden, wo die Basis für eine Formalisierung von Programmen gelegt wird, mit der dann auch Korrektheits-, Äquivalenz- und Terminationsbeweise möglich werden. Im Konkreten weist Floyd jedem Befehl die Eigenschaft zu, von einer vorher geltenden logischen Aussage eine nachher gültige logische Aussage abzuleiten.

Als Einführungswerke in Programmverifikation im Allgemeinen und den Hoare-Kalkül im Speziellen seien [Wan80], [HR04] und [HS92] genannt. [Wan80] ist ein allgemein gutes Buch über Programmieren, das den Stil der Wirlernen-programmieren-Bücher positiv geprägt hat. In Kapitel 6 wendet er sich der Programmverifikation zu und gibt eine leicht verständliche Einführung in ein dem Hoare-Kalkül nachempfundenes System.

In [HR04] Kapitel 4 findet man eine sehr ausführliche und lesenswerte Einführung in den Hoare-Kalkül. Die Autoren gehen dabei im Detail auf partielle und totale Korrektheitsbeweise ein und geben viele Übungsbeispiele.

Hohlfeld und Struckmanns Buch [HS92] handelt nur über Verifikation. Die Autoren geben zunächst eine sehr genaue Abhandlung über den Hoare-Kalkül, wo sie auch nicht mit Beweisen sparen. In der zweiten Hälfte wenden sie sich der Programmiersprache PASCAL zu und stellen Verifikationssysteme konkret dafür vor. Ich habe mich beim Verfassen dieser Arbeit stark auf [HS92] gestützt, muss den Leser aber darauf hinweisen, sich genau in die dortige Notation einzulesen, da sie stark von meiner abweicht.

Wer sich mit den Themen Korrektheit und Vollständigkeit tiefergehend beschäftigen möchte, sei auf [Coo78], [Cla79] und [Jos85] verwiesen. [Coo78] hat für die Vollständigkeit die Bedingung der Expressivität eingeführt. [Cla79] findet Beispiele, wo Expressivität für einen korrekten und vollständigen Hoare-Kalkül nicht ausreicht (Stichwort Rekursion). Clark definiert 5 Kriterien, die diesen verhindern, und diskutiert jene einfacheren Sprachen, die entstehen, wenn eines der Kriterien ausgelassen wird. Für 4 dieser Sprachen wird in [Cla79] bzw. [Old81] gezeigt, dass sie vollständige Hoare-Kalküle haben. [Jos85] zeigt das auch für die fünfte Sprache.

1.4 Notation und Unterschiede zu TIL

In dieser Arbeit werden keine obskuren oder ungewöhnlichen logischen Notationen benutzt ohne erklärt zu werden. Vorab nur eine kleine Konvention: Die logischen Symbole in der in Abschnitt 2 definierten Prädikatenlogik werden sich von den logischen Symbolen für Aussagen *über* die Prädikatenlogik unterscheiden. Im Konkreten wählen wir:

Operator	in der PL	über die PL
Negation	$\neg$	$\sim$
Äquivalenz	$\equiv$	$=, \Longleftrightarrow$
Konjunktion	$\wedge$	$*$
Disjunktion	$\vee$	$+$
Implikation	$\supset$	$\Longrightarrow$

Tabelle 1: Notationskonventionen

Die Operatoren in Tabelle 1 sind dabei nach ihrer Priorität geordnet. Wir werden trotzdem oft Klammern setzen, um Verwirrungen zu vermeiden.

Die Notationsunterschiede zu TIL sind gering. Wir werden Variablenbelegungen mit I bezeichnen, in TIL werden sie ξ genannt. Unser I darf nicht mit dem $\mathcal{I}$ aus TIL verwechselt werden, das neben einer Variablenbelegung auch Domäne und Signatur enthält. Die Menge aller $\mathcal{I}$ wird mit $PINT_\Sigma$ bezeichnet, wir nennen die Menge aller I ENV.

TIL bezeichnet die Prädikatenlogik mit $\mathcal{PF}_\Sigma$, und wertet sie mit der Funktion $val_\mathcal{I}$ aus. In dieser Arbeit wird die Prädikatenlogik mit $PL(\mathcal{D})$ bezeichnet und von $\mathcal{M}_{PL}$ ausgewertet.

Des weiteren sei noch darauf hingewiesen, dass die Programmiersprache "Assignment Language" (AL) in TIL nicht ganz der hier benutzten "Toy Programming Language" (TPL) entspricht. TPL ist um den leeren Befehl **skip** erweitert und hat sonst nur leichte syntaktische Abweichungen.

2 Prädikatenlogik

Es wird vorausgesetzt, dass der Leser mit den Grundzügen der Prädikatenlogik bereits vertraut ist. Hier werden kurz die Symbole und Definitionen angegeben, die sich durch den Rest der Arbeit ziehen werden.

2.1 Datentyp

Wir benötigen ein allgemeines Konzept von Datentypen, das variabel sein soll, damit unsere restlichen Definitionen möglichst allgemein bleiben. Wir

definieren einen Datentyp als ein Quadrupel $\mathcal{D} = \langle D, F, P, K \rangle$, bestehend aus

- der Domäne (nicht leer)
- einer endlichen Menge von Funktionen vom Typ $D^n \to D^m$
- einer endlichen Menge von totalen Prädikaten vom Typ $D^n \to \{\mathbf{true}, \mathbf{false}\}$
- Konstanten aus der Domäne

Zu den Mengen F, P, K gibt es die Signaturmengen $FS(\mathcal{D}), PS(\mathcal{D}), KS(\mathcal{D})$, die die den Funktionen, Prädikaten und Konstanten zugehörigen Symbole bezeichnen. Die Funktion ϕ bildet Signatursymbole auf die im Datentyp spezifizierten Prädikate, Funktionen und Konstanten ab. Wir bezeichnen $\phi_{\mathcal{D}} : FS(\mathcal{D}) \to F$, $\phi_{\mathcal{D}} : PS(\mathcal{D}) \to P$, $\phi_{\mathcal{D}} : KS(\mathcal{D}) \to K$ der Einfachkeit halber alle als $\phi_{\mathcal{D}}$.

2.2 Terme über Datentypen

Um mit Variablen hantieren zu können, führen wir Interpretationen I ein, die Variablensymbole auf Domänenelemente abbilden. Die Menge ENV enthält alle möglichen Interpretationen, IVS alle Individuenvariablensymbole. $ENV : IVS \to D$.

2.2.1 Syntax von Termen über Datentypen

Die Menge $\mathcal{T}(\mathcal{D})$ aller Terme über dem Datentyp $\mathcal{D}$ enthält

(T1) alle $v \in IVS$

(T2) alle $k \in KS(\mathcal{D})$

(T3) und alle Terme $f'(t_1, \ldots, t_n)$, wenn $f' \in FS_n(\mathcal{D})$ und $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{D})$.

2.2.2 Semantik von Termen über Datentypen

Die Menge $\mathcal{T}(\mathcal{D})$ wird interpretiert von der Semantikfunktion $\mathcal{M}_{\mathcal{T}} : ENV \times \mathcal{T}(\mathcal{D}) \to D$, wobei

(MT1) Variablen der gegebenen Interpretation entsprechend gedeutet werden: $\mathcal{M}_{\mathcal{T}}(I, v) = I(v)$

(MT2) Konstanten dem Datentyp entsprechend: $\mathcal{M}_{\mathcal{T}}(I, c') = c$
mit $c' \in KS(\mathcal{D})$, $c \in K(\mathcal{D})$, $\phi_{\mathcal{D}}(c') = c$.

(MT3) und Funktionen auf ihre Argumente einzeln angewendet werden:
$\mathcal{M}_{\mathcal{T}}(I, f'(t_1, \ldots, t_n)) = f(\mathcal{M}_{\mathcal{T}}(I, t_1), \ldots, \mathcal{M}_{\mathcal{T}}(I, t_n))$
mit $f' \in FS_n$, $f \in F_n$, $\phi_{\mathcal{D}}(f') = f$.

Das Ergebnis einer Auswertung von $\mathcal{M}_{\mathcal{T}}$ ist ein Domänenelement.

2.3 Syntax von PL

Als prädikatenlogische Formeln über dem Datentyp $\mathcal{D}$ erlauben wir die folgenden:

(PL1) $p'(t_1,\ldots,t_n) \in PL(\mathcal{D})$, wenn $p' \in PS_n(\mathcal{D})$ und $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{D})$

(PL2) $(F \circ G) \in PL(\mathcal{D})$, wenn $F, G \in PL(\mathcal{D})$ und $\circ \in \{\wedge, \vee, \supset, \equiv\}$

(PL3) $\neg F \in PL(\mathcal{D})$, wenn $F \in PL(\mathcal{D})$

(PL4) $\mathsf{Q}vF \in PL(\mathcal{D})$, wenn $F \in PL(\mathcal{D})$, $v \in IVS$ und $\mathsf{Q} \in \{\forall, \exists\}$

2.4 Semantik von PL

Die Funktion $\mathcal{M}_{PL} : ENV \times PL(\mathcal{D}) \to \{\mathbf{true}, \mathbf{false}\}$ wertet einen prädikatenlogischen Ausdruck in Bezug auf eine Variableninterpretation aus und bestimmt, ob er wahr oder falsch ist.

(MPL1) $\mathcal{M}_{PL}(I, p'(t_1,\ldots,t_n)) = p(\mathcal{M}_{\mathcal{T}}(I,t_1),\ldots,\mathcal{M}_{\mathcal{T}}(I,t_n))$
mit $p' \in PS_n(\mathcal{D})$, $p \in P_n(\mathcal{D})$, $\phi_{\mathcal{D}}(p') = p$.

(MPL2) $\mathcal{M}_{PL}(I, (F \circ' G)) = \mathcal{M}_{PL}(I,F) \circ \mathcal{M}_{PL}(I,G)$
wobei $\circ' \in \{\wedge, \vee, \supset, \equiv\}$ und $\circ \in \{*, +, \Longrightarrow, \Longleftrightarrow\}$.

(MPL3) $\mathcal{M}_{PL}(I, \neg F) = \sim \mathcal{M}_{PL}(I,F)$

(MPL4) $\mathcal{M}_{PL}(I, (\forall v)F) \Longleftrightarrow$ für alle I', die sich höchstens in v von I unterscheiden, gilt $\mathcal{M}_{PL}(I', F)$.

(MPL5) $\mathcal{M}_{PL}(I, (\exists v)F) \Longleftrightarrow$ für mindestens ein I', das sich höchstens in v von I unterscheidet, gilt $\mathcal{M}_{PL}(I', F)$.

Das Ergebnis einer Auswertung von $\mathcal{M}_{PL}$ ist also ein Wahrheitswert.

3 Die Programmiersprache TPL

Die Programmiersprache Toy Programming Language (TPL) ist eine reduzierte imperative Programmiersprache mit ähnlicher Syntax wie PASCAL. Sie soll Zuweisungen, Verzweigungen und Schleifen ausdrücken können und ist wie folgt definiert.

3.1 Syntax von TPL

Die formale Sprache $Toy(\mathcal{D})$ besteht aus:

- den Schlüsselwörtern von TPL:
  $T = \{\mathbf{skip}, \leftarrow, \mathbf{begin}, ;, \mathbf{end}, \mathbf{if}, \mathbf{then}, \mathbf{else}, \mathbf{fi}, \mathbf{while}, \mathbf{do}, \mathbf{od}\}$

- Variablensymbolen IVS
- Termen $\mathcal{T}(\mathcal{D})$ für Variablenzuweisungen
- prädikatenlogischen Ausdrücken $PL(\mathcal{D})$ für Pfadentscheidungen

Dadurch ergibt sich $Toy(\mathcal{D}) \subseteq (T \cup IVS \cup \mathcal{T}(\mathcal{D}) \cup PL(\mathcal{D}))^*$. $Toy(\mathcal{D})$ enthält aber nur Zeichenketten, die sich aus den folgenden induktiven Regeln ableiten lassen.

(Toy1) $\mathbf{skip} \in Toy(\mathcal{D})$

(Toy2) $v \leftarrow t \in Toy(\mathcal{D})$, wenn $v \in IVS$ und $t \in \mathcal{T}(\mathcal{D})$

(Toy3) $\mathbf{begin}\ \alpha\ ;\ \beta\ \mathbf{end} \in Toy(\mathcal{D})$, wenn $\alpha, \beta \in Toy(\mathcal{D})$

(Toy4) $\mathbf{if}\ E\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta\ \mathbf{fi} \in Toy(\mathcal{D})$, wenn $\alpha, \beta \in Toy(\mathcal{D})$ und $E \in PL_0(\mathcal{D})$[1]

(Toy5) $\mathbf{while}\ E\ \mathbf{do}\ \alpha\ \mathbf{od} \in Toy(\mathcal{D})$, wenn $\alpha \in Toy(\mathcal{D})$ und $E \in PL_0(\mathcal{D})$

3.2 Semantik von TPL

Um Ausdrücke in $Toy(\mathcal{D})$ auswerten zu können, definieren wir die Semantikfunktion $\mathcal{M}_{Toy}$ wie folgt. Sie erhält zwei Argumente: eine Variablenbelegung $\in ENV$, in Bezug auf welche die Auswertung erfolgen soll, und ein Programm bzw. einen Programmteil $\in Toy(\mathcal{D})$. Sie gibt die Variablenbelegung nach Durchführung des Programms zurück.

(MToy1) $\mathcal{M}_{Toy}(I, \mathbf{skip}) = I$

(MToy2) $\mathcal{M}_{Toy}(I, v \leftarrow t) = I'$
wobei $I'(v) = \mathcal{M}_{\mathcal{T}}(I, t)$ und $I'(u) = I(u)$ für alle $u \in IVS, u \neq v$

(MToy3) $\mathcal{M}_{Toy}(I, \mathbf{begin}\ \alpha\ ;\ \beta\ \mathbf{end}) = \mathcal{M}_{Toy}(\mathcal{M}_{Toy}(I, \alpha), \beta)$

(MToy4) $\mathcal{M}_{Toy}(I, \mathbf{if}\ E\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta\ \mathbf{fi}) = \begin{cases} \mathcal{M}_{Toy}(I, \alpha) & \text{falls } \mathcal{M}_{PL}(I, E) = \mathbf{true} \\ \mathcal{M}_{Toy}(I, \beta) & \text{falls } \mathcal{M}_{PL}(I, E) = \mathbf{false} \end{cases}$

(MToy5) $\mathcal{M}_{Toy}(I, \mathbf{while}\ E\ \mathbf{do}\ \alpha\ \mathbf{od}) = \begin{cases} \mathcal{M}_{Toy}(\mathcal{M}_{Toy}(I, \alpha), \mathbf{while}\ E\ \mathbf{do}\ \alpha\ \mathbf{od}) & \\ & \text{falls } \mathcal{M}_{PL}(I, E) = \mathbf{true} \\ I & \text{falls } \mathcal{M}_{PL}(I, E) = \mathbf{false} \end{cases}$

[1] $PL_0(\mathcal{D})$ bezeichnet die quantorenfreien Ausdrücke in $PL(\mathcal{D})$

4 Korrektheitsaussagen

Die Hoare'sche Logik bietet eine Notation, in der Aussagen über die Korrektheit von Programmen gemacht werden können. Dabei gibt man an, was nach Ausführung des Programmes gelten muss, sofern davor ein bestimmter Systemzustand gegolten hat. Mit den Regeln des Hoare-Kalküls in Abschnitt 5 werden diese Aussagen dann bewiesen.

Das Hoare-Kalkül bedient sich dabei einer sehr allgemeinen und mächtigen Technik: der *Problemreduktion*. Um zu zeigen, dass ein komplexes Programm gewisse Anforderungen erfüllt, genügt es, zu zeigen, dass es in Teile zerlegt werden kann, die zusammen die gestellten Anforderungen erfüllen. Man denkt also *vom Großen ins Kleine*. Das sollte nicht vergessen werden, vor allem, da Ableitungen im Hoare-Kalkül nach oben verzweigend notiert werden. Diese Bäume sind also von unten nach oben zu lesen.

Ein Hoare-Tripel ist eine Zeichenkette der Form $P\ \{\pi\}\ Q$ mit $P, Q \in PL(\mathcal{D})$ und $\pi \in Toy(\mathcal{D})$. Dabei wird P die *Vorbedingung (Precondition)* genannt und Q die *Nachbedingung (Postcondition)*. Das Paar (P, Q) wird auch *Spezifikation von* π genannt. π ist das zu überprüfende Programm unserer gewählten Programmiersprache TPL. π darf keine Variable enthalten, die in P oder Q quantifiziert vorkommt. Ist das doch der Fall, muss Variablenumbenennung/-ersetzung erfolgen.

Die Funktion $\mathcal{M}_*$ definiert die Semantik einer partiellen Korrektheitsaussage.

$$\mathcal{M}_*(P\ \{\pi\}\ Q) = \begin{cases} \textbf{true} & \text{falls } \forall I \in ENV \text{ gilt, dass:} \\ & \quad \text{wenn } \mathcal{M}_{PL}(I, P) = \textbf{true} \\ & \quad \text{und } \mathcal{M}_{Toy}(I, \pi) \text{ ist definiert,} \\ & \quad \text{dann } \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \pi), Q) = \textbf{true} \\ \textbf{false} & \text{sonst} \end{cases}$$

Wir beobachten, dass die Nachbedingung nur dann gültig sein muss, wenn die Auswertung von π, also $\mathcal{M}_{Toy}(I, \pi)$, definiert ist, das heißt, wenn π terminiert. Diese Beschränkung wird mit *partiell* ausgedrückt. Wenn man die Termination in den Korrektheitsbeweis miteinschließt, spricht man von *totaler* Korrektheit. Wir gehen auf diesen Unterschied in Abschnitt 6 näher ein.

Da eine Auswertung von $\mathcal{M}_*(P\ \{\pi\}\ Q)$ alle möglichen Variablenbelegungen überprüfen muss, ist sie relativ aufwendig. Mit den im folgenden dargestellten Regeln des Hoare-Kalküls kann das viel einfacher erfolgen.

5 Regeln des Hoare-Kalküls

Im folgenden betrachten wir die verschiedenen Regeln des Hoare-Kalküls. Ihre Korrektheit wird in Abschnitt 6 behandelt. Wir bezeichnen die Menge der Korrektheitsaussagen über dem Datentyp $\mathcal{D}$ mit $\mathcal{H}(\mathcal{D})$. Statt der

ausführlichen Beschreibung

Wenn A und B in $\mathcal{H}(\mathcal{D})$ sind, ist auch C in $\mathcal{H}(\mathcal{D})$.

verwenden wir die Abkürzung $\vdash_{\mathcal{H}(\mathcal{D})} \frac{A \quad B}{C}$. Das Symbol $\vdash_{\mathcal{H}(\mathcal{D})}$ drückt dabei aus, dass es sich um eine syntaktisch gültige Ableitung in $\mathcal{H}(\mathcal{D})$ handelt.

5.1 Leere Anweisung

Die leere Anweisung bestehend aus dem Schlüsselwort **skip** ist trivial und kann als Axiom verstanden werden. Unabhängig von den Vorbedingungen können sie auch als Nachbedingungen gelten.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{}{P\ \{\mathbf{skip}\}\ P}\ (\text{HA1}) \qquad \text{für alle } P \in PL(\mathcal{D})$$

5.2 Zuweisung

5.2.1 Vorwärtsverkettung

Wenn einer neu eingeführten Variablen ein Term zugewiesen wird, in dem die neue Variable auch nicht vorkommt, gilt die Gleichsetzung als weitere Nachbedingung. Dies ist ein Axiom.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{}{P\ \{v \leftarrow t\}\ P \wedge v = t}\ (\text{HA2})$$

Wenn v in t oder P vorkommt, ist es trickreicher. In der Nachbedingung müssen alle Vorkommen von v durch t ersetzt werden (wenn v in t vorkommt, gilt also der alte Wert), und es muss gelten, dass Q mit Substitution weiterhin aus P folgt.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{P \supset Q\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right]}{P\ \{v \leftarrow t\}\ Q}\ (\text{H1}) \qquad \begin{array}{l}\text{für alle } P, Q \in PL(\mathcal{D}),\ v \in IVS,\\ t \in \mathcal{T}(\mathcal{D})\end{array}$$

5.2.2 Rückwärtsverkettung

Wenn die Nachbedingungen einer Zuweisung bekannt sind, lassen sich sehr leicht Vorbedingungen definieren. Alles, was für den neuen Wert von v (also t) gilt, muss auch vorher schon für t gegolten haben. Wir können also alle Vorkommen von v durch t ersetzen.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{}{P\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right]\ \{v \leftarrow t\}\ P}\ (\text{HA3}) \qquad \begin{array}{l}\text{für alle } P \in PL(\mathcal{D}),\ v \in IVS,\\ t \in \mathcal{T}(\mathcal{D})\end{array}$$

5.3 Konkatenation / Hintereinanderausführung

Möchte man zwei Anweisungsblöcke hintereinander ausführen, von denen die Nachbedingung des einen die Vorbedingung des anderen ist/sein kann, können sie in einen **begin-end**-Block zusammengefügt werden. Die Zwischenbedingung kann dabei entfallen, sodass die Vorbedingung des ersten und die Nachbedingung des zweiten als solche übernommen werden.

Anders betrachtet kann man eine Konkatenation in zwei Teile zerlegen, wenn man eine geeignete Aussage über den Zustand zwischen ihnen treffen kann, der als Nachbedingung der ersten und Vorbedingung der zweiten Ausführung verstanden werden kann.

Die Regel kann deshalb so notiert werden:

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{P\ \{\alpha\}\ R \quad R\ \{\beta\}\ Q}{P\ \{\textbf{begin}\ \alpha\ ;\ \beta\ \textbf{end}\}\ Q}\ (\text{H2}) \qquad \begin{array}{l}\text{wobei}\ \ P, R, Q \in PL(\mathcal{D}),\\ \alpha, \beta \in Toy(\mathcal{D})\end{array}$$

5.4 Bedingungen

Sind die Vorbedingungen zweier Ausdrücke komplementär und haben sie dieselbe Nachbedingung, so können sie zu einem **if-else**-Konstrukt zusammengefügt werden. Können aus den beiden Vorbedingungen Gemeinsamkeiten extrahiert werden, werden diese zur neuen Vorbedingung. Die Definition der Komplementarität E wird als "Bedingung" in den **if**-Ausdruck übernommen, sodass α E erfüllt.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{P \wedge E\ \{\alpha\}\ Q \quad P \wedge \neg E\ \{\beta\}\ Q}{P\ \{\textbf{if}\ E\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ \textbf{fi}\}\ Q}\ (\text{H3}) \qquad \begin{array}{l}\text{wobei}\ \ P, Q \in PL(\mathcal{D}),\\ E \in PL_0(\mathcal{D}),\ \alpha, \beta \in\\ Toy(\mathcal{D})\end{array}$$

5.5 Schleifen

Ist die Nachbedingung eines Ausdrucks eine Untermenge der Vorbedingung ($P \wedge E \supset P$), kann das als Schleife verstanden werden. Sollte nach Ausführung die Spezifität der Vorbedingung noch zusätzlich gelten, kann er erneut durchlaufen werden. Wir notieren das mit **while**. Die Nachbedingung des **while**-Ausdrucks muss allerdings $\neg E$ enthalten, damit dieser ordentlich beendet ist.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{P \wedge E\ \{\alpha\}\ P}{P\ \{\textbf{while}\ E\ \textbf{do}\ \alpha\ \textbf{od}\}\ P \wedge \neg E}\ (\text{H4'}) \qquad \begin{array}{l}\text{wobei}\ P \in PL(\mathcal{D}),\ E \in\\ PL_0(\mathcal{D}),\ \alpha \in Toy(\mathcal{D})\end{array}$$

Alternativ können Schleifen unter Applikation der Konsequenzregel unten (H5) auch durch Schleifeninvarianten verallgemeinert dargestellt werden.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{P \supset Inv \quad Inv \wedge E\ \{\alpha\}\ Inv \quad (Inv \wedge \neg E) \supset Q}{P\ \{\textbf{while}\ E\ \textbf{do}\ \alpha\ \textbf{od}\}\ Q}\ \text{(H4)}$$

wobei $P, Q, Inv \in PL(\mathcal{D})$, $E \in PL_0(\mathcal{D})$, $\alpha \in Toy(\mathcal{D})$

5.6 Folgerung / Konsequenz

Vorbedingungen können durch allgemeinere Ausdrücke ersetzt werden, Nachbedingungen durch spezifischere.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{P \supset Q \quad Q\ \{\alpha\}\ R \quad R \supset S}{P\ \{\alpha\}\ S}\ \text{(H5)} \qquad \begin{array}{l}\text{wobei } P, Q, R, S \in PL(\mathcal{D}),\\ \alpha \in Toy(\mathcal{D})\end{array}$$

6 Die Korrektheit des Hoare-Kalküls

In diesem Abschnitt zeigen wir, dass die oben vorgestellten Ableitungsregeln korrekt (engl. *sound*) sind. Das heißt, jede ableitbare Korrektheitsaussage ist auch tatsächlich gültig. Korrektheit ist neben Vollständigkeit (siehe Abschnitt 7) eine wichtige Eigenschaft von Kalkülen.

Wie schon oben in Abschnitt 4 angedeutet, muss man in Bezug auf die Korrektheit des Hoare-Kalküls zwischen der partiellen und totalen Korrektheit unterscheiden.

Die partielle Korrektheit besagt, dass wenn für ein Programm π die Vorbedingung P gilt und das Programm terminiert, die Nachbedingung Q gelten muss. Formal:

$$\begin{aligned}\forall I \in ENV: \quad & \mathcal{M}_{PL}(I, P)\ *\ \mathcal{M}_{Toy}(I, \pi) \text{ ist definiert}\\ & \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \pi), Q)\end{aligned}$$

In Abschnitt 6.1 zeigen wir für die bereits vorgestellten Ableitungsregeln, dass sie partielle Korrektheit erhalten.

Im Gegensatz zur partiellen Korrektheit gilt es auch, die totale Korrektheit zu betrachten. Diese erweitert die partielle Korrektheit um den Terminationsbeweis. Sie besagt also: Wenn die Vorbedingung P gilt, muss π terminieren und die Nachbedingung Q gelten. Formal:

$$\begin{aligned}\forall I \in ENV: \quad & \mathcal{M}_{PL}(I, P) \Longrightarrow\\ & \mathcal{M}_{Toy}(I, \pi) \text{ ist definiert}\ *\ \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \pi), Q)\end{aligned}$$

Wir gehen auf die für totale Korrektheit notwendigen Adaptionen in Abschnitt 6.2 ein.

6.1 (Partielle) Korrektheit

Man spricht von einer partiellen Korrektheitsüberprüfung, wenn die Termination gegeben sein muss, um eine Aussage über den Zusammenhang von Vorbedingung und Nachbedingung zu treffen. In anderen Worten: Terminiert der Code nicht, gilt auf jeden Fall partielle Korrektheit. Wir lassen in den folgenden Beweisen deshalb die Voraussetzung von Definiertheit beiseite.

Eine Ableitung im Hoare-Kalkül kann als Baum bzw. als Folge dargestellt werden (eine Aussage hat eine oder mehrere Bedingungen). Die einzelnen Folgenglieder sind dabei entweder Axiome des Hoare-Kalküls, prädikatenlogische Formeln oder mit Hilfe von Ableitungsregeln des Hoare-Kalküls aus vorhergegangenen Folgengliedern berechnet.

6.1.1 Leere Anweisung

(HA1) ist ein Axiom. Wir müssen zeigen, dass für alle $P \in PL(\mathcal{D})$ $\mathcal{M}_*(P\ \{\mathbf{skip}\}\ P) = \mathbf{true}$ gilt.

Wir erinnern uns, dass wir in (MToy1) $\mathcal{M}_{Toy}(I, \mathbf{skip}) = I$ definiert haben. Wenn wir also $\mathcal{M}_*(P\ \{\mathbf{skip}\}\ P)$ auswerten, erhalten wir für alle $I \in ENV$:

$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \mathbf{skip}), P)$$
$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I, P)$$

was trivialerweise für alle $P \in PL(\mathcal{D})$ wahr ist.

6.1.2 Zuweisung

(HA2) ist ebenfalls ein Axiom. Wir müssen für alle $P \in PL(\mathcal{D})$, $v \in IVS$, $t \in \mathcal{T}(\mathcal{D})$ zeigen, dass,

(i) sofern v weder in P noch in t vorkommt, $\mathcal{M}_*(P\ \{v \leftarrow t\}\ P \wedge v = t)$ immer gilt;

(ii) falls $P \supset Q\binom{t}{v}$, $\mathcal{M}_*(P\ \{v \leftarrow t\}\ Q) = \mathbf{true}$ gilt;

(iii) und $\mathcal{M}_*(P\binom{t}{v}\ \{v \leftarrow t\}\ P)$ immer gilt.

Unabhängigkeit. Unter der Bedingung, dass v weder in P noch in t vorkommt, gilt für die Korrektheit von $P\ \{v \leftarrow t\}\ P \wedge v = t$

$$\begin{aligned}
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, v \leftarrow t), P \wedge v = t)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I', P \wedge v = t)\\
&\qquad\qquad \text{mit } I'(v) = \mathcal{M}_{\mathcal{T}}(I, t), \forall u \neq v : I'(u) = I(u)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I', P) * \mathcal{M}_{PL}(I', v = t)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I', P) * \mathcal{M}_{\mathcal{T}}(I', v) = \mathcal{M}_{\mathcal{T}}(I', t)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I', P) * I'(v) = \mathcal{M}_{\mathcal{T}}(I', t)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I', P) * \mathbf{true}
\end{aligned}$$

Da v nicht in P und t vorkommt, unterscheidet sich die Auswertung von P bzw. t unter den Interpretationen I und I' nicht.

Vorwärtsverkettung. Wir zeigen die Gültigkeit der Hoare-Regel (H1), indem wir die Korrektheitsaussage auf ihre Bedingung zurückführen.

$$\begin{aligned}
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, v \leftarrow t), Q)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I', Q)\\
&\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I, Q\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right])
\end{aligned}$$

mit $I'(v) = \mathcal{M}_{\mathcal{T}}(I, t), \forall u \neq v : I'(u) = I(u)$. Die Gleichsetzung von $\mathcal{M}_{PL}(I', Q)$ und $\mathcal{M}_{PL}(I, Q\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right])$ erlaubt uns Lemma 6.1. Somit haben wir die Korrektheitsaussage $P\ \{v \leftarrow t\}\ Q$ auf $P \supset Q\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right]$ zurückgeführt.

Lemma 6.1. *Sei $v \in IVS$, $t \in \mathcal{T}(\mathcal{D})$, $I, I' \in ENV$ mit $I' = \mathcal{M}_{Toy}(I, v \leftarrow t)$, $P \in PL(\mathcal{D})$, wobei in P weder v noch die Variablen in t quantifiziert vorkommen. Dann gilt $\mathcal{M}_{PL}(I', P) = \mathcal{M}_{PL}(I, P\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right])$.*

Rückwärtsverkettung. Ist die Nachbedingung einer Zuweisung bekannt, kann sehr einfach eine Aussage über die Vorbedingung getroffen werden: Was in der Nachbedingung für $v_{neu} = t$ gilt, muss in der Vorbedingung, wo $v_{alt} \neq v_{neu}$, dennoch für t gelten. Wir ersetzen also alle Vorkommen von v in der Nachbedingung durch t, um eine Vorbedingung zu erhalten. Die Korrektheit von $P\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right]\ \{v \leftarrow t\}\ P$ zeigen wir mit

$$\begin{aligned}
&\mathcal{M}_{PL}(I, P\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right]) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, v \leftarrow t), P)\\
&\mathcal{M}_{PL}(I, P\left[\begin{smallmatrix} t \\ v \end{smallmatrix}\right]) \Longrightarrow \mathcal{M}_{PL}(I', P)
\end{aligned}$$

was nach Lemma 6.1 gilt.

6.1.3 Konkatenation / Hintereinanderausführung

Durch die Vorbedingung $P\ \{\alpha\}\ R$ wissen wir, dass für alle $I_1 \in ENV$ gilt:

$$\begin{aligned}\mathcal{M}_{PL}(I_1, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I_1, \alpha), R)\\ \mathcal{M}_{PL}(I_1, P) &\Longrightarrow \mathcal{M}_{PL}(I_1', R)\\ &\qquad \text{mit } I_1' = \mathcal{M}_{Toy}(I_1, \alpha)\end{aligned}$$

Durch die Vorbedingung $R\ \{\beta\}\ Q$ wissen wir, dass für alle $I_2 \in ENV$ gilt, dass

$$\begin{aligned}\mathcal{M}_{PL}(I_2, R) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I_2, \alpha), Q)\\ \mathcal{M}_{PL}(I_2, R) &\Longrightarrow \mathcal{M}_{PL}(I_2', Q)\\ &\qquad \text{mit } I_2' = \mathcal{M}_{Toy}(I_2, \beta)\end{aligned}$$

Da wir I_1' als Spezialfall von I_2 verstehen können, wissen wir, dass

$$\begin{array}{rcl}\mathcal{M}_{PL}(I_1, P) \Longrightarrow & \mathcal{M}_{PL}(I_1', R) \Longrightarrow & \mathcal{M}_{PL}(I_2', Q)\\ \mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), R) & \Longrightarrow & \mathcal{M}_{PL}(\mathcal{M}_{Toy}(\mathcal{M}_{Toy}(I, \alpha), \beta), Q)\end{array}$$

Durch Ausschluss des Mittelglieds erreichen wir $P\ \{\textbf{begin}\ \alpha\ ;\ \beta\ \textbf{end}\}\ Q$ und damit die zu zeigende Korrektheit von (H2):

$$\begin{aligned}\mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(\mathcal{M}_{Toy}(I, \alpha), \beta), Q)\\ \mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \textbf{begin}\ \alpha\ ;\ \beta\ \textbf{end}), Q)\end{aligned}$$

6.1.4 Bedingungen

Wenn wir (H3) auf Korrektheit untersuchen wollen, müssen wir eine Fallunterscheidung treffen (wir erinnern uns an die Semantik-Definition von $\mathcal{M}_{Toy}(I, \textbf{if}\ E\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ \textbf{fi})$).

Fall 1: $\mathcal{M}_{PL}(I, E) = \textbf{true}$

$$\begin{aligned}\mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \textbf{if}\ E\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ \textbf{fi}), Q)\\ \mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), Q)\end{aligned}$$

Durch einfache Umformung erhalten wir also aus

$$\begin{aligned}\mathcal{M}_{PL}(I, E) &\Longrightarrow (\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), Q))\\ (\mathcal{M}_{PL}(I, E) * \mathcal{M}_{PL}(I, P)) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), Q)\end{aligned}$$

was genau der ersten Bedingung $P \wedge E\ \{\alpha\}\ Q$ entspricht.

Fall 2: $\mathcal{M}_{PL}(I, E) = \textbf{false}$

$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \textbf{if } E \textbf{ then } \alpha \textbf{ else } \beta \textbf{ fi}), Q)$$
$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \beta), Q)$$

Parallel zum ersten Fall formen wir um und erhalten

$$\sim\mathcal{M}_{PL}(I, E) \Longrightarrow (\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \beta), Q))$$
$$(\sim\mathcal{M}_{PL}(I, E) * \mathcal{M}_{PL}(I, P)) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \beta), Q),$$

also die zweite Bedingung $P \wedge \neg E \ \{\beta\} \ Q$.

6.1.5 Schleifen

Betrachten wir zunächst die Regel (H4'). Aus der Voraussetzung $P \wedge E \ \{\alpha\} \ P$ können wir für alle $I \in ENV$ das Folgende ableiten:

$$(\mathcal{M}_{PL}(I, P) * \mathcal{M}_{PL}(I, E)) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), P)$$
$$(\mathcal{M}_{PL}(I, P) * \mathcal{M}_{PL}(I, E)) \Longrightarrow \mathcal{M}_{PL}(I', P)$$

Wenn $\mathcal{M}_{PL}(I, E) = \textbf{false}$ ist die Aussage trivialerweise wahr. Ist $\mathcal{M}_{PL}(I, E) = \textbf{true}$, gilt im Besonderen $\mathcal{M}_{PL}(I, P) = \mathcal{M}_{PL}(I', P)$ für $I' = \mathcal{M}_{Toy}(I, \alpha)$.

Durch die Semantik-Defintion von $\mathcal{M}_{Toy}(I, \textbf{while } E \textbf{ do } \alpha \textbf{ od})$ müssen wir eine Fallunterscheidung treffen.

Fall 1: $\mathcal{M}_{PL}(I, E) = \textbf{true}$

$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \textbf{while } E \textbf{ do } \alpha \textbf{ od}), P \wedge \neg E)$$
$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(\mathcal{M}_{Toy}(I, \alpha), \textbf{while } E \textbf{ do } \alpha \textbf{ od}), P \wedge \neg E)$$
$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I', \textbf{while } E \textbf{ do } \alpha \textbf{ od}), P \wedge \neg E)$$
$$\text{mit } I' = \mathcal{M}_{Toy}(I, \alpha)$$

Die letzten beiden Schritte werden wiederholt, bis die Schleife mit $\mathcal{M}_{PL}(I'', E) = \textbf{false}$ terminiert. I'' ist somit die resultierende Interpretation von $\mathcal{M}_{Toy}(I, \textbf{while } E \textbf{ do } \alpha \textbf{ od})$.

$$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I'', P \wedge \neg E)$$
$$\mathcal{M}_{PL}(I, P) \Longrightarrow (\mathcal{M}_{PL}(I'', P) * \sim\mathcal{M}_{PL}(I'', E))$$
$$(\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I'', P)) * (\mathcal{M}_{PL}(I, P) \Longrightarrow \sim\mathcal{M}_{PL}(I'', E))$$

Der erste Teil dieser Konjunktion ist wahr, da wir wissen, dass $\forall I : \mathcal{M}_{PL}(I, P) * \mathcal{M}_{PL}(I, E) \Longrightarrow \mathcal{M}_{PL}(I', P)$ gilt, und da $\mathcal{M}_{PL}(I', E)$ für alle I' vor I'' (das umfasst auch I) durch die Definition von I'' gegeben ist.

Der zweite Teil ist wahr, da wir $\mathcal{M}_{PL}(I'', E) = \textbf{false}$ definiert haben, und somit das hintere Glied der Implikation **true** ist.

Fall 2: $\mathcal{M}_{PL}(I, E) = \textbf{false}$

$$\begin{aligned}
\mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \textbf{while } E \textbf{ do } \alpha \textbf{ od}), P \wedge \neg E) \\
\mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(I, P \wedge \neg E) \\
\mathcal{M}_{PL}(I, P) &\Longrightarrow (\mathcal{M}_{PL}(I, P) * {\sim}\mathcal{M}_{PL}(I, E)) \\
(\mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(I, P)) * (\mathcal{M}_{PL}(I, P) \Longrightarrow {\sim}\mathcal{M}_{PL}(I, E))
\end{aligned}$$

$\mathcal{M}_{PL}(I, P) \Longrightarrow \mathcal{M}_{PL}(I, P)$ ist eine Tautologie.

$\mathcal{M}_{PL}(I, P) \Longrightarrow {\sim}\mathcal{M}_{PL}(I, E)$ ist wahr, da ${\sim}\mathcal{M}_{PL}(I, E) = \textbf{true}$ gilt.

Der Korrektheitsbeweis der alternativen Darstellung (H4) folgt aus dem Korrektheitsbeweis von (H5) in Abschnitt 6.1.6.

6.1.6 Folgerung / Konsequenz

Wir zeigen, dass mit $P, Q, R, S \in PL$ und $\alpha \in Toy(\mathcal{D})$ aus den bekannten Formeln $P \supset Q$, $R \supset S$ und $Q\ \{\alpha\}\ R$ der neue Hoare-Kalkül $P\ \{\alpha\}\ S$ hergeleitet werden kann. Dazu betrachten wir zunächst $Q\ \{\alpha\}\ R$ und führen zur Vereinfachung die Abkürzungen Φ und Ψ ein.

$$\begin{aligned}
\mathcal{M}_{PL}(I, Q) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), R) \\
\Phi(Q) &\Longrightarrow \Psi(R)
\end{aligned}$$

Was wir also zeigen müssen, ist

$$\frac{P \supset Q \quad \Phi(Q) \Longrightarrow \Psi(R) \quad R \supset S}{\Phi(P) \Longrightarrow \Psi(S)}$$

Da für die beiden prädikatenlogische Formeln P, Q unabhängig von ihrer Interpretation gilt, dass $P \supset Q$, gilt auch $\Phi(P) \Longrightarrow \Phi(Q)$ (d.h. unter der Interpretation I). Ebenso gilt wegen $R \supset S$ auch $\Psi(R) \Longrightarrow \Psi(S)$.

Die Ableitung[2]

$$\frac{\Phi(P) \Longrightarrow \Phi(Q) \quad \Phi(Q) \Longrightarrow \Psi(R) \quad \Psi(R) \Longrightarrow \Psi(S)}{\begin{aligned}\Phi(P) &\Longrightarrow \Psi(S) \\ \mathcal{M}_{PL}(I, P) &\Longrightarrow \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \alpha), S)\end{aligned}}$$

ist durch die Transitivität der Implikation korrekt.

6.2 Totale Korrektheit

Totale Korrektheit erweitert partielle Korrektheit um einen Terminationsbeweis. In unserer Programmiersprache TPL gibt es nur eine Konstruktion,

[2]Der waagrechte Strich hier ist nicht mit einer Hoare-Regel zu verwechseln, sondern in seiner allgemeinen Bedeutung zu verstehen: Wenn oben, dann unten.

aus der Nicht-Termination entstehen könnte: Schleifen. (In anderen Programmiersprachen wäre etwa auch Rekursion eine solche Ursache.)

Um von einer Schleife zu beweisen, dass sie terminiert, muss ein Ausdruck (auch "Variante" genannt) zu finden sein, der die folgenden Eigenschaften aufweist:

- Der Datentyp des Ausdrucks ist eine natürliche Zahl.
- Der Wert des Ausdrucks nimmt mit jedem Schleifendurchlauf streng monoton ab.
- Solange die Schleifeninvariante gilt, ist der Wert des Ausdrucks ≥ 0. Das heißt, die Variante ist vor, während und nach der Schleife nichtnegativ.

Wenn der Datentyp unserer Programmiersprache keine natürlichen Zahlen beinhaltet, muss eine Reduktion dieses Datentyps auf die natürlichen Zahlen erfolgen. Beispielsweise könnte man bei $\mathcal{D} = \textit{Datentyp der Strings}$ die Länge des Strings verwenden und bei $\mathcal{D} = \textit{Datentyp der Listen}$ die Länge der Liste. Diese Datentypen müssten dann Funktionen zur Verkürzung von Strings bzw. Listen implementieren.

6.2.1 Schleifen

Da unser Problem also die Schleifen sind, müssen wir die Hoare-Regel dafür etwas anpassen. Wir erweitern die bekannte Regel (H4) durch zwei weitere Prämissen, die die Variante t einführen:

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{\mathit{Inv} \wedge E \ \{\pi\} \ \mathit{Inv} \quad \mathit{Inv} \wedge E \wedge t = t_0 \ \{\pi\} \ t < t_0 \quad \mathit{Inv} \supset t \geq 0}{\mathit{Inv} \ \{\textbf{while } E \textbf{ do } \pi \textbf{ od}\} \ \mathit{Inv} \wedge \neg E}$$

t_0 ist dabei eine Hilfsvariable, die den Wert von t zu Beginn des Schleifendurchlaufs speichert, damit überprüft werden kann, dass t tatsächlich dekrementiert wurde.

Diese Regel kann noch vereinfacht werden: Zunächst fassen wir die erste und zweite Prämisse zusammen.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{\mathit{Inv} \wedge E \wedge t = t_0 \ \{\pi\} \ \mathit{Inv} \wedge t < t_0 \quad \mathit{Inv} \supset t \geq 0}{\mathit{Inv} \ \{\textbf{while } E \textbf{ do } \pi \textbf{ od}\} \ \mathit{Inv} \wedge \neg E}$$

Danach lockern wir die Einschränkungen auf die Invariante *Inv* etwas auf: Termination wird nicht verhindert, wenn $t \geq 0$ vor der Ausführung von π nicht gilt, solang es nachher gilt. Die Prämisse $\mathit{Inv} \supset t \geq 0$ verlangt das aber. Wir erhalten:

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{\mathit{Inv} \wedge E \wedge t = t_0 \ \{\pi\} \ \mathit{Inv} \wedge 0 \leq t < t_0}{\mathit{Inv} \ \{\textbf{while } E \textbf{ do } \pi \textbf{ od}\} \ \mathit{Inv} \wedge \neg E}$$

Auch hier können wir eine unnötige Einschränkung aufheben: $0 \leq t < t_0$ muss nicht gelten, wenn die Schleife wegen $\neg E$ terminieren würde.

$$\vdash_{\mathcal{H}(\mathcal{D})} \frac{Inv \wedge E \wedge t = t_0 \ \{\pi\} \ Inv \wedge (E \supset 0 \leq t < t_0)}{Inv \ \{\textbf{while } E \textbf{ do } \pi \textbf{ od}\} \ Inv \wedge \neg E} \ \text{(HR4")}$$

(HR4") ist die neue Regel für Schleifen, mit der totale Korrektheit erreicht wird. Betrachten wir zunächst die Prämisse. Aus der Tatsache, dass die Vorbedingung $t = t_0$ und die Nachbedingung $t < t_0$ enthält, müssen wir schließen, dass in π eine Anweisung wie $t \leftarrow t - 1$ vorkommt. (t um mehr als 1 zu verringern wäre natürlich auch möglich.) Da wir t_0 als neue Variable definiert haben, ist sichergestellt, dass t_0 in π nicht verändert wird.

Um nun die Termination der abgeleiteten Schleife zu zeigen, betrachten wir $E \supset 0 \leq t$. Mit jedem Schleifendurchlauf wird t verringert. Sofern nicht zuvor (unabhängig von t) $\neg E$ eintritt, erreicht t also früher oder später 0. Wird die Schleife jedoch mit $t = 0$ betreten und t weiter reduziert, wäre $E \supset 0 \leq t$ verletzt. Wir schließen also, dass E einen Konjunganden wie $t \geq 0$ enthalten muss. Somit wird die Schleife spätestens abgebrochen, wenn $t = 0$ erreicht ist, und das ist durch die streng monotone Dekrementierung von t sichergestellt. Die Schleife muss terminieren.

(Anmerkung: Der Schleifenkern π muss terminieren, weil eventuell darin enthaltene Schleifen ebenfalls mit (HR4") gebildet wurden.)

7 Die Vollständigkeit des Hoare-Kalküls

In Abschnitt 6 haben wir gezeigt, dass jede Ableitung mit den vorgestellten Regeln und Axiomen des Hoare-Kalküls auch gültig ist. Wir schreiben

$$\vdash_{\mathcal{H}(\mathcal{D})} P \ \{\pi\} \ Q \implies \models_{\mathcal{H}(\mathcal{D})} P \ \{\pi\} \ Q$$

Jetzt wollen wir die Umkehrung untersuchen. Wenn eine (partielle) Korrektheitsaussage gültig ist, existiert dann immer eine Ableitung, die das zeigt?

Die Auswertung von $\models_{\mathcal{H}(\mathcal{D})} P \ \{\pi\} \ Q$ ist vom gewählten Datentyp $\mathcal{D}$ abhängig. Da wir auf $\mathcal{D}$ keinen Einfluss haben und nichts über dessen Vollständigkeit wissen, können wir über die nötigen Auswertungen keine Aussagen treffen. Es wäre etwa vorstellbar, dass die Funktionen und Prädikate in $\mathcal{D}$ rekursiv gestaltet sind, sodass die Funktionen und Prädikate gar nicht terminieren müssen. Der Hoare-Kalkül $\mathcal{H}(\mathcal{D})$ kann also als ganzes nicht vollständig sein.

Wir können allerdings *relative Vollständigkeit* feststellen. Diese besagt, dass wir ein Orakel haben, das $\mathcal{D}$ für uns vollständig interpretiert. Wir müssen jedoch noch eine weitere Einschränkung treffen: Der Datentyp $\mathcal{D}$ muss *expressiv* sein. Für die Definition von Expressivität benötigen wir den

Begriff der *schwächsten möglichen Vorbedingung.*

Für $\pi \in Toy(\mathcal{D}), Q \in PL$ wird die *schwächste mögliche Vorbedingung* *wlp* (engl. *weakest liberal precondition*) wie definiert als die Menge jener Interpretationen, die vor der Ausführung von π gelten können, sodass die aus π resultierende Interpretation Q wahr macht.

$$wlp(\pi, Q) = \{I \in ENV \mid \mathcal{M}_{PL}(\mathcal{M}_{Toy}(I, \pi), Q)\}$$

Die Signatur von *wlp* ist in der Literatur eigentlich $Toy(\mathcal{D}) \times ENV \to ENV$. Wir verstehen hier den spezifikationssprachlichen Ausdruck Q als $\{I \in ENV \mid \mathcal{M}_{PL}(I, Q)\}$, die Menge der Interpretationen, die ihn wahr machen. Gleichermaßen kann das Ergebnis von $wlp(\pi, Q)$ als $\in PL(\mathcal{D})$ verstanden werden.

Durch die Anwendung von *wlp* hat man also eine nützliche Methode, um im Fall von Rückwärtsanalyse eine geeignete und, wie der Name sagt, schwächstmögliche Vorbedingung zu finden. Dies kann auch sehr einfach automatisiert werden. Für Zwischenbedingungen einer Konkatenation und Schleifeninvarianten eignet sich *wlp* jedoch nicht, und da es auch kaum einen anderen automatisierbaren Ersatz für das Auffinden derselben gibt, ist dies einer der Gründe, warum der Hoare-Kalkül nicht automatisiert eingesetzt wird.

Ein Datentyp $\mathcal{D}$ ist in Bezug auf eine Prädikatenlogik $PL(\mathcal{D})$ und eine Programmiersprache $Toy(\mathcal{D})$ *expressiv*, wenn *wlp* mit allen möglichen Argumenten stets ausgedrückt werden kann. Wir bezeichnen die Menge aller expressiven Datentypen mit *expr*.

$$expr(PL, Toy) = \{\mathcal{D} \mid \forall \pi \in Toy(\mathcal{D}), \forall P \in PL(\mathcal{D}), \forall Q \in PL(\mathcal{D}) : \\ P = wlp(\pi, Q)\}$$

Als Beispiel einer expressiven Datenstruktur sei die Peano-Arithmetik genannt, eine nicht-expressive Datenstruktur wäre hingegen die Presburger-Arithmetik. Andere Beispiele siehe auch [Wan78]. Wir gehen hier nicht näher darauf ein.

Für $\mathcal{D} \in expr(PL, Toy)$ gilt also die relative Vollständigkeit:

$$\models_{\mathcal{H}(\mathcal{D})} P\ \{\pi\}\ Q \implies \vdash_{\mathcal{H}(\mathcal{D})} P\ \{\pi\}\ Q$$

7.1 Automatisierbarkeit

Neben der oben genannten Schwierigkeit des automatisierten Auffindens von Zwischenbedingungen einer Konkatenation und Schleifeninvarianten gibt es

die allgemeintheoretische Einschränkung durch den *Gödelschen Unvollständigkeitssatz*. Damit wurde bewiesen, dass es in hinreichend starken widerspruchsfreien Systemen immer unbeweisbare gültige Aussagen gibt. Daher kann es auch keinen algorithmischen Test auf Wahrheit von Korrektheitsaussagen geben.

Für totale Korrektheitsaussagen müsste noch die Termination bewiesen werden. Hier greift die Nicht-Entscheidbarkeit des Halteproblems von Turingmaschinen: Es gibt keinen Algorithmus, der von einem beliebigen Programm entscheiden kann, ob es terminieren wird.

8 Rechenbeispiele

Im Folgenden zeigen wir die partielle Korrektheit in fünf Rechenbeispielen. Alle Beispiele basieren dabei auf dem Datentyp $\mathcal{Z}$ der ganzen Zahlen, folgendermaßen definiert:

$$\mathcal{Z} = \langle z, \{+, -, *\}, \{=, \neq, <, \leq, >, \geq\}, z\rangle$$

8.1 Vertauschen zweier Variablenwerte

Wir zeigen die Korrektheit des Programmes π, das die Werte von x und y ohne Benutzung einer Hilfsvariablen vertauschen soll.

$$\begin{aligned}\pi = \ & \textbf{begin} \\ & \quad \textbf{begin } x \leftarrow x - y \ ; \ y \leftarrow x + y \ \textbf{end} \ ; \\ & \quad x \leftarrow y - x \\ & \textbf{end}\end{aligned}$$

Dazu verwenden wir die beiden Spezifikationsvariablen x_0 und y_0 für die ursprünglichen Werte von x und y. Unsere Vorbedingung P lautet also $x = x_0 \wedge y = y_0$ und unsere Nachbedingung Q $y = x_0 \wedge x = y_0$.

Wir haben es hier mit drei Zuweisungsregeln zu tun, und wir verwenden dafür die Regel (HA3). Da diese Regel die Rückwärtsverkettung beschreibt, betrachten wir die Formel also von hinten nach vorne.

Als letzte Operation wird $x \leftarrow y - x$ durchgeführt, und nachher gilt Q. Vorher muss also $Q\left[\begin{smallmatrix} y-x \\ x \end{smallmatrix}\right]$ gegolten haben, das ist $(y = x_0) \wedge (y - x = y_0)$. Wir kürzen diese Formel mit S ab.

Vom inneren **begin**-Block ist bekannt, dass nachher S und vorher P gelten muss. Zwischen den beiden Anweisungen gilt die noch zu eruierende Bedingung R. Durch dieselbe Überlegung wie oben erhalten wir $S\left[\begin{smallmatrix} x+y \\ y \end{smallmatrix}\right] = R$ mit $((x + y) = x_0) \wedge ((x + y) - x = y_0)$, also $(x + y = x_0) \wedge (y = y_0)$.

Jetzt bleibt noch zu zeigen, dass aus $R\left[\begin{smallmatrix} x-y \\ x \end{smallmatrix}\right]$ auch P folgt: $((x-y)+y = x_0) \wedge (y = y_0)$, und das ist dasselbe wie P, nämlich $(x = x_0) \wedge (y = y_0)$.

Wir schreiben das zur Erhöhung der Übersichtlichkeit als Baum, gegeben in Ableitung (1) auf Seite 30.

Außerdem ist die folgende Veranschaulichung hilfreich:

$$\begin{aligned} P &\stackrel{\text{def}}{=} (x = x_0) \wedge (y = y_0) \\ &\quad \{x \leftarrow x - y\} \\ R &\stackrel{\text{def}}{=} (x + y = x_0) \wedge (y = y_0) \\ &\quad \{y \leftarrow x + y\} \\ S &\stackrel{\text{def}}{=} (y = x_0) \wedge (y - x = x_0) \\ &\quad \{x \leftarrow y - x\} \\ Q &\stackrel{\text{def}}{=} (y = x_0) \wedge (x = y_0) \end{aligned}$$

8.2 Multiplikation

Hier zeigen wir die Auswertung eines Programmes π, das mit Hilfe von Addition und Subtraktion Multiplikation simuliert. Die Funktion $*$ unseres Zahlentyps wird deshalb nur in der Spezifikationssprache, jedoch nicht in der Programmsprache verwendet (sonst wäre π nur eine Zeile lang). Der eine Faktor y wird solange aufsummiert (oder subtrahiert, falls er < 0 ist), wie der andere Faktors x anzeigt. Die so gewonnene Summe p ist das Produkt von x und y.

$$\begin{aligned} \pi = \ & \textbf{begin} \\ & \quad \textbf{begin } x \leftarrow x_0 \ ; \ p \leftarrow 0 \ \textbf{end} \ ; \\ & \quad \textbf{while } x > 0 \ \textbf{do} \\ & \quad\ \ \textbf{begin } p \leftarrow p + y_0 \ ; \ x \leftarrow x - 1 \ \textbf{end} \\ & \quad \textbf{od} \\ & \textbf{end} \end{aligned}$$

Die Vorbedingung P enthält nur $x_0 \geq 0$ um unseren Algorithmus einfach zu halten. Die Nachbedingung Q muss festhalten, dass $p = x_0 * y_0$ gilt. Über die Schleifeninvariante Inv können wir $x*y_0+p = x_0*y_0$ und $x \geq 0$ aussagen.

Die genaue Ableitung ist in (2) auf Seite 30 gegeben. Zunächst teilen wir π in den Initialisationsblock und die Schleife, zwischen denen Inv gelten muss. Da P und Inv sich nicht durch Zuweisungen allein ineinander überführen lassen, reduzieren wir dieses Problem auf $P \supset Inv \begin{bmatrix} 0 \\ p \end{bmatrix} \begin{bmatrix} x_0 \\ x \end{bmatrix}$:

$$\begin{aligned} & P \supset Inv \begin{bmatrix} 0 \\ p \end{bmatrix} \begin{bmatrix} x_0 \\ x \end{bmatrix} \\ x_0 \geq 0 \supset & ((x * y_0 + p = x_0 * y_0) \wedge x \geq 0) \begin{bmatrix} 0 \\ p \end{bmatrix} \begin{bmatrix} x_0 \\ x \end{bmatrix} \\ x_0 \geq 0 \supset & ((x_0 * y_0 + p = x_0 * y_0) \wedge x_0 \geq 0) \begin{bmatrix} 0 \\ p \end{bmatrix} \\ x_0 \geq 0 \supset & (x_0 * y_0 + 0 = x_0 * y_0) \wedge x_0 \geq 0 \\ & A \supset \textbf{true} \wedge A \end{aligned}$$

Für den Schleifenblock müssen wir zeigen, dass aus der Nachbedingung der Schleife laut (H4') Q folgt:

$$\begin{aligned} Inv \wedge \neg(x > 0) &\supset Q \\ (x * y_0 + p = x_0 * y_0) \wedge (x \geq 0) \wedge (x \leq 0) &\supset (x = 0) \wedge (p = x_0 * y_0) \\ (x * y_0 + p = x_0 * y_0) \wedge (x = 0) &\supset (x = 0) \wedge (p = x_0 * y_0) \\ (0 * y_0 + p = x_0 * y_0) &\supset (p = x_0 * y_0) \\ A &\supset A \end{aligned}$$

Um die Zuweisungen innerhalb der Schleife auszuwerten, müssen wir noch $Inv \wedge x > 0 \supset Inv \begin{bmatrix} x-1 \\ x \end{bmatrix} \begin{bmatrix} p+y_0 \\ p \end{bmatrix}$ zeigen.

$$\begin{aligned} Inv \wedge (x > 0) &\supset Inv \begin{bmatrix} x-1 \\ x \end{bmatrix} \begin{bmatrix} p+y_0 \\ p \end{bmatrix} \\ (x * y_0 + p = x_0 * y_0) \wedge (x \geq 0) \wedge (x > 0) &\supset (x-1) * y_0 + (p + y_0) = x_0 * y_0 \\ (x * y_0 + p = x_0 * y_0) \wedge (x \geq 0) &\supset x * y_0 - y_0 + p + y_0 = x_0 * y_0 \\ (x * y_0 + p = x_0 * y_0) \wedge (x \geq 0) &\supset x * y_0 + p = x_0 * y_0 \\ A \wedge B &\supset A \end{aligned}$$

8.3 Integer-Division mit Rest

Als nächstes betrachten wir das Programm π zur Berechnung der ganzzahligen Division von zwei Eingaben x (Dividend) und y (Divisor). Das Ergebnis wird in die beiden Variablen q (Quotient) und r (Rest) gespeichert.

$\pi =$ **begin**
  begin $q \leftarrow 0$; $r \leftarrow x$ **end** ;
  while $r \geq y$ **do**
    begin $r \leftarrow r - y$; $q \leftarrow q + 1$ **end**
  od
end

Als Vorbedingung P erwarten wir uns, dass x und y positive Zahlen sind und y nicht 0 sein darf: $x \geq 0 \wedge y > 0$. Die Nachbedingung Q muss aussagen, dass q und r die richtigen Werte haben, also $q*y+r = x$, und dass außerdem r minimal ist, also $r < y$.

Die Idee des Algorithmus ist die folgende: Wir beginnen mit $q = 0$ und $r = x$, diese Werte erfüllen die Invariante $q * y + r = x$ (denn $0 + r = x$). Solange r größer als y ist, subtrahieren wir y von r und erhöhen q um 1, dadurch bleibt die Invariante gültig. Sobald wir y nicht mehr von r abziehen können, ist die Minimalitätsbedingung in Q erfüllt und das Programm wird beendet. (Wenn $x < y$, wird die Schleife gar nicht durchlaufen. Dadurch erhalten wir z.B. $2 : 3 = 0$ und 2 Rest.)

Im ersten Teil des Programms (gegeben in Ableitung (3b) auf Seite 31) werden die beiden Initialwerte von q und r zugewiesen. Die Vorbedingungseinschränkung $y > 0$ dient nur zur Schleifentermination und kann auch weggelassen werden (immerhin beweisen wir nur partielle Korrektheit), sie wird mithilfe von (H5) eingeführt.

Wir wollen nach den beiden Zuweisungen Inv erreichen, und kommen mit (HA3) auf die Zwischenstufe $R \stackrel{\text{def}}{=} Inv\left[\begin{smallmatrix}x\\r\end{smallmatrix}\right]$. Für die erste Zuweisung müssen wir noch beweisen, dass (H1) anwendbar ist, das heißt, dass aus der Vorbedingung $x \geq 0$ auch $R\left[\begin{smallmatrix}0\\q\end{smallmatrix}\right]$ folgt.

$$\begin{aligned}
x \geq 0 &\supset R\left[\begin{smallmatrix}0\\q\end{smallmatrix}\right] \\
x \geq 0 &\supset Inv\left[\begin{smallmatrix}x\\r\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\q\end{smallmatrix}\right] \\
x \geq 0 &\supset ((q * y + r = x) \wedge (r \geq 0))\left[\begin{smallmatrix}x\\r\end{smallmatrix}\right]\left[\begin{smallmatrix}0\\q\end{smallmatrix}\right] \\
x \geq 0 &\supset (0 * y + x = x) \wedge (x \geq 0) \\
x \geq 0 &\supset (x = x) \wedge (x \geq 0) \\
A &\supset \mathbf{true} \wedge A
\end{aligned}$$

Im zweiten Teil (gegeben in Ableitung (3c)) wenden wir uns der Schleife zu. Um (H4') anwenden zu können, beweisen wir, dass $Inv \wedge \neg(r \geq y) \supset Q$.

$$\begin{aligned}
Inv \wedge \neg(r \geq y) &\supset Q \\
(q * y + r = x) \wedge (r \geq 0) \wedge \neg(r \geq y) &\supset (q * y + r = x) \wedge (r \geq 0) \wedge (r < y) \\
A \wedge B \wedge \neg(r \geq y) &\supset A \wedge B \wedge (r < y)
\end{aligned}$$

Anschließend beweisen wir die beiden Zuweisungen mit (HA3) und erhalten $T \stackrel{\text{def}}{=} Inv\left[\begin{smallmatrix}q+1\\q\end{smallmatrix}\right]$ und $S \stackrel{\text{def}}{=} T\left[\begin{smallmatrix}r-y\\r\end{smallmatrix}\right]$. Abschließend zeigen wir, dass die Vorbedingung der Schleife $Inv \wedge r \geq y$ ein Spezialfall von S ist.

$$\begin{aligned}
Inv \wedge (r \geq y) &\supset S \\
Inv \wedge (r \geq y) &\supset Inv\left[\begin{smallmatrix}q+1\\q\end{smallmatrix}\right]\left[\begin{smallmatrix}r-y\\r\end{smallmatrix}\right] \\
(q * y + r = x) \wedge (r \geq 0) \wedge (r \geq y) &\supset ((q+1) * y + (r-y) = x) \wedge (r - y \geq 0) \\
(q * y + r = x) \wedge (r \geq 0) \wedge (r \geq y) &\supset (q * y + y + r - y = x) \wedge (r - y \geq 0) \\
A \wedge (r \geq 0) \wedge (r \geq y) &\supset A \wedge (r - y \geq 0) \\
A \wedge (r \geq 0) \wedge (r \geq y) &\supset A \wedge (r \geq y) \\
A \wedge (r \geq 0) \wedge B &\supset A \wedge B
\end{aligned}$$

8.4 Größter gemeinsamer Teiler

Nun beweisen wir die Korrektheit eines Programms zur Berechnung des größten gemeinsamen Teilers von zwei Input-Werten x und y. Dafür definieren wir zunächst die Funktion gcd:

$$\begin{aligned} gcd(x,y) =& (\exists u)(\exists s_1)(\exists s_2)[(s_1 \neq s_2) \wedge (u * s_1 = x) \wedge (u * s_2 = y) \wedge \\ & (\forall v)(\exists t_1)(\exists t_2)[((t_1 \neq t_2) \wedge (v * t_1 = x) \wedge (v * t_2 = y)) \supset (v \leq u)]] \end{aligned}$$

Für die Berechnung von $gcd(x, y)$ verwenden wir die folgende Abänderung des Euklidischen Algorithmus:

$$gcd(x,y) = \begin{cases} x & \text{falls } x = y \\ gcd(x-y, y) & \text{falls } x > y \\ gcd(x, y-x) & \text{falls } x < y \end{cases}$$

Wir schreiben also das Programm π, dessen Korrektheit wir gleich überprüfen werden:

$$\begin{aligned} \pi = \; & \textbf{while } x \neq y \textbf{ do} \\ & \quad \textbf{if } x > y \textbf{ then } x \leftarrow x - y \textbf{ else } y \leftarrow y - x \textbf{ fi} \\ & \textbf{od} \end{aligned}$$

Wir wissen, dass wir den *gcd* gefunden haben, wenn x und y denselben Wert angenommen haben. Zu diesem Zeitpunkt brechen wir die Schleife ab und können das Ergebnis aus x oder y auslesen.

Da wir x und y verändern, merken wir uns die ursprünglichen Werte in x_0 und y_0. Eine geeignete Vorbedingung P wäre deshalb $(x = x_0) \wedge (y = y_0) \wedge (x \geq 1) \wedge (y \geq 1)$ und eine geeignete Nachbedingung Q wäre $(x = y) \wedge (x = gcd(x_0, y_0)) \wedge (x \geq 1)$.

Das Programm π ist relativ kurz und besteht nur aus der wiederholten Anwendung des Algorithmus. Wir beginnen daher mit der Definition einer Schleifeninvarianten *Inv*. Eulers Algorithmus besagt, dass sich der *gcd* nicht ändert, wenn seine Argumente nach dem oben genannten Schema verändert werden, daraus ergibt sich $(gcd(x, y) = gcd(x_0, y_0)) \wedge (x \geq 1) \wedge (y \geq 1)$.

Wir müssen noch zeigen, dass *Inv* aus P ...

$$\begin{gathered} P \supset Inv \\ (x = x_0) \wedge (y = y_0) \wedge (x \geq 1) \wedge (y \geq 1) \supset gcd(x,y) = gcd(x_0, y_0) \wedge (x \geq 1) \wedge (y \geq 1) \\ (x = x_0) \wedge (y = y_0) \supset gcd(x,y) = gcd(x_0, y_0) \end{gathered}$$

... und Q aus $Inv \wedge x = y$, der Nachbedingung der Schleife laut (H4), folgt.

$$\begin{gathered} Inv \wedge (x = y) \supset Q \\ gcd(x,y) = gcd(x_0, y_0) \wedge (x = y) \wedge (x \geq 1) \supset (x = y) \wedge x = gcd(x_0, y_0) \wedge (x \geq 1) \\ x = gcd(x_0, y_0) \wedge (x = y) \wedge (x \geq 1) \supset (x = y) \wedge x = gcd(x_0, y_0) \wedge (x \geq 1) \\ A \wedge B \wedge C \supset A \wedge B \wedge C \end{gathered}$$

Jetzt können wir mit Hilfe der Regel (H4') eine Prämisse finden. Da wir dazu die Schleifenbedingung $x \neq y$ in die Vorbedingung aufnehmen müssen, ist für den nächsten Schritt, die Auswertung des **if**-Blocks, sichergestellt, dass in keinem der beiden Zweige $x = y$ gilt.

Wir müssen die beiden Zweige des **if**-Blocks getrennt betrachten, in beiden gilt die Nachbedingung der Schleife, *Inv*, als Nachbedingung. Die Vorbedingung lautet $Inv \wedge x > y$ bzw. $Inv \wedge x < y$. Für die Auswertung der beiden Zuweisungen benutzen wir wieder die Regel (HA3), dadurch erhalten wir $Inv\begin{bmatrix} x-y \\ x \end{bmatrix}$ bzw. $Inv\begin{bmatrix} y-x \\ y \end{bmatrix}$. Es gilt also noch zu zeigen, dass $Inv \wedge x > y \supset Inv\begin{bmatrix} x-y \\ x \end{bmatrix}$ und $Inv \wedge x < y \supset Inv\begin{bmatrix} y \\ y-x \end{bmatrix}$.

$$\begin{aligned} Inv \wedge (x > y) &\supset Inv\begin{bmatrix} x-y \\ x \end{bmatrix} \\ gcd(x,y) = gcd(x_0,y_0) \wedge (x \geq 1) \wedge (y \geq 1) \wedge (x > y) &\supset Inv\begin{bmatrix} x-y \\ x \end{bmatrix} \\ gcd(x-y,y) = gcd(x_0,y_0) \wedge (x-y \geq 1) \wedge (y \geq 1) &\supset Inv\begin{bmatrix} x-y \\ x \end{bmatrix} \\ Inv\begin{bmatrix} x-y \\ x \end{bmatrix} &\supset Inv\begin{bmatrix} x-y \\ x \end{bmatrix} \end{aligned}$$

$$\begin{aligned} Inv \wedge (x < y) &\supset Inv\begin{bmatrix} y-x \\ y \end{bmatrix} \\ gcd(x,y) = gcd(x_0,y_0) \wedge (x \geq 1) \wedge (y \geq 1) \wedge (x < y) &\supset Inv\begin{bmatrix} y-x \\ y \end{bmatrix} \\ gcd(x,y-x) = gcd(x_0,y_0) \wedge (x \geq 1) \wedge (y-x \geq 1) &\supset Inv\begin{bmatrix} y-x \\ y \end{bmatrix} \\ Inv\begin{bmatrix} y-x \\ y \end{bmatrix} &\supset Inv\begin{bmatrix} y-x \\ y \end{bmatrix} \end{aligned}$$

8.5 Potenz

Im letzten Beispiel zeigen wir, wie das Überlegen von geeigneten Schleifeninvarianten den Programmierstil beeinflussen kann. Wir betrachten ein Programm zur Berechnung der Potenz von zwei Eingaben. Dazu müssen wir unseren Datentyp $\mathcal{Z}$ so verändern, dass er die Spezifikationsfunktion $\uparrow$ zur Potenzierung, die Programmfunktion *half*$(\cdot)$ zur ganzzahligen Division durch 2 (abgerundet) und das Programmprädikat *even*$(\cdot)$, das anzeigt, ob sein Argument eine gerade Zahl ist, unterstützt.

Auch in diesem Beispiel sehen wir wieder (vergleiche Abschnitt 8.2), wie wichtig der Unterschied zwischen Spezifikationssprache und Programm(ier)-sprache ist. Wenn uns die Funktion zur Potenzierung in der Programmiersprache zur Verfügung stünde, wäre das Problem sehr viel schneller gelöst.

Wir erwarten uns von unserem Programm, dass die Eingabewerte x_0 und $y_0 \geq 0$ sind (wieder zur Vereinfachung des Algorithmus). Nach der Ausführung soll die berechnete Variable p das Ergebnis $x_0 \uparrow y_0$, anders notiert ${x_0}^{y_0}$, enthalten.

Wir schreiben also das Programm π, das dem in Abschnitt 8.2 stark

ähnelt.

$\pi =$ **begin**
begin $y \leftarrow y_0$; $p \leftarrow 1$ **end** ;
while $y > 0$ **do**
begin $p \leftarrow p * x_0$; $y \leftarrow y - 1$ **end**
od
end

Der Korrektheitsbeweis von π geht analog zu dem in Abschnitt 8.2.

Die Schleifeninvariante $Inv \stackrel{\text{def}}{=} (p * {x_0}^y = {x_0}^{y_0})$ beruht darauf, dass $p * {x_0}^y = p * x_0 * {x_0}^{y-1}$. Wenn wir eine alternative Invariante wählen, können wir den Algorithmus anders gestalten und seine Laufzeit verbessern. $p * {x_0}^y = p * (x_0 * x_0)^{half(y)}$ ergibt den Algorithmus π'.

$\pi' =$ **begin**
$\alpha =$ **begin**
begin $x \leftarrow x_0$; $y \leftarrow y_0$ **end** ;
$p \leftarrow 1$
end ;
while $y > 0$ **do**
begin
$\beta =$ **while** $even(y)$ **do**
begin $x \leftarrow x * x$; $y \leftarrow half(y)$ **end**
od ;
$\gamma =$ **begin** $p \leftarrow p * x$; $y \leftarrow y - 1$ **end**
end
od
end

Wenn/solange y eine gerade Zahl ist, kann der Invarianten-Trick angewendet werden. Ist y ungerade (geworden), verhält sich π' wie π, bis y wieder gerade ist oder 0 erreicht hat.

Wir betrachten π' am Beispiel $x_0 = 3$, $y_0 = 9$ und erwarten das Ergebnis 19683.

Insgesamt benötigen wir zwei Durchläufe der äußeren Schleife γ und durchlaufen beim zweiten Durchgang die innere Schleife β dreimal. Der Algorithmus π' ist also schneller als π. π benötigt immer y_0 Schleifendurchläufe, π' hingegen im Best Case (wenn y_0 eine 2er-Potenz ist) nur einen Durchlauf der äußeren Schleife und $\log_2(y_0)$ der inneren.

	x	y	p
	3	9	1
γ		$9-1=8$	$1*3=3$
β	$3*3=9$	$half(8)=4$	
β	$9*9=81$	$half(4)=2$	
β	$81*81=6561$	$half(2)=1$	
γ		$1-1=0$	$3*6561=19683$

Dieses Beispiel soll demonstrieren, welchen Einfluss Invariantenüberlegungen auf den Programmierstil haben können.

In Ableitung (5) auf Seite 33 ist die genaue Ableitung von π' gegeben. Als Vorbedingung P wird $x_0 \geq 0 \wedge y_0 \geq 0$ angenommen, als Nachbedingung Q natürlich $p = {x_0}^{y_0}$. Zunächst wird α ausgewertet und auf die äußere Schleifeninvariante $Inv \stackrel{\text{def}}{=} p * x^y = {x_0}^{y_0} \wedge y \geq 0$ hingeführt. Für die äußere Schleife zeigen wir, dass $Inv \wedge \neg(y > 0) \supset Q$:

$$\begin{aligned} Inv \wedge \neg(y > 0) &\supset Q \\ (p * x^y = {x_0}^{y_0}) \wedge (y \geq 0) \wedge (y \leq 0) &\supset (p = {x_0}^{y_0}) \\ (p * x^y = {x_0}^{y_0}) \wedge (y = 0) &\supset (p = {x_0}^{y_0}) \end{aligned}$$

da $u^0 = 1$ gilt $\forall u$:

$$\begin{aligned} (p * 1 = {x_0}^{y_0}) \wedge (y = 0) &\supset (p = {x_0}^{y_0}) \\ A \wedge B &\supset A \end{aligned}$$

Mit $Inv \wedge (y > 0)$ als Vor- und Inv als Nachbedingung werten wir **begin** β ; γ **end** aus. Mit (HA3) auf γ angewendet erreichen wir die Zwischenstufe $R \stackrel{\text{def}}{=} Inv\left[\begin{smallmatrix} y-1 \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} x*p \\ p \end{smallmatrix}\right]$, also $(p * x^y = {x_0}^{y_0}) \wedge (y \geq 1)$. R ist damit die Nachbedingung der inneren Schleife β. Wir benennen $Inv \wedge y > 0$ mit Inv' und zeigen unten mit $Inv' \wedge \neg even(y) \supset R$, dass Inv' die innere Schleifeninvariante ist.

$$\begin{aligned} Inv' \wedge \neg even(y) &\supset R \\ Inv \wedge (y > 0) \wedge \neg even(y) &\supset Inv\left[\begin{smallmatrix} y-1 \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} p*x \\ p \end{smallmatrix}\right] \\ (p * x^y = {x_0}^{y_0}) \wedge (y > 0) \wedge \neg even(y) &\supset ((p * x) * x^{y-1} = {x_0}^{y_0}) \wedge (y - 1 \geq 0) \\ (p * x^y = {x_0}^{y_0}) \wedge (y > 0) \wedge \neg even(y) &\supset (p * x^y = {x_0}^{y_0}) \wedge (y \geq 1) \end{aligned}$$

Da y eine ganze Zahl ist, sind $y > 0$ und $y \geq 1$ gleichwertig. Ob $even(y)$ gilt, ist nicht relevant.

Für die Auswertung von β mit (HA3) müssen wir noch $Inv' \wedge even(y) \supset$

$Inv'\begin{bmatrix} half(y) \\ y \end{bmatrix}^{[x*x]}_{[x]}$ zeigen.

$$Inv' \wedge even(y) \supset Inv'\begin{bmatrix} half(y) \\ y \end{bmatrix}^{[x*x]}_{[x]}$$

$$(p * x^y = {x_0}^{y_0}) \wedge (y > 0) \wedge even(y) \supset (p * (x * x)^{half(y)} = {x_0}^{y_0})$$

Da $half(y)$ die ganzzahlige Division durch 2 darstellt (z. B. $half(7) = 3$), muss $even(y)$ gelten, damit sichergestellt ist, dass $(x * x)^{half(y)} = x^y$. Laut linkem Teil der Implikation gilt $even(y)$, und damit ist die Implikation korrekt.

8.6 Ableitungen

8.6.1 Ableitung zu 8.1

$$\dfrac{\dfrac{\overbrace{R\left[\begin{smallmatrix}x-y\\x\end{smallmatrix}\right]}^{P}\ \{x \leftarrow x-y\}\ \overbrace{(x+y=x_0)\wedge(y=y_0)}^{R=S\left[\begin{smallmatrix}x+y\\y\end{smallmatrix}\right]}\quad R\ \{y \leftarrow x+y\}\ \overbrace{(y=x_0)\wedge(y-x=y_0)}^{S=Q\left[\begin{smallmatrix}y-x\\x\end{smallmatrix}\right]}}{P\ \{\textbf{begin}\ x \leftarrow x-y\ ;\ y \leftarrow x+y\ \textbf{end}\}\ S}\ \text{(H2)} \qquad S\ \{x \leftarrow y-x\}\ Q}{P\ \{\textbf{begin begin}\ x \leftarrow x-y\ ;\ y \leftarrow x+y\ \textbf{end}\ ;\ x \leftarrow y-x\ \textbf{end}\}\ Q}\ \text{(H2)} \tag{1}$$

8.6.2 Ableitung zu 8.2

$$\dfrac{\text{siehe (2b)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{Inv\wedge(x>0)\supset Inv\left[\begin{smallmatrix}x-1\\x\end{smallmatrix}\right]\left[\begin{smallmatrix}p+y_0\\p\end{smallmatrix}\right]}{Inv\wedge(x>0)\ \{p \leftarrow p+y_0\}\ Inv\left[\begin{smallmatrix}x-1\\x\end{smallmatrix}\right]}\ \text{(H1)} \qquad \overset{\text{(HA3)}}{Inv\left[\begin{smallmatrix}x-1\\x\end{smallmatrix}\right]\ \{x \leftarrow x-1\}\ Inv}}{Inv\wedge(x>0)\ \{\textbf{begin}\ p \leftarrow p+y_0\ ;\ x \leftarrow x-1\ \textbf{end}\}\ Inv}\ \text{(H2)}}{Inv\ \{\textbf{begin}\ p \leftarrow p+y_0\ ;\ x \leftarrow x-1\ \textbf{end}\}\ Inv\wedge\neg(x>0)}\ \text{(H4')} \qquad Inv\wedge\neg(x>0)\supset Q}{Inv\ \{\textbf{while}\ x>0\ \textbf{do begin}\ p \leftarrow p+y_0\ ;\ x \leftarrow x-1\ \textbf{end od}\}\ Q}\ \text{(H5)}}{P\ \{\textbf{begin begin}\ x \leftarrow x_0\ ;\ p \leftarrow 0\ \textbf{end}\ ;\ \textbf{while}\ x>0\ \textbf{do begin}\ p \leftarrow p+y_0\ ;\ x \leftarrow x-1\ \textbf{end od end}\}\ Q}\ \text{(H2)} \tag{2a}$$

$$\dfrac{\dfrac{P\supset Inv\left[\begin{smallmatrix}0\\p\end{smallmatrix}\right]\left[\begin{smallmatrix}x_0\\x\end{smallmatrix}\right]}{P\ \{x \leftarrow x_0\}\ Inv\left[\begin{smallmatrix}0\\p\end{smallmatrix}\right]}\ \text{(H1)} \qquad \overset{\text{(HA3)}}{Inv\left[\begin{smallmatrix}0\\p\end{smallmatrix}\right]\ \{p \leftarrow 0\}\ Inv}}{P\ \{\textbf{begin}\ x \leftarrow x_0\ ;\ p \leftarrow 0\ \textbf{end}\}\ Inv}\ \text{(H2)} \tag{2b}$$

8.6.3 Ableitung zu 8.3

$$\dfrac{\dfrac{\text{siehe Ableitung (3b)}}{P\ \{\textbf{begin}\ q \leftarrow 0\ ;\ r \leftarrow x\ \textbf{end}\}\ Inv} \quad \dfrac{\text{siehe Ableitung (3c)}}{Inv\ \{\textbf{while}\ r \geq y\ \textbf{do begin}\ r \leftarrow r - y\ ;\ q \leftarrow q + 1\ \textbf{end od}\}\ Q}\ \text{(H2)}}{P\ \{\textbf{begin begin}\ q \leftarrow 0\ ;\ r \leftarrow x\ \textbf{end}\ ;\ \textbf{while}\ r \geq y\ \textbf{do begin}\ r \leftarrow r - y\ ;\ q \leftarrow q + 1\ \textbf{end od end}\}\ Q} \tag{3a}$$

$$\dfrac{P \supset x \geq 0 \quad \dfrac{\dfrac{(x \geq 0) \supset R\left[\begin{smallmatrix}0\\q\end{smallmatrix}\right]}{x \geq 0\ \{q \leftarrow 0\}\ R}\ \text{(H1)} \quad \dfrac{R\ \{r \leftarrow x\}\ Inv}{x \geq 0\ \{\textbf{begin}\ q \leftarrow 0\ ;\ r \leftarrow x\ \textbf{end}\}\ Inv}\ \text{(H2)}}{\underbrace{(x \geq 0) \wedge (y > 0)}_{P}\ \{\textbf{begin}\ q \leftarrow 0\ ;\ r \leftarrow x\ \textbf{end}\}\ \underbrace{(q * y + r = x) \wedge (r \geq 0)}_{Inv}}\ \text{(H5)} \tag{3b}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{S\ \{r \leftarrow r - y\}\ T}\ \text{(HA3)} \quad \dfrac{}{T\ \{q \leftarrow q + 1\}\ Inv}\ \text{(HA3)}}{S\ \{\textbf{begin}\ r \leftarrow r - y\ ;\ q \leftarrow q + 1\ \textbf{end}\}\ Inv}\ \text{(H2)} \quad Inv \wedge r \geq y \supset S}{Inv \wedge r \geq y\ \{\textbf{begin}\ r \leftarrow r - y\ ;\ q \leftarrow q + 1\ \textbf{end}\}\ Inv}\ \text{(H5)}}{Inv\ \{\textbf{while}\ r \geq y\ \textbf{do begin}\ r \leftarrow r - y\ ;\ q \leftarrow q + 1\ \textbf{end od}\}\ Inv \wedge \neg(r \geq y)}\ \text{(H4')} \quad Inv \wedge \neg(r \geq y) \supset Q}{Inv\ \{\textbf{while}\ r \geq y\ \textbf{do begin}\ r \leftarrow r - y\ ;\ q \leftarrow q + 1\ \textbf{end od}\}\ Q}\ \text{(H5)} \tag{3c}$$

8.6.4 Ableitung zu 8.4

$$\dfrac{P \supset Inv \quad \dfrac{\text{siehe Ableitung (4b)} \quad \text{siehe Ableitung (4c)}}{Inv \wedge x \neq y \ \{\textbf{if } x > y \textbf{ then } x \leftarrow x - y \textbf{ else } y \leftarrow y - x \textbf{ fi}\} \ Inv}\ (\text{H3}) \quad Inv \wedge (x = y) \supset Q}{P \ \{\textbf{while } x \neq y \textbf{ do if } x > y \textbf{ then } x \leftarrow x - y \textbf{ else } y \leftarrow y - x \textbf{ fi od}\} \ Q}\ (\text{H4}) \tag{4a}$$

$$\dfrac{Inv \wedge (x > y) \supset Inv\begin{bmatrix} x-y \\ x \end{bmatrix} \quad \overset{(\text{HA3})}{Inv\begin{bmatrix} x-y \\ x \end{bmatrix}} \ \{x \leftarrow x - y\} \ Inv}{Inv \wedge (x > y) \ \{x \leftarrow x - y\} \ Inv}\ (\text{H5}) \tag{4b}$$

$$\dfrac{Inv \wedge (x < y) \supset Inv\begin{bmatrix} y-x \\ y \end{bmatrix} \quad \overset{(\text{HA3})}{Inv\begin{bmatrix} y-x \\ y \end{bmatrix}} \ \{y \leftarrow y - x\} \ Inv}{Inv \wedge (x < y) \ \{y \leftarrow y - x\} \ Inv}\ (\text{H5}) \tag{4c}$$

8.6.5 Ableitung zu 8.5

$$\dfrac{\dfrac{\text{siehe Ableitung (5b)}}{P\ \{\alpha\}\ Inv} \quad \dfrac{\text{siehe Ableitung (5e)}}{Inv\ \{\textbf{while}\ y > 0\ \textbf{do begin}\ \beta\ ;\ \gamma\ \textbf{end od}\}\ Q}}{P\ \{\textbf{begin}\ \alpha\ ;\ \textbf{while}\ y > 0\ \textbf{do begin}\ \beta\ ;\ \gamma\ \textbf{end od end}\}\ Q}\ \text{(H2)} \tag{5a}$$

$$\dfrac{P \supset P \wedge (x_0{}^{y_0} = x_0{}^{y_0}) \quad \dfrac{\text{siehe (5c)}}{P \wedge (x_0{}^{y_0} = x_0{}^{y_0})\ \{\alpha\}\ P \wedge Inv} \quad P \wedge Inv \supset Inv}{P\ \{\alpha\}\ Inv}\ \text{(H5)} \tag{5b}$$

$$\dfrac{\dfrac{P \wedge (x_0{}^{y_0} = x_0{}^{y_0})\ \{x \leftarrow x_0\}\ P \wedge (x^{y_0} = x_0{}^{y_0}) \quad (\text{HA3}) \qquad P \wedge (x^{y_0} = x_0{}^{y_0})\ \{y \leftarrow y_0\}\ P \wedge (x^y = x_0{}^{y_0}) \quad (\text{HA3})}{P \wedge (x_0{}^{y_0} = x_0{}^{y_0})\ \{\textbf{begin}\ x \leftarrow x_0\ ;\ y \leftarrow y_0\ \textbf{end}\}\ x^y = x_0{}^{y_0}}\ (\text{H2}) \qquad \text{siehe (5d)}}{P \wedge (x_0{}^{y_0} = x_0{}^{y_0})\ \{\textbf{begin begin}\ x \leftarrow x_0\ ;\ y \leftarrow y_0\ \textbf{end}\ ;\ p \leftarrow 1\ \textbf{end}\}\ P \wedge Inv}\ (\text{H2}) \tag{5c}$$

$$P \wedge (1 * x^y = x_0{}^{y_0})\ \{p \leftarrow 1\}\ P \wedge (p * x^y = x_0{}^{y_0}) \quad (\text{HA3}) \tag{5d}$$

$$\dfrac{\dfrac{\dfrac{\text{siehe (5f)}}{Inv \wedge (y > 0)\ \{\beta\}\ R} \qquad \dfrac{\overbrace{Inv\left[\begin{smallmatrix} y-1 \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} p*x \\ p \end{smallmatrix}\right]}^{R}\ \{p \leftarrow p * x\}\ Inv\left[\begin{smallmatrix} y-1 \\ y \end{smallmatrix}\right] \quad (\text{HA3}) \qquad Inv\left[\begin{smallmatrix} y-1 \\ y \end{smallmatrix}\right]\ \{y \leftarrow y - 1\}\ Inv \quad (\text{HA3})}{R\ \{\textbf{begin}\ p \leftarrow p * x\ ;\ y \leftarrow y - 1\ \textbf{end}\}\ Inv}\ (\text{H2})}{Inv \wedge (y > 0)\ \{\textbf{begin}\ \beta\ ;\ \gamma\ \textbf{end}\}\ Inv}\ (\text{H2})}{Inv\ \{\textbf{while}\ y > 0\ \textbf{do begin}\ \beta\ ;\ \gamma\ \textbf{end od}\}\ Inv \wedge \neg(y > 0)}\ (\text{H4'}) \qquad Inv \wedge \neg(y > 0) \supset Q}{Inv\ \{\textbf{while}\ y > 0\ \textbf{do begin}\ \beta\ ;\ \gamma\ \textbf{end od}\}\ Q}\ (\text{H5}) \tag{5e}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{Inv' \wedge even(y) \supset Inv'\left[\begin{smallmatrix} half(y) \\ y \end{smallmatrix}\right]\left[\begin{smallmatrix} x*x \\ x \end{smallmatrix}\right]}{Inv' \wedge even(y)\ \{x \leftarrow x * x\}\ Inv'\left[\begin{smallmatrix} half(y) \\ y \end{smallmatrix}\right]}\ (\text{H1}) \qquad Inv'\left[\begin{smallmatrix} half(y) \\ y \end{smallmatrix}\right]\ \{y \leftarrow half(y)\}\ Inv' \quad (\text{HA3})}{Inv' \wedge even(y)\ \{\textbf{begin}\ x \leftarrow x * x\ ;\ y \leftarrow half(y)\ \textbf{end}\}\ Inv'}\ (\text{H2})}{Inv'\ \{\textbf{while}\ even(y)\ \textbf{do begin}\ x \leftarrow x * x\ ;\ y \leftarrow half(y)\ \textbf{end od}\}\ Inv' \wedge \neg even(y)}\ (\text{H4'}) \qquad Inv' \wedge \neg even(y) \supset R}{\underbrace{Inv \wedge (y > 0)}_{Inv'}\ \{\textbf{while}\ even(y)\ \textbf{do begin}\ x \leftarrow x * x\ ;\ y \leftarrow half(y)\ \textbf{end od}\}\ R}\ (\text{H5}) \tag{5f}$$

Literatur

[Cla79] E. M. Clark. Programming language constructs for which it is impossible to obtain good Hoare-like axioms. *Journal of the ACM*, 26:129–147, 1979.

[Coo78] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7:70–90, 1978.

[Flo67] R. W. Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19:19–32, 1967.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, October 1969.

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science. Modelling and Reasoning about Systems*. Cambridge University Press, second edition, 2004.

[HS92] Bernhard Hohlfeld and Werner Struckmann. *Einführung in die Programmverifikation. Theorie und Anwendung der Hoareschen Methode am Beispiel der Programmiersprache PASCAL*, volume 88 of *Reihe Informatik*. BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1992.

[Jos85] Bernhard R. M. M. Josko. *Über einen Hoare-Kalkül für ALGOL-ähnliche Programmiersprachen mit Prozeduren endlicher Art*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 1985.

[Old81] Ernst-Rüdiger Olderog. Sound and complete hoare-like calculi based on copy rules. *Acta Informatica*, 16:161–197, 1981.

[Wan78] Mitchell Wand. A new incompleteness result for hoare's system. *Journal of the ACM*, pages 168–175, 1978.

[Wan80] Mitchell Wand. *Induction, Recursion, and Programming*. Elsevier Science Publishing Co., 1980.