II.1.4 Verifikation

Dienstag, 17. Oktober 2017 11:00

- Wir können einfache Prog. Schreiben. Wie überprüft man, ob Prog. korrekt sind?
- Korrektheit: erfüllt das Programm seine Spezifikation?
- Bsp: $n = 4$

$i = 4$

$res = 1$

↓

$res = 4$

$i = 3$

↓

$res = 4 * 3$

$i = 2$

↓

$res = 4 * 3 * 2$

$i = 1$

- Technik zur Verifikation: Hoare-Kalkül (Tony Hoare)

7 Regeln zur Herleitung von Korrektheitsaussagen.

Vorteile:

- teilweise automatisierbar
- Verifikation leicht überprüfbar
- gibt Rahmen / Anleitung zur Verifikation

- Korrektheitsaussagen haben folgende Form:

$$\underbrace{\langle \varphi \rangle}_{\text{Vorbedingung}} \; P \; \underbrace{\langle \psi \rangle}_{\text{Nachbedingung}}$$

"phi" — Prog. — "psi"

bedeutet:

Wenn vor der Ausführung des Programms P die Vorbedingung φ gilt und wenn die Ausführung

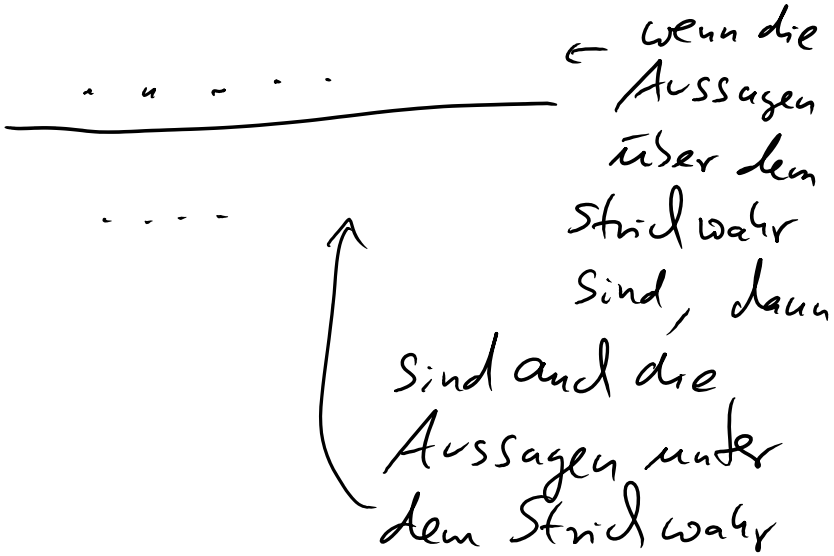
von P terminiert, dann gilt hinterher die Nachbedingung φ.

Im Fakultäts-Bsp P wollen wir zeigen:

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

Dies ist die Spezifikation von P u. Ziel ist, dies zu beweisen.

Hoare-Kalkül hat 7 Regeln



Zuweisungsregel

über Strich steht nichts, d.h. alle Aussagen der Form

$\langle \varphi[x/t] \rangle \; x = t; \quad \langle \varphi \rangle$

Bsp:

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

$\varphi[x/5]$ — t — φ

$\langle 5 > 4 \rangle \; x = 5; \; \langle x > 4 \rangle$

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

Schreibweise:
ergänze Programm um Zusicherungen (Assertions), die ausdrücken, welche Aussagen an best. Stelle im Prog. gelten. Schritt v. einer Zusicherung zur nächsten geschieht durch Regel d. Hoare-Kalküls.

Konsequenzregel 1

Verstärkung der Vorbedingung

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

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

Mit Konseq.-Regel 1 erhält man:

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

Schreibweise:

- Man darf 2 Zusicherungen direkt untereinander schreiben, wenn die untere aus der oberen Zusicherung folgt.
- Wenn zwischen 2 Zusicherungen eine Anweisung steht, dann muss d. Schnitt v. oberer zu

unterer Zusicherung mit einer Regel d. H-Kalküls geschehen.

Konsequenzregel 2

Abschwächung der Nachbedingung

Sequenzregel

<true>
<5=5>
x = 5;
<x=5>
<x*x+6 = 31>
res = x*x+6;
<res = 31>

d.U.:

<true> x=5; <x=5>

<x=5> res = x*x+6; <res=31>

Bedingungsregel 1

$\wedge$ bedeutet "und" (&&)

$\neg$ bedeutet "nicht" (!)

<true>

<y=y>

res = y;

<res=y> $\leftarrow \varphi$

B

if (x > y) {

<res=y $\wedge$ x>y> $\leftarrow \varphi \wedge B$

<x = max(x,y)>

res = x;

<res = max(x,y)> $\leftarrow \psi$

}

<res = max(x,y)> $\leftarrow \psi$

↑

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

$\underbrace{res=y}_{\varphi} \wedge \underbrace{\neg x>y}_{\neg B} \Rightarrow \underbrace{res = max(x,y)}_{\psi}$

Bedingungsregel 2

<true>

```
if (x < 0) {
<true ∧ x<0> = <x<0>
<-x = |x|>
    res = -x;
<res = |x|>
}

else {
<true ∧ ¬x<0> = <x≥0>
<x = |x|>
    res = x;
<res = |x|>
}

<res = |x|>
```

Schleifenregel

Zeige nur Regel für while-Schleifen. (Analoge Regeln können für do- und for-Schleife formuliert werden.)

Idee: finde eine Schleifeninvariante φ. D.h.: wenn φ vor Ausführung des Schleifenrumpfs P gilt und die Schleifenbedingung B zutrifft,

dann gilt φ auch nach Ausführung des Schleifenrumpfs P.

Ziel: Finde eine geeignete Schleifeninvariante φ, die die folgenden 3 Bedingungen erfüllt:

(a) φ ist wirklich eine Schleifeninvariante, d.h. $\langle \varphi \wedge B \rangle \; P \; \langle \varphi \rangle$

(b) φ muss aus der Vorbedingung vor der while-Schleife folgen.

(c) Aus $\varphi \wedge \neg B$ muss die gewünschte Nachbedingung folgen.

Im Fakultätsbsp: Suche φ, so dass:

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

(b) $i = n \wedge res = 1 \Rightarrow \varphi$

(c) $\varphi \wedge \neg i > 1 \Rightarrow res = n!$

Um φ zu finden, gehe v. Nachbedingung aus u. modifiziere sie so, dass sie in jedem Schleifendurchlauf gilt.

Hierzu: lasse Schleife mit Beispielwerten "laufen" und untersuche, wie sich die Werte der versch. Prog-Variablen zu-

einander verhalten.

i	res	n
5	1	5
4	5	5
3	5.4	5
2	5.4.3	5
1	5.4.3.2	5

$i! \cdot res = n!$ ist Schleifeninvariante Q und sie erfüllt Beding. ⓐ, ⓑ, ⓒ

Zusicherungen können als Assertions ("assert ...") in den Java-Code geschrieben werden. Wenn man das Prog. mit

```
java -ea ...
```

ausführt, dann werden die Assertions zur Laufzeit überprüft.

Bislang haben wir nur partielle

Korrektheit untersucht. Es
fehlt noch Beweis der Ter-
minierung.
Ziel: Finde eine Variante für
jede Schleife, die nie negativ
wird und die bei jedem
Schleifendurchlauf kleiner wird.

---

2 kreative Schritte:
- finde die Schleifeninvarianten
- finde die Schleifenvarianten

Über diese beiden Konzepte
sollte man als Prog. nachdenken,
wenn man Schleifen schreibt.

Im allg. kann es für Terminierung
nötig sein, nicht nur die
Schleifenbedingung B, sondern
auch zusätzliche Schleifeninvarianten

zu betrachten. D.h:

$B \wedge \varphi \Rightarrow V \geq 0$ und

$\langle V = m \wedge B, \varphi \rangle$

P

$\langle V < m \rangle$