II.1.4 Verifikation

Freitag, 28. Oktober 2016 17:00

Wie überprüft man, ob ein Programm korrekt ist?

Bsp: $n = 4$

$i=4$ $res=1$ ⇝ $i=3$ $res=4$ ⇝ $i=2$ $res=3*4$ ⇝ $i=1$ $res=2*3*4$

Prog. ist korrekt (wobei $n! = 1$ für $n \leq 0$)

Hilfsmittel f. Prog.-Verif: Hoare Kalkül (Tony Hoare)

- Spezifikationen (zur partiellen Korrektheit) im Hoare-Kalkül:

$$\langle \varphi \rangle \; P \; \langle \psi \rangle$$

phi psi

Wenn vor der Ausführung des Prog. P die Vorbedingung φ gilt

und das Prog. P terminiert,

dann gilt hinterher die Nachbedingung ψ

Bsp: $\langle true \rangle \; P \; \langle res = n! \rangle$

- Dies ist eine semantische Aussage.

Hoare-Kalkül: 7 "syntaktische" Regeln, um solche Korrektheitsaussagen herzuleiten.

Vorteil v. H-Kalkül: • teilweise automatisierbar

- Verifikation überprüfbar
- Rahmen/Anleitung zur Verifikation

Schreibweise der Regeln:

$$\frac{\dots}{\dots}$$

Wenn die Aussagen oberhalb des Strichs wahr sind, dann sind auch die Aussagen unterhalb des Strichs wahr.

bzw.: Wenn man zeigen will, dass die Aussage unter d. Strich wahr ist, braucht man nur die Aussage über d. Strich zu zeigen.

Wenn nichts über d. Strich steht: Aussage unterhalb d. Strichs ist wahr.

Damit nach der Anweisung

$x = t;$

die Bedg. φ gilt, muss vorher $\varphi[x/t]$ gegolten haben.

$\langle \underbrace{Y = 5}_{\varphi[x/t]} \rangle \quad x = \underbrace{5}_{t}; \quad \langle \underbrace{Y = x}_{\varphi} \rangle$

$\langle 7 > 4 \rangle \quad x = 7; \quad \langle x > 4 \rangle$

$\langle \Gamma = \Gamma \rangle \quad x = \Gamma. \quad \langle x = \Gamma \rangle$

$\langle 5=5 \rangle \quad x=5; \quad \langle x=5 \rangle$

Verif. mit H-Kalkül:

Ergänze Prog-Text um Zusicherungen, die an der entspr. Stelle im Prog. immer gelten.

Wenn zwischen 2 Zusicherungen eine Prog.-Anweisung steht, dann muss Schritt von erster zu 2. Zusicherung mit einer Regel des H-Kalküls vorgenommen werden.

Konsequenzregel 1

$\langle \underbrace{5 \geq y}_{\varphi} \rangle \; x=5; \; \langle \underbrace{x \geq y}_{\psi} \rangle$ mit Zuweisungsregel

$\underbrace{3 \geq y}_{\alpha} \Rightarrow \underbrace{5 \geq y}_{\varphi}$

Mit Konsequenzregel 1 folgt dann

$\langle \underbrace{3 \geq y}_{\alpha} \rangle \quad x=5; \quad \langle \underbrace{x \geq y}_{\psi} \rangle$

Schreibweise:

- Wenn zwischen 2 Zusicherungen keine Anweisung ist, dann folgt die untere aus der oberen.
- Wenn zwischen 2 Zusicherungen eine Anweisung steht, dann entspricht das einer Anwendung einer Regel des H-Kalküls.

$\langle x > 1 \rangle$

Regel des [illegible]

Bsp:

$\langle x > 1 \rangle$
$\langle x-1 > 0 \rangle$
$x = x-1;$
$\langle x > 0 \rangle$

← forme $x > 1$ so um, dass nur noch "$x-1$" vorkommt

Konsequenzregel 2

$\langle 3 \geq y \rangle \quad x=5; \quad \langle x \geq y \rangle$

$\varphi \qquad\qquad\qquad \psi$

$x \geq y \quad \Rightarrow \quad x+1 \geq y$

$\psi \qquad\qquad\quad \beta$

Daraus folgt mit Konseq.-Regel 2:

$\langle 3 \geq y \rangle \quad x=5; \quad \langle x+1 \geq y \rangle$

$\varphi \qquad\qquad\qquad \beta$

Sequenzregel

φ

$\langle true \rangle$
$\langle 5=5 \rangle$
$x = 5;$
$\langle x = 5 \rangle$

$\langle true \rangle \; x=5; \; \} P$

$\langle x=5 \rangle$

ψ

$\langle x*x+6 = 31 \rangle$
$res = x*x+6;$
$\langle res = 31 \rangle$

$\langle x=5 \rangle \quad res = x*x+6; \quad \langle res = 31 \rangle$

$\psi \qquad\qquad Q \qquad\qquad \beta$

Sequenzregel erlaubt die Herleitung von

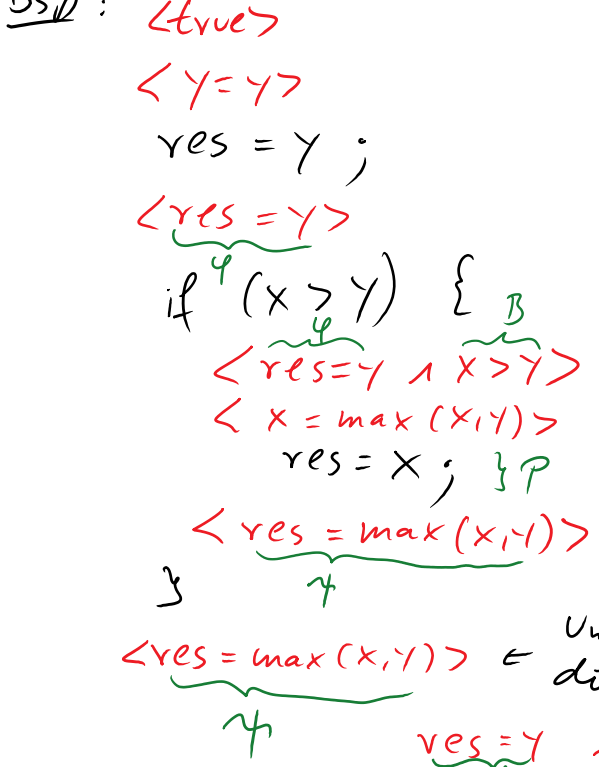
$\langle true \rangle \quad x=5; \quad res = x*x+6; \quad \langle res = 31 \rangle$

$\varphi \qquad P \qquad\qquad Q \qquad\qquad \beta$

Bedingungsregel 1

$\wedge$: und (&& in Java)
$\neg$: nicht (! in Java)

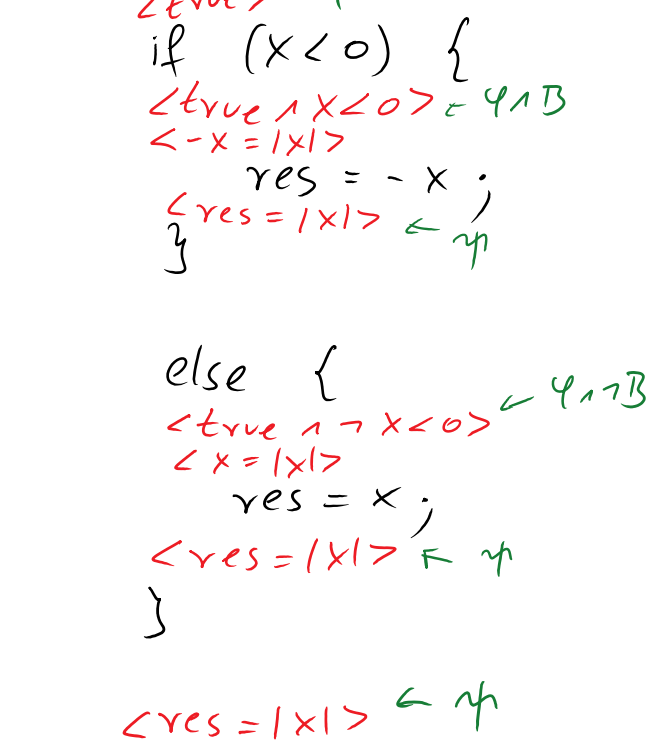
Bsp:



Um diese Zusicherung hier schreiben zu dürfen, muss man noch zeigen:

$res = y \wedge \neg x > y \Rightarrow res = max(x,y)$

Bedingungsregel 2



Schleifenregel

Hier: nur für while-Schleifen

analoge Regeln für do- und
for-Schleifen sind leicht möglich
(oder man übersetzt do- und for-Schl. in while-Schl.)

Idee: Zeige, dass φ eine
Schleifeninvariante ist:
Wenn φ vor der Schleife galt, dann gilt φ auch nach jeder
Ausführg. des Schleifenrumpfs. $\Rightarrow$ φ gilt auch nach der Schleife.

Gesucht ist Schleifeninvariante φ
mit folgenden Eigenschaften:

ⓐ φ ist wirklich Schleifeninvariante, d.h.
$\langle \varphi \wedge i>1 \rangle$ res = res * i; i = i-1; $\langle \varphi \rangle$

ⓑ φ folgt aus der Vorbedingung, d.h.
$i=n \wedge res=1 \Rightarrow \varphi$

ⓒ Aus φ und negierter Schleifenbedg. folgt Nachbedingung:
$\varphi \wedge \neg i>1 \Rightarrow res = n!$

Lösung: φ ist $i! * res = n!$

ⓐ $\langle i! * res = n! \wedge i>1 \rangle$
$\langle (i-1)! * res * i = n! \rangle$
res = res * i;
$\langle (i-1)! * res = n! \rangle$
i = i-1;
$\langle i! * res = n! \rangle$

ⓑ $i=n \wedge res=1 \Rightarrow i! * res = n!$

ⓒ $i! * res = n! \wedge \neg i>1 \Rightarrow res = n!$

Vorgehen: Lasse Prog. mit Beispiel-Eingaben laufen, um Zusammenhang zw.
i, n, res zu erkennen.

i	res	n
5	1	5
4	5	5
3	5·4	5
2	5·4·3	5
1	5·4·3·2	5

Suche Zusammenhang, der
ähnlich zur gewünschten Nachbedg.
res = n! ist.

Lösung: $i! \cdot res = n!$

Zusicherungen lassen sich
in Java direkt in den Programmtext schreiben (Assertions).
Um diese Assertions bei der Ausführung zu beachten:

java -ea ... ← Name der Klasse

Bislang: partielle Korrektheit

(d.h. korrekt, falls es terminiert)

Jetzt: Terminierung

Bsp: Variante der Schleife ist i

- $\underbrace{i>1}_{B} \Rightarrow \underbrace{i \geq 0}_{V}$
- $\{\underbrace{i=m \wedge i>1}_{V,B}\}$

  res = res * i;
  i = i - 1;

  $\}$ P

  $\{\underbrace{i}_{V} < m\}$

Denn:

$\{i = m \wedge i > 1\}$
$\{i-1 < m\}$
res = res * i;
$\{i-1 < m\}$
i = i - 1;
$\{i < m\}$

Verifikation hilft

- bei sicherheitskrit. Programmen
- beim Entwurf v. Programmen (Programmierer sollte sich beim Programmieren Schleifeninvariante u. Schleifenvariante überlegen).

Terminierung des Additionsprogramms:

Variante x

- $\underbrace{x>0}_{\text{Schleifenbedg.}} \Rightarrow \underbrace{x \geq 0}_{\text{Variante}}$
- $\{x = m \wedge x > 0\}$
  $\{x-1 < m\}$
  x = x - 1;

$\langle x < m \rangle$

res = res + 1;

$\langle x < m \rangle$

Terminierung des Subtraktionsprogramms

Variante: $x - z$

- $\underbrace{x > z}_{\text{Schl.-Bedg.}} \Rightarrow \underbrace{x - z \geq 0}_{\text{Variante}}$ ✓

- $\langle x - z = m \wedge x > z \rangle$
  $\langle x - (z+1) < m \rangle$
  z = z + 1;
  $\langle x - z < m \rangle$

  res = res + 1;

  $\langle x - z < m \rangle$