Grundlagen der Programmierung (Vorlesung 11)

Ralf Möller, FH-Wedel

- Vorige Vorlesung
  - Prädikatenlogik mit Axiomen z.B. für natürliche Zahlen
  - Elemente von Programmiersprachen (Variablen, Felder)
  - Spezifikationen
- Inhalt dieser Vorlesung
  - Verifikation von Algorithmen
  - Zuweisungen und Kontrollstrukturen
- Lernziele
  - Grundlagen der systematischen Programmentwicklung

4.1 Spezifikationen

Notation

$\hookrightarrow$ $\qquad \{ \underbrace{V}_{\substack{Vor- \\ bedingung}} \} \quad \underbrace{S}_{\substack{An- \\ weisung}} \quad \{ \underbrace{P}_{\substack{Nach- \\ bedingung}} \}$

$\hookrightarrow$ Notation zur Begründung (Erklärung, Dokumentation), warum ein Programm(–Stück) korrekt arbeitet

$\{\ldots\}$ ist ein Kommentar,
hat bei der Programmausführung keinen Effekt
enthält ein Prädikat (Bedingung, Ausdruck, der zu `true` oder `false` ausgewertet werden kann)

V **und** P sind Prädikate

S ist ein Programm

Zustands-beschreibung	der Zustand eines Programms (Belegung der Variablen) kann mit solchen Prädikaten beschrieben werden
V	spezifiziert den Definitionsbereich der zu berechnenden Funktion
P	spezifiziert die zu berechnende Funktion
$\hookrightarrow$	V und P werden Zusicherungen *(assertions)* genannt

Beispiele	Programmspezifikation
ein Programm	soll ...
(1)	... eine Variable x auf den Wert 42 setzen, es soll immer funktionieren
(2a)	... die Summe zweier Zahlen x und y in einer Variablen s speichern
(2b)	... das Maximum zweier Zahlen x und y in einer Variablen r speichern
(3)	... die Werte zweier Variablen vertauschen
(4)	... 2 natürliche Zahlen x und y ganzzahlig mit Rest teilen und in den Variablen q und r Resultat und Rest speichern
(5)	... die 1. n natürlichen Zahlen $(n \geq 0)$ in einer Variablen s aufsummieren
(6)	... für eine natürliche Zahl n den größten Teiler $r < n$ bestimmen

Korrektheit	von Programmen
relativ	zu einer Spezifikation
$\hookrightarrow$	ein Programm(-stück) ist korrekt,
(1)	wenn es in einem Anfangszustand gestartet wird, in dem die Vorbedingung V gilt, d.h die Variablenbelegung gehört zum Definitionsbereich,
(2)	wenn es terminiert
(3)	wenn im Endzustand die Nachbedingung P gilt
(1+3)	partielle Korrektheit
$\hookrightarrow$	Vorbedingung darf verstärkt werden $\hookrightarrow$ Definitionsbereich wird eingeschränkt
$\hookrightarrow$	Nachbedingung darf abgeschwächt werden $\hookrightarrow$ *„es wird weniger berechnet“*

Beweisregel	Stärkung einer Vorbedingung und Schwächung einer Nachbedingung	
falls		
(1)	$V \rightarrow V_1$	gültig
(2)	$\{V_1\} \;\; S \;\; \{P_1\}$	korrekt
(3)	$P_1 \rightarrow P$	gültig
dann gilt	$\{V\} \;\; S \;\; \{P\}$	korrekt

Vorgehen	Programmentwurf
(1)	Problem spezifizieren
(1a)	Mit welchen Variablen soll gearbeitet werden $\Rightarrow$ Deklarationen
(1b)	Was soll berechnet werden $\Rightarrow P$
(1c)	Wann soll das Programm funktionieren $\Rightarrow V$
(2)	hieraus ein Programm konstruieren

Variablen Die in der Spezifikation und im Programm benutzten Variablen und deren Typ (Wertebereich) werden mit Hilfe von Deklarationen festgelegt.

Alle übrigen Namen sind Konstanten

$\hookrightarrow$ $\mathsf{var}\ x : \mathsf{N}_0$

$\hookrightarrow$ $\mathsf{var}\ b : \mathsf{B}$

$\hookrightarrow$ $\mathsf{var}\ f : \mathsf{array}\ [0 \ldots n-1]\ \mathsf{of}\ \mathsf{N}_0$

4.2 Anweisungen: Syntax und Semantik

4.2.1 Zuweisungen

Zuweisung bestehen aus einer Variablen und einem Ausdruck

$Variable \ := \ Ausdruck$ $\qquad x := 1$

$b := \mathsf{true}$

Zustand Programmzustand:
Belegung der Programmvariablen mit einem Wert zu einem bestimmten Zeitpunkt der Programmausführung

Bedeutung Semantik

informell der *Ausdruck* wird ausgewertet und der *Variablen* zugewiesen,
danach speichert die Variable den berechneten Wert

$\hookrightarrow$ Zustandstransformation,
Zustandsänderung

Bedeutung	Semantik
formal	durch eine Regel, die beschreibt, wie aus einer Nachbedingung und einer Zuweisungsanweisung die (schwächste) Vorbedingung berechnet werden kann
Notation	P sei ein Ausdruck, in dem x vorkommt, A sei irgendein beliebiger Ausdruck
$P[x \leftarrow A]$	ist der Ausdruck, in dem alle Vorkommen von x durch A ersetzt worden sind (und möglicherweise durch zusätzliche Klammern der syntaktische Aufbau von P erhalten bleibt).
Beweisregel	für Zuweisungen
	$\{P[x \leftarrow A]\}\ \ x := A\ \ \{P\}$
☞	Argumentation läuft rückwärts
☞	Prädikate dürfen in gleichwertige einfachere Prädikate umgeformt werden

Beispiele: Zuweisung x := A

- Beispiel 1
  - P: (x * x) > 0
  - A: y + 1
  - V = P[x ← A]: (y+1)*(y+1)>0 (korrekt unter dieser Vorbedingung)
- Beispiel 2
  - P: x = 42
  - A: 42
  - V = P[x ← A]: 42=42 (tautologisch, korrekt ohne bes. Vorbed.)
- Beispiel 3
  - P: x = 42
  - A: 43
  - V = P[x ← A]: 43=42 (unerfüllbar, nie korrekt)

Wann arbeitet das folgende Programm korrekt?

- var x : Z, f: array[0..n-1] of Z, i: N_0
- { 42 = 42 } x := 42 { x = 42 } immer
- { 42 = 43 } x := 43 { x = 42 } nie
- { x + 1 > 0 } x := x + 1 { x > 0 }
  - wenn x vor der Zuweisung größer oder gleich 0 ist
- { y * y = 16 } x := y * y { x = 16 }
  - wenn y vor der Zuweisung gleich 4 oder -4 ist

Erweiterung mehrfache Zuweisungen

Syntax $x_1, \ldots, x_n := A_1, \ldots, A_n$

Notation

$P[x_1 \leftarrow A_1, \ldots, x_n \leftarrow A_n]$

ist der Ausdruck, in dem alle Vorkommen von x_i durch A_i (gleichzeitig) ersetzt worden sind

Semantik

Beweisregel für mehrfache Zuweisungen

$$\{P[x_1 \leftarrow A_1, \ldots, x_n \leftarrow A_n]\} \quad x_1, \ldots, x_n := A_1, \ldots, A_n \quad \{P\}$$

$\hookrightarrow$ für Spezifikation häufig sehr nützlich

Beispiele: Parallele Zuweisung

- Beispiel 1
  - $\{ \text{true} \}\ x, y := 0, 1\ \{ y = 2^x \}$
  - $(y = 2^x)[x \leftarrow 0, y \leftarrow 1] \equiv 1 = 2^0 \equiv \text{true}$
- Beispiel 2
  - $\{y=w1 \wedge x=w2\}\ x, y := y, x\ \{x=w1 \wedge y=w2\}$
- Beispiel 3
  - $\{ x + y = r\}\ x, y := x-1, y+1\ \{ x + y = r\}$

$\equiv (x + y = r)\ [x \leftarrow x-1, y \leftarrow y+1]$

$\equiv (x-1)+(y+1)=r$

$\equiv x+y=r$

Programmkonstruktion

- $\{ y = 2^x \}\ x, y := x+1, E\ \{ y = 2^x \}$
- $(y = 2^x)[x \leftarrow x+1, y \leftarrow E]$

$\equiv E = 2^{x+1}$

$\equiv E = 2^x * 2$ (Arithm.)

$\equiv E = y * 2$ (Vorbed.)

- Programmstück: $x, y := x+1, y*2$
- Ohne Nachdenken, nur ausrechnen

Beweisregel für die Zuweisung: Präzisierung

- Vorbedingung verstärken
- Wann kann die rechte Seite ausgewertet werden?
- { "A kann ausgewertet werden" } $x := A$ { P }
- { Def(A), $P[x \leftarrow A]$ } $x := A$ { P }
- Beispiele:
  - $\{ y > 0, P[q \leftarrow x \text{ div } y]\}\ q := x \text{ div } y\ \{ P \}$
  - var $i{:}N_0$, $m{:}Z$, f:array [0..n-1] of Z (Def(f) = ?)

    $\{0 \leq i \wedge i < n \wedge P[\, m \leftarrow f[i]\,]\}\ m := f[i]\ \{ P \}$

    ($0 \leq i$ redundant wegen $i{:}N_0$)

Mittelwert berechnen, Vorbedingung finden

- var f : array [0..n-1] of R, am : R, i : N_0
- { ... } arithm. Mittel berechnen {am = amittel(f,n)}
- amittel(g,k) entspr. $\frac{1}{k}\sum_{j=0}^{k-1} g[j]$
- Berechnung in der Form, daß zu jedem Zeitpunkt der Mittelwert des Anfangstücks bis i in am steht
- Versuch 1
  - {...} i, am := 0, E {am = amittel(f,i)}
  - (am = amittel(f,i))[i<-0, am<-E]
  - E = (1/0) ?? Falsche Initialisierung

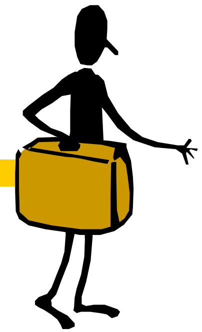
Mittelwert berechnen, Vorbedingung

- Versuch 2
  - {...} i, am := 1, E {am = amittel(f,i)}
  - (am = amittel(f,i))[i $\leftarrow$ 1, am $\leftarrow$ E]
  - E = (1/1) * f[0]
  - E = f[0]   (Arithm.)
- also: Vorbedingung n > 0

Mittelwert berechnen, Programmschritt

- {am = amittel(f,i)} i,am := i+1, E {am = amittel(f,i)}
- (am = amittel(f,i))[i $\leftarrow$ i+1, am $\leftarrow$ E]
- E = $$\frac{1}{i+1}\sum_{j=0}^{i+1-1} f[j] = \frac{1}{i+1}(f[i] + \sum_{j=0}^{i-1} f[j])$$
- E = $$\frac{1}{i+1}(f[i] + i(\frac{1}{i}\sum_{j=0}^{i-1} f[j])) = \frac{1}{i+1}(f[i] + i * am)$$
- Zusätzliche Vorbedingung: $0 \leq i < n$
- Programmschritt: i,am := i+1,(f[i] + (i * am))/(i+1)

Zusammenfassung, Kernpunkte



- Logik und die systematische Entwicklung von Programmen
- Zuweisung
- Fallunterscheidung

Was kommt beim nächsten Mal?

- Korrektheit von Anweisungfolgen