Semantik & Verifikation von Programmen

050026 VO Theoretische Informatik

Wolfgang Dvořák

Theory and Applications of Algorithms Gruppe,
Fakultät für Informatik, Universität Wien

Sommersemester 2016

Einleitung

Eine **Programmiersprache** ist ein Notationssystem zur Beschreibung von Berechnungen.

- Muss sowohl für den Programmierer als auch
- für die Implementierung auf Computern geeignet sein.

Mensch und Maschine haben sehr unterschiedliche Anforderungen:

- **Höhere Programmiersprachen** (high-level): begünstigen den Menschen/Programmierer
- **Maschinenorientierte Programmiersprachen** (low-level): orientieren sich mehr an den Erfordernissen der Maschine

Anforderungen an Programmiersprachen

Anforderungen für **Lesbarkeit durch Menschen**:

- nahe an gewohnter Umgangssprache (z.B. Englisch)
- übersichtliche Struktur
- kompakte Notation
- ...

Anforderungen für **Ausführbarkeit durch Maschinen**:

- präzise und eindeutig
- leicht in Strukturen der Maschine umsetzbar
- ...

Anforderungen an Programmiersprachen

Ergebnis: Programmiersprachen mit folgenden Eigenschaften

- Umgangssprachliche Komponenten (Wörter wie “if“, “while“ . . .),
- Notation teilweise angelehnt an mathematische Schreibweise,
- Gliederung in Zeilen mit Einrückung,
- formale Syntax, d.h. kein Auslegungsspielraum
- formale Semantik
- maschinennahe Grundbausteine (ganze Zahlen, usw.) und Datenstrukturen

Syntax, Semantik und Pragmatik

Verschiedene Ebenen für Eigenschaften von Programmiersprachen.

Syntax:

- Die Syntax regelt die "Rechtschreibung" einer Programmiersprache. Sie lässt sich formal festlegen, z.B. mit Syntax-Diagrammen oder in einer Grammatik.

Semantik:

- Die Semantik regelt die Bedeutung einzelner Sprachelemente und ihr Zusammenspiel. Sie lässt sich schwieriger exakt festhalten als die Syntax.

Pragmatik:

- Die Pragmatik beschäftigt sich mit der "usability" der Sprache.

Oft sind die Grenzen zwischen Syntax und Semantik, sowie zwischen Semantik und Pragmatik verschwommen.

Formale Semantik

Warum Formale Semantik?

- Zur Konstruktion geeigneter Compiler für höhere Programmiersprachen.
- Programmierer benötigt genaue Kenntnis der Semantik
  $\hookrightarrow$ Sicherheit beim Entwurf von Programmen
- Rechnerunterstützte Programmentwicklung
- Formulierung und Beweisen von Programmeigenschaften
  $\hookrightarrow$ Verifikation von Programmen

In dieser Vorlesung liegt der Fokus auf Verifikation.

Formale Semantik - Verifikation

Wozu Programme verifizieren?[1]

In einem Teil eines großen Programms wird eine ganzzahlige Division x/y durchgeführt, d.h. der Quotient q und der Rest r berechnet.

$r := x; q := 0;$
while $r > y$ **do**
    $r := r - y;$
    $q := q + 1;$
end while

Wir wollen wissen ob das Programm korrekt arbeitet.

[1] Frei nach *David Gries - The Science of Programming.* Kapitel: *Why use Logic? Why Prove Programs Correct?*

Verifikation - Ein einführendes Beispiel

Wir führen eine Ausgabe ein um die Berechnung zu überprüfen:

write (“divident x = “ x, “divisor y = “ y);
$r := x; q := 0;$
while $r > y$ **do**
    $r := r - y;$
    $q := q + 1;$
end while
write ($“y \cdot q + r =$ “$y \cdot q + r$)

Ist das Programmstück in einer Schleife bekommen wir viele Ausgaben.

Wir interessieren uns nur für die Fälle bei denen ein Fehler auftritt.

$\hookrightarrow$ Assertions / Zusicherungen

Verifikation - Ein einführendes Beispiel

Assertions

- sind Aussagen/Zusicherungen über den Zustand eines Computer-Programms.
- geben eine Fehlermeldung aus wenn sie während der Ausführung des Programms verletzt werden.
- können oft über Kompileroptionen gesteuert/deaktiviert werden.

$\{y > 0\}$
$r := x; q := 0;$
while $r > y$ **do**
    $r := r - y;$
    $q := q + 1;$
end while
$\{x = y \cdot q + r \wedge 0 \leq r < y\}$

Nach einigen Testläufen bekommen wir einen Fehler für

$$x = 6, y = 3, q = 1, r = 3$$

Verifikation - Ein einführendes Beispiel

Wir ersetzen $r > y$ durch $r \geq y$:

$\{y > 0\}$
$r := x; q := 0;$
while $r \geq y$ **do**
    $r := r - y;$
    $q := q + 1;$
end while
$\{x = y \cdot q + r \wedge 0 \leq r < y\}$

Nach einigen Testläufen bekommen wir einen weiteren Fehler für

$$r = -2, x = -2.$$

Das Problem ist, dass das Programm nur für nicht-negative x gedacht ist und x in der Eingabe negativ ist.

Verifikation - Ein einführendes Beispiel

Wir machen die assertions so stark wie möglich:

$\{x \geq 0 \wedge y > 0\}$
$r := x; q := 0;$
$\{0 \leq r \wedge 0 < y \wedge x = y * q + r\}$
while $r \geq y$ **do**
  $\{0 \leq r \wedge 0 < y \leq r \wedge x = y * q + r\}$
  $r := r - y;\ q := q + 1;$
  $\{0 \leq r \wedge 0 < y \wedge x = y * q + r\}$
end while
$\{x = y \cdot q + r \wedge 0 \leq r < y\}$

Wann immer $x \geq 0 \wedge y > 0$ zu Beginn erfüllt ist dann ist $x = y \cdot q + r \wedge 0 \leq r < y$ nach der Ausführung erfüllt.

Kann man durch systematisches Vorgehen solche Fehler vermeiden?

Verifikation - Ein einführendes Beispiel

Wir hatten zwei Arten von Fehlern

- Lückenhafte Spezifikation des Programms
- Fehler im Programm

Um Erstes zu vermeiden hilft nur eine

- Gründliche Spezifikation was das Programm tun soll: Die erste Assertion (precondition) und die letzte Assertion (postcondition) sollten vor dem Programm geschrieben werden.

Der Fehler im Programm

- wäre dann mit formalen Methoden erkennbar gewesen, d.h. man kann zeigen dass ein Programm der Spezifikation entspricht.

Verifikation - Ein einführendes Beispiel

Formale Verifikation kann

- Zeit beim Debugging sparen und
- in kritischen Systemen eine höhere Sicherheit gewähren. Es werden auch (seltene) Fehler gefunden die nicht durch Testcases abgedeckt sind.

Formale Verifikation benötigt aber

- eine formale Spezifikation des Programms und
- eine formale Spezifikation der Semantik der Programmiersprache.

Formale Semantik

Es gibt verschiedene Methoden zur formalen Spezifikation der Semantik von Programmiersprachen.

In der Vorlesung:

- **Axiomatische Semantik**: Wir betrachtet logische Aussagen über Zustände des Programms. Die Semantik von Programmen wird durch Veränderungen dieser logischen Aussagen spezifiziert.

andere Methoden: *Operationale Semantik*, *Denotationelle Semantik*, ...

Die Sprache While

Wir betrachten die simple Programmiersprache **While** bestehend aus

- arithmetischen Operationen $+, -, *, /, mod$
- Vergleichsoperatoren $=, <, >, \leq, \geq, \neq$
- *not* Operator
- Zuweisungen z.B. $x := 2$
- if Anweisungen
- While Schleifen
- skip, abort Anweisungen

Axiomatische Semantik

Idee: Wir definieren wie sich die Ausführung eines Programms auf Zusicherungen auswirkt. Zusicherungen formulieren wir dabei als (Prädikatenlogische) Formeln.

Definition

Seien Q und R Prädikate, und S ein Programm. Dann nennen wir

$$\{Q\}\ S\ \{R\}$$

ein Hoare Tripel (nach Tony Hoare). Q wird als Vorbedingung (engl. precondition) und R als die Nachbedingung (engl. postcondition) bezeichnet.

Die Bedeutung eines Hoare Tripels ist, dass wenn Q vor der Ausführung des Programms S erfüllt ist dann ist R nach der Ausführung erfüllt.

Beispiel

$$\{x > 3\}\ x := x + 2\ \{x > 5\}$$

Partielle vs. Totale Korrektheit

Partielle Korrektheit: Idee wenn das Programm ein Ergebnis liefert ist dieses korrekt. $\{Q\}\ S\ \{R\}$ liest sich wie folgt: Wenn die Vorbedingung Q erfüllt ist und S terminiert dann ist die Nachbedingung R erfüllt.

Totale Korrektheit: Das Programm terminiert und liefert ein korrektes Ergebnis. $\{Q\}\ S\ \{R\}$ liest sich wie folgt: Wenn die Vorbedingung Q erfüllt ist dann terminiert S und die Nachbedingung R ist erfüllt.

In beiden Fällen wird keine Aussage gemacht was passiert wenn Q nicht erfüllt ist.

$$\text{Totale Korrektheit} = \text{Partielle Korrektheit} + \text{Terminierung}$$

Unser Fokus liegt auf totaler Korrektheit.

Zusammengesetzte Anweisung

Betrachte zwei Hoare Tripel:

- $\{x > 3\}\ x := x + 2\ \{x > 5\}$
- $\{x > 5\}\ y := x - 3\ \{y > 2\}$

Für das Programm dass beide Anweisung hintereinander ausführt können wir Folgendes schließen.

- $\{x > 3\}\ x := x + 2; y := x - 3\ \{y > 2\}$

Allgemein gilt: Wir müssen nur die Semantik von elementaren Anweisung definieren – daraus ergibt sich die Semantik für Programme.

Satz

Sei P, Q, R Prädikate und $S = S_1; S_2$ ein Programm.
Wenn $\{P\}\ S_1\ \{Q\}$ und $\{Q\}\ S_2\ \{R\}$ gilt dann auch $\{P\}\ S_1; S_2\ \{R\}$

Axiomatischen Semantik - Stärke von Zusicherungen

Beispiel

Wenn $\{x > 3\}\ S\ \{x > 5\}$ gilt (z.B. für S=„x:=x+2“) dann auch:

- $\{x > 3\}\ S\ \{x > 0\}$
- $\{x > 5\}\ S\ \{x > 5\}$
- $\{x > 5\}\ S\ \{x > 0\}$

Definition

Eine Formel P ist stärker als eine Formel Q wenn $P \Rightarrow Q$ (P impliziert Q), Q ist dann schwächer als P.

Das stärkste Prädikat ist false (**F**) und das Schwächste ist true (**T**).

Satz

Gilt $\{P\}\ S\ \{Q\}$ dann können wir die Vorbedingung P durch jede Stärkere P' und die Nachbedingung Q durch jede Schwächere Q' ersetzen, sodass $\{P'\}\ S\ \{Q'\}$.

Weakest Precondition

Idee: Wir definieren die Semantik in dem wir für jede Anweisung die schwächste Vorbedingung (weakest precondition) die notwendig ist um eine gegebene Nachbedingung zu erfüllen definieren.[2]

Definition (weakest precondition)

Sei R ein Prädikat und S ein Programm. Dann ist $\mathbf{wp}(\mathbf{S}, \mathbf{R})$ ein Prädikat, das die Menge aller Zustände mit der folgenden Eigenschaft definiert:

- Die Ausführung von S terminiert nach endlich vielen Schritten.
- Der nach Ausführung von S erreichte Zustand erfüllt R.

Es gilt dann also

- $\{wp(S, R)\}\ S\ \{R\}$ und
- für jedes Q mit $\{Q\}\ S\ \{R\}$ gilt $Q \Rightarrow wp(S, R)$.

[2] Wir folgen dem Formalismus in den Büchern [Gries (1989), O'Regan (2006)].

Weakest Precondition - Elementare Eigenschaften

- Excluded Miracle:

$$wp(S, \mathbf{F}) \equiv \mathbf{F}$$

- Monotonie:

$$\text{Aus } Q \Rightarrow R \text{ folgt } wp(S, Q) \Rightarrow wp(S, R)$$

- Distributivität der Konjuktion:

$$wp(S, Q) \wedge wp(S, R) \equiv wp(S, Q \wedge R)$$

- Distributivität der Disjunktion (für deterministisches S [3]):

$$wp(S, Q) \vee wp(S, R) \equiv wp(S, Q \vee R)$$

[3] In den Beispielen dieser LVA ist S immer deterministisch.

Weakest Precondition - Skip, Abort, Sequenz

- **Skip**: hat keinen Effekt

$$wp(\text{“}skip\text{“}, R) \equiv R$$

- **Abort**: produziert einen Fehler

$$wp(\text{“}abort\text{“}, R) \equiv \mathbf{F}$$

- **Sequenz**: wird schrittweise von hinten abgearbeitet

$$wp(\text{“}S1; S2\text{“}, R) \equiv wp(\text{“}S1\text{“}, wp(\text{“}S2\text{“}, R))$$

Weakest Precondition - Skip, Abort, Sequenz

Beispiel

- $wp(\text{“}skip; skip\text{“}, x \geq 5) \equiv wp(\text{“}skip\text{“}, wp(\text{“}skip\text{“}, x \geq 5)) \equiv wp(\text{“}skip\text{“}, x \geq 5) \equiv x \geq 5$
- $wp(\text{“}abort\text{“}, x \geq 5) \equiv \mathbf{F}$
- $wp(\text{“}S1; S2\text{“}, \mathbf{F}) \equiv wp(\text{“}S1\text{“}, wp(S2, \mathbf{F})) \equiv wp(\text{“}S1\text{“}, \mathbf{F}) \equiv \mathbf{F}$

Einschub: Logik - Die Operatoren CAND, COR

Manchmal können boolesche Ausdrücke nicht ausgewertet werden oder man will das aus Effizienzgründen vermeiden (Lazy Evaluation).

Führt man für solche Ausdrücke den Wert U (undefined) ein kann man aber mitunter trotzdem Konjunktionen und Disjunktionen auswerten.

Die Operationen conditional and/or (cand/cor) erweitern $\wedge/\vee$ auf U.

A cand B	T	F	U
T	T	F	U
F	F	F	F
U	U	U	U

A cor B	T	F	U
T	T	T	T
F	T	F	U
U	U	U	U

Einschub: Logik - Die Operatoren CAND, COR

A cand B	T	F	U
T	T	F	U
F	F	F	F
U	U	U	U

A cor B	T	F	U
T	T	T	T
F	T	F	U
U	U	U	U

Beispiel

Wir betrachten die Belegungen α, β, γ mit

X	$\alpha(X)$
x	F
y	U

X	$\beta(X)$
x	T
y	U

Dann gilt:

- $\alpha(x \text{ cand } y) = F$, $\alpha(x \text{ cor } y) = U$
- $\beta(x \text{ cand } y) = U$, $\beta(x \text{ cor } y) = T$ und
- $\alpha(y \text{ cand } x) = \alpha(y \text{ cor } x) = U$

Weakest Precondition - Zuweisung

Beispiel

Wir betrachten "$x := y^2 + 1$" mit der Nachbedingung $R : x \geq 2$.
Damit $x \geq 2$ nach der Zuweisung gilt muss vorher $y^2 + 1 \geq 2$ gelten.

$$wp(\text{“}x := y^2 + 1\text{“}, R) \equiv (y^2 + 1 \geq 2)$$

Wir haben in der Nachbedingung x durch $y^2 + 1$ ersetzt.

Beispiel

Wir betrachten "$x := 1/y$" mit der Nachbedingung $R : y \geq 0$.

- Da y nicht verändert wird bleibt R unverändert.
- Aber für $y = 0$ ist $1/y$ nicht definiert – Programm terminiert nicht.

$$wp(\text{“}x := 1/y\text{“}, R) \equiv (y \neq 0) \,\text{cand}\, (y \geq 0)$$

Wir haben eine zusätzliche Bedingung dass die Zuweisung terminiert.

Weakest Precondition - Zuweisung

Wir betrachten eine Zuweisung "$x := e$" wobei e ein Term ist.

Zunächst ersetzen wir x in der Nachbedingung R.

- $R_x[e]$ entspricht der Formel bei der wir in R die Variable x durch den Term e ersetzen. [4]

Mit unter kann der Term e nicht ausgewertet werden (z.B. $1/x$ für $x = 0$ oder $\log(y)$ für $y \leq 0$) und das Programm terminiert nicht korrekt.

- **DEF(e)** bezeichnet die Menge aller Zustände in denen der Term e berechnet werden kann.

Wir erhalten:

$$wp(\text{“}x := e\text{“}, R) = DEF(e) \,\text{cand}\, R_x[e]$$

[4] Die Notation in der Literatur ist R^x_e, aber um die Schreibweise (vor allem bei Übung und Prüfung) zu vereinfachen verwenden wir die gegebene Schreibweise.

Weakest Precondition - Zuweisung

Beispiel

$$wp(\text{“}x := y \cdot 12; x := 12/x\text{“}, x = 1)$$

Wir bearbeiten das Programm schrittweise und beginnen hinten.

$$wp(\text{“}x := y \cdot 12\text{“}, wp(\text{“}x := 12/x\text{“}, x = 1))$$

Jetzt betrachten wir $wp(\text{“}x := 12/x\text{“}, x = 1)$ und erhalten

$$wp(\text{“}x := 12/x\text{“}, x = 1) \equiv DEF(12/x) \text{cand} (x = 1)_x[12/x] \equiv$$
$$DEF(x) \text{ cand } (x \neq 0) \text{ cand } (12/x = 1)$$

Das setzen wir jetzt oben ein

$$wp(\text{“}x := y \cdot 12\text{“}, DEF(x) \text{ cand } x \neq 0 \text{ cand } 12/x = 1) \equiv$$
$$DEF(y \cdot 12) \text{ cand } (DEF(x) \text{ cand } x \neq 0 \text{ cand } 12/x = 1)_x [y \cdot 12] \equiv$$
$$DEF(y) \text{ cand } (DEF(y) \text{ cand } y \neq 0 \text{ cand } y = 1) \equiv DEF(y) \text{ cand } (y = 1)$$

Schlussendlich bekommen wir

$$wp(\text{“}x := y \cdot 12; x := 12/x\text{“}, x = 1) \equiv DEF(y) \text{ cand } (y = 1)$$

Weakest Precondition - Zuweisung

Mehr Beispiele (zum Selbststudium)

- $wp(\text{“}x := 1\text{“}, x = 1) \equiv DEF(1)\,\text{cand}(x = 1)_x[1] \equiv (1 = 1) \equiv \mathbf{T}$
- $wp(\text{“}x := 1\text{“}, x \neq 1) \equiv (1 \neq 1) \equiv \mathbf{F}$
- $wp(\text{“}a := a + 1\text{“}, a > 10) \equiv DEF(a + 1)\,\text{cand}(a > 10)_a[a + 1] \equiv DEF(a)\,\text{cand}(a + 1 > 10) \equiv DEF(a)\,\text{cand}\,(a > 9)$
- $wp(\text{“}a := 2 \cdot b + 1\text{“}, a = 13) \equiv DEF(2 \cdot b + 1)\,\text{cand}(a = 13)_a[2 \cdot b + 1] \equiv DEF(b)\,\text{cand}(2 \cdot b + 1 = 13) \equiv DEF(b)\,\text{cand}(b = 6)$
- $wp(\text{“}x := a/b\text{“}, P(x)) \equiv DEF(a/b)\,\text{cand}\,P(x)_x[a/b] \equiv DEF(a) \wedge DEF(b)\,\text{cand}(b \neq 0)\,\text{cand}\,P(a/b)$
- $wp(\text{“}x := e\text{“}, y = c) \equiv DEF(e)\,\text{cand}(y = c)$ $\quad c$ eine Konstante
- $wp(\text{“}x := 5\text{“}, x \neq 5) \equiv DEF(5)\,\text{cand}(x \neq 5)_x[5] \equiv (5 \neq 5) \equiv \mathbf{F}$
- $wp(\text{“}a := a + 1\text{“}, a < 0) \equiv DEF(a + 1)\ \text{cand}(a < 0)_a[a + 1] \equiv DEF(a)\,\text{cand}\,(a + 1 < 0) \equiv DEF(a)\,\text{cand}\,(a < -1)$

Weakest Precondition - Zuweisung

Noch mehr Beispiele (zum Selbststudium)

- $wp(\text{“}z := x; x := y; y := z\text{“}, x = X \wedge y = Y) \equiv$
  $wp(\text{“}z := x; x := y\text{“}, wp(\text{“}y := z\text{“}, x = X \wedge y = Y)) \equiv$
  $wp(\text{“}z := x; x := y\text{“}, DEF(z) \,\text{cand}\, x = X \wedge z = Y) \equiv$
  $wp(\text{“}z := x\text{“}, wp(\text{“}x := y\text{“}, DEF(z) \,\text{cand}\, x = X \wedge z = Y)) \equiv$
  $wp(\text{“}z := x\text{“}, DEF(y) \,\text{cand}\, DEF(z) \,\text{cand}\, y = X \wedge z = Y) \equiv$
  $DEF(x) \,\text{cand}\, DEF(y) \,\text{cand}(y = X \wedge x = Y)$

- $wp(\text{“}x := x \cdot x\text{“}, x^4 = 10) \equiv DEF(x \cdot x) \,\text{cand}(x^4 = 10)_x[x \cdot x] \equiv$
  $DEF(x) \,\text{cand}((x \cdot x)^4 = 10) \equiv DEF(x) \,\text{cand}(x^8 = 10)$

Weakest Precondition - if Anweisung

Wir betrachten eine if Anweisung "if B then S_1 else S_2". [5]

- B muss auswertbar sein
- Wenn B erfüllt ist dann muss S_1 die Nachbedingung erfüllen
- Wenn $\neg B$ erfüllt ist dann muss S_2 die Nachbedingung erfüllen

$$wp(\text{"if } B \text{ then } S_1 \text{ else } S_2\text{"}, R) = DEF(B) \text{ cand } (B \rightarrow wp(S_1, R)) \wedge (\neg B \rightarrow wp(S_2, R))$$

Beispiel

Betrachte "$wp($"if $x \geq 2$ then $x := x \cdot 2$ else $x := 4$", $x \geq 4)$

$wp($"if $x \geq 2$ then $x := x \cdot 2$ else $x := 4$", $x \geq 4) \equiv$

$DEF(x \geq 2) \text{ cand } ((x \geq 2) \rightarrow (x \geq 4)_x[x \cdot 2]) \wedge ((x < 2) \rightarrow (x \geq 4)_x[4]) \equiv$

$DEF(x) \text{ cand } ((x \geq 2) \rightarrow (2x \geq 4)) \wedge ((x < 2) \rightarrow (4 \geq 4)) \equiv DEF(x)$

[5] In der Literatur wir eine allgemeinere Form der Alternativanweisung betrachtet, die auch etwas komplexere (aber dem selben Prinzip folgende) Vorbedingungen verlangt.

Weakest Precondition - Arrays

Wir betrachten eindimensionale Arrays. Ein Array ist eine Funktion von seinem Indexbereich in den Wertebereich der Elemente.

Beispiel

Sei b als *var* b : *array*[0..2] *of integer*.

- Der Wert der Variablen b ist dann eine partielle Funktion von dem Indexbereich $\{0, 1, 2\}$ in die Menge der darstellbaren ganzen Zahlen.
- Eine Zuweisung $b[i] := \ldots$ bewirkt die Assoziation von b mit einem neuen Array, also einer neuen Funktion.

DEFX(**b**) bezeichnet den Indexbereich von b (im Beispiel $\{0, 1, 2\}$).

Definition

Gegeben ein Array b, dann ist $\mathbf{b}\langle\mathbf{i}, \mathbf{e}\rangle$ das Array das wir erhalten wenn wir den i-ten Eintrag von b auf e setzen (durch die Zuweisung $b[i] := e$).

Weakest Precondition - Arrays

Damit eine Zuweisung "$b[i] := e$" funktioniert muss

- i ein gültiger Index und
- e ein auswertbarer Ausdruck sein.

Die weakest precondition ist dann:

$$wp(\text{“}b[i] := e\text{“}, R) = i \in DEFX(b) \,\text{cand}\, DEF(e) \,\text{cand}\, R_b[b\,\langle i, e\rangle]$$

Beispiel

$wp(\text{“}a[3] = x \cdot a[4]\text{“}, a[3] \geq a[2]) \equiv$
$3 \in DEFX(a) \,\text{cand}\, DEF(x \cdot a[4]) \,\text{cand}\, (a[3] \geq a[2])_a[a\,\langle 3, x \cdot a[4]\rangle] \equiv$
$DEFX(x) \,\text{cand}\, 3, 4 \in DEFX(a) \,\text{cand}\, a\langle 3, x \cdot a[4]\rangle\,[3] \geq a\,\langle 3, x \cdot a[4]\rangle\,[2] \equiv$
$DEFX(x) \,\text{cand}\, 2, 3, 4 \in DEFX(a) \,\text{cand}\, x \cdot a[4] \geq a[2]$

Im letzten Schritt verwenden wir, dass
(i) $a\,\langle 3, x \cdot a[4]\rangle\,[3] = x \cdot a[4]$ und
(ii) der Eintrag 2 unverändert ist, dh $a\langle 3, x \cdot a[4]\rangle[2] = a[2]$.

Weakest Precondition - Arrays

Mehr Beispiele (zum Selbststudium)

Beispiel

$$wp(\text{“}b[1] := (x = 1) \wedge (y = 1)\text{“}, b[1] = b[2]) \equiv$$
$$1 \in DEFX(b) \,\text{cand}\, DEF((x = 1) \wedge (y = 1)) \,\text{cand}$$
$$(b[1] = b[2])_b[b\,\langle 1, (x = 1) \wedge (y = 1)\rangle] \equiv$$
$$1 \in DEFX(b) \,\text{cand}$$
$$(b\,\langle 1, (x=1)\wedge(y=61)\rangle\,[1] = b\,\langle 1, (x=1)\wedge(y=1)\rangle\,[2]) \equiv$$
$$1 \in DEFX(b) \,\text{cand}(((x = 1) \wedge (y = 1)) = b[2])$$

Beispiel

$$wp(\text{“}b[1] := 2 + a[2])\text{“}, b[1] = b[2]) \equiv$$
$$1 \in DEFX(b) \,\text{cand}\, DEF(2 + a[2]) \,\text{cand}(b[1] = b[2])_b[b <1, 2 + a[2] >] \equiv$$
$$1 \in DEFX(b) \,\text{cand}\, 2 \in DEFX(a) \,\text{cand}(b <1, 2+a[2] > [1] = b <1, 2+a[2] > [2])$$
$$\equiv 1 \in DEFX(b) \,\text{cand}\, 2 \in DEFX(a) \,\text{cand}\, (2 + a[2] = b[2])$$

Weakest Precondition - Arrays

Mehr Beispiele (zum Selbststudium)

Beispiel

$$
\begin{aligned}
&wp\big(\text{“}i := 1, b[1] := 2 + a[2]\text{“}, b[i] = b[2]\big) \equiv \\
&wp\big(\text{“}i := 1\text{“}, 1 \in DEFX(b) \wedge 2 \in DEFX(a)\, \mathrm{cand}(b[i] = b[2])_b[b <1, 2 + a[2]>]\big) \\
&\equiv wp\big(\text{“}i := 1\text{“}, 1 \in DEFX(b) \wedge 2 \in DEFX(a)\, \mathrm{cand} \\
&\qquad\qquad (b <1, 2 + a[2]>[i] = b <1, 2 + a[2]>[2])\big) \\
&\equiv wp\big(\text{“}i := 1\text{“}, 1 \in DEFX(b) \wedge 2 \in DEFX(a)\, \mathrm{cand}(b <1, 2 + a[2]>[i] = b[2])\big) \\
&\equiv (1 \in DEFX(b) \wedge 2 \in DEFX(a)\, \mathrm{cand}(b <1, 2 + a[2]>[i] = b[2]))_i[1] \\
&\equiv 1 \in DEFX(b) \wedge 2 \in DEFX(a)\, \mathrm{cand}(b <1, 2 + a[2]>[1] = b[2]) \\
&\equiv 1 \in DEFX(b) \wedge 2 \in DEFX(a)\, \mathrm{cand}(2 + a[2] = b[2])
\end{aligned}
$$

Weakest Precondition – Formale Verifikation

> *„Testing shows the presence, not the absence of bugs“*
> *Edsger W. Dijkstra*

Das Konzept der Weakest Precondition kann unmittelbar zur Verifikation eines Programms verwendet werden.

Für ein Programm S mit gegebener Spezifikation

- Berechne die weakest precondition $wp(S, R)$ für S und die gegebenen Nachbedingung R.
- Teste ob die in der Spezifikation gegebene precondition die berechnete weakest precondition $wp(S, R)$ erfüllt.

Weakest Precondition - Beispiel

Gegeben sei das Prädikat R: $(c = 1)$ und das folgende Programmstück S:

$$a := a + c; \ (1)$$
$$b := a - c; \ (2)$$
$$c := c \ / \ a; \ (3)$$

1. Geben Sie die Wertebelegungen der Variablen a, b und c während der Ausführung von S ausgehend von den Anfangszuständen: $\{a = -2, b = 0, c = 2\}$ und $\{a = 0, b = 1, c = 1\}$ tabellarisch an und überprüfen Sie in beiden Fällen, ob R nach der Ausführung von S erfüllt ist.
2. Berechnen Sie $wp(S, R)$. Halten Sie sich dabei an den in der Vorlesung eingeführten Formalismus.

Weakest Precondition - Beispiel

Geben Sie die Wertebelegungen der Variablen während der Ausführung von S ausgehend von den Anfangszuständen: $\{a = -2, b = 0, c = 2\}$ und $\{a = 0, b = 1, c = 1\}$ tabellarisch an und überprüfen Sie in beiden Fällen, ob R nach der Ausführung von S erfüllt ist.

S \ Var	a	b	c
0	-2	0	2
1	0	0	2
2	0	-2	2
3	-	-	2/0

S \ Var	a	b	c
0	0	1	1
1	1	1	1
2	1	0	1
3	1	0	1

Für den Anfangszustand $\{a = -2, b = 0, c = 2\}$ haben wir eine Division durch 0 und damit ein Abbruch (Abort).

Für den Anfangszustand $\{a = 0, b = 1, c = 1\}$ terminiert die Berechnung korrekt und das Prädikat R ist erfüllt.

Weakest Precondition - Beispiel

Berechnen Sie $wp(S, R)$. Halten Sie sich dabei an den in der Vorlesung eingeführten Formalismus.

$$wp(\text{“}a := a + c; b := a - c; c := c/a\text{“}, c = 1) \equiv$$
$$wp(\text{“}a := a + c; b := a - c\text{“}, wp(\text{“}c := c/a\text{“}, c = 1) \equiv$$
$$wp(\text{“}a := a + c; b := a - c\text{“}, DEF(c/a)\,\text{cand}(c = 1)_c[c/a]) \equiv$$
$$wp(\text{“}a := a + c; b := a - c\text{“}, DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}\,(a \neq 0)\,\text{cand}\,(c/a = 1)) \equiv$$
$$wp(\text{“}a := a + c; b := a - c\text{“}, DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a \neq 0)\,\text{cand}\,(c = a)) \equiv$$
$$wp(\text{“}a := a + c\text{“}, wp(\text{‘}b := a - c\text{‘}, DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a \neq 0)\,\text{cand}\,(c = a))) \equiv$$
$$wp(\text{“}a := a + c\text{“}, DEF(a - c)\,\text{cand}\,(DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a \neq 0)\,\text{cand}$$
$$(c = a))_b[a - c]) \equiv$$
$$wp(\text{“}a := a + c\text{“}, DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a \neq 0)\,\text{cand}\,(c = a)) \equiv$$
$$DEF(a + c)\,\text{cand}\,(DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a \neq 0)\,\text{cand}\,(c = a))_a[a + c] \equiv$$
$$DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a + c \neq 0) \wedge (c = a + c) \equiv$$
$$DEF(a)\,\text{cand}\,DEF(c)\,\text{cand}(a = 0) \wedge (c \neq 0)$$

Axiomatischen Semantik - while Schleife

Wir betrachten eine Schleife der Form: While B do S

Die Idee ist eine Zusicherung Q zu finden die sich durch S nicht ändert.

Definition

Eine **Schleifeninvariante** ist eine Zusicherung I sodass $\{I \wedge B\}\ S\ \{I\}$.

Für partielle Korrektheit und die Definition der Semantik reicht das.

Satz

Wenn $\{I \wedge B\}\ S\ \{I\}$ *gilt dann gilt für partielle Korrektheit auch* $\{I\}$ *While* B *do* S $\{I \wedge \neg B\}$.

$$\frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \text{While}\ B\ \text{do}\ S\ \{I \wedge \neg B\}}$$

Für totale Korrektheit müssen wir zeigen, dass die Schleife terminiert.

Axiomatischen Semantik - while Schleife

Idee: Um Terminierung zu zeigen sucht man einen Term t der

- aufgrund der Invariante immer ≥ 0 ist und
- in der Schleife immer um mindestens 1 verringert wird.

Der Term t wird auch **Schleifenvariante** genannt.

Schleifenvariante

Für totale Korrektheit haben wir dann:

$$\frac{I \Rightarrow t \geq 0, \{I \wedge B \wedge t = w\}\ S\ \{I \wedge t < w\}}{\{I\}\ \text{While } B \text{ do } S\ \{I \wedge \neg B\}}$$

wobei

- t ein Term über ganze Zahlen ist und
- w eine ganzzahlige Variable die in I, B, t und S nicht vorkommt.

Axiomatischen Semantik - while Schleife

Beispiel

while $r \geq y$ **do**
$r := r - y;\ q := q + 1;$
end while

mit Postcondition $\{x = y \cdot q + r \wedge 0 \leq r < y\}$.
Eine Schleifeninvariante $I := \{x = y * q + r \wedge 0 \leq r \wedge 0 < y\}$.
Als Schleifenvariante t wählen wir $t := r$.

- Es gilt $I \Rightarrow r \geq 0$
- Es gilt $\{I \wedge r \geq y \wedge r = w\}\ r := r - y; q := q + 1;\ \{I \wedge r < w\}$
- Es gilt $I \wedge \neg B \equiv I \wedge \neg(r \geq y) \equiv (x = y \cdot q + r) \wedge 0 \leq r < y$

Also auch:
$\{0 \leq r \wedge 0 < y \wedge x = y * q + r\}$
while $r \geq y$ **do**
$r := r - y;\ q := q + 1;$
end while
$\{x = y \cdot q + r \wedge 0 \leq r < y\}$

Weakest Precondition - While Schleife

Wir betrachten eine Schleife der Form: While B do S [6].

Wir können die weakest precondition dafür angeben, dass die Schleife

- nie durchläuft und die postcondition R erfüllt ist.

$$H_0(R) = DEF(B) \,\text{cand}\, \neg B \wedge R$$

- maximal einmal durchläuft und die postcondition R erfüllt ist.

$$H_1(R) = DEF(B) \,\text{cand}\, H_0(R) \vee (B \wedge wp(S, \neg B \wedge R))$$

- maximal k-mal durchläuft und die postcondition R erfüllt ist.

$$H_k(R) = DEF(B) \,\text{cand}\, H_0(R) \vee (B \wedge wp(S, H_{k-1}(R))$$

Wir erhalten

$$wp(\text{“While } B \text{ do } S\text{“}, R) = \exists k \geq 0 : H_k(R)$$

[6] In der Literatur wir eine allgemeinere Form der Schleife betrachtet, die auch etwas komplexere (aber dem selben Prinzip folgende) Vorbedingungen verlangt.

Weakest Precondition - While Schleife

Beispiel

Betrachten wir die Schleife S

while $r \geq y$ **do**
$r := r - y;\ q := q + 1;$
end while

mit Postcondition $R = \{x = y \cdot q + r \wedge 0 \leq r < y\}$.

- $H_0(R) \equiv r < y \wedge R \equiv (x = y \cdot q + r) \wedge 0 \leq r < y$
- $H_1(R) \equiv H_0(R) \vee ((r \geq y) \wedge wp(''r := r - y; q := q + 1;\ ``, (x = y \cdot q + r) \wedge 0 \leq r < y)) \equiv$
  $H_0(R) \vee ((r \geq y) \wedge (x = y(q+1) + r - y) \wedge 0 \leq r - y < y) \equiv$
  $H_0(R) \vee ((x = yq + r) \wedge y \leq r < 2y)$
- $H_2(R) \equiv H_0(R) \vee H_1(R) \vee ((x = yq + r) \wedge 2y \leq r < 3y)$
- $H_k(R) \equiv \bigvee_{i=0}^{k-1} H_i(R) \vee ((x = yq + r) \wedge (k-1)y \leq r < k \cdot y)$

$$wp(S, R) \equiv \exists k > 0 : ((x = yq + r) \wedge (k-1)\, y \leq r < k \cdot y) \equiv (x = yq + r) \wedge (y > 0 \wedge r \geq 0)$$

Weiterführende Literatur


David Gries (1989).
The Science of Programming.
Springer-Verlag, ISBN: 978-0-387-96480-5


Gerard O'Regan (2006).
Mathematical Approaches to Software Quality.
Springer-Verlag, ISBN: 978-1-84628-242-3


Elfriede Fehr (1988).
Semantik von Programmiersprachen.
Springer-Verlag, ISBN: 978-3-642-70271-6


Hanne Riis Nielson, Flemming Nielson (1992).
Semantics with Applications: A Formal Introduction.
Wiley Professional Computing, ISBN: 0-471-92980-8
http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html