DETAILLIERTE LSG. LOOP - INVARIANTE

anhand Serie 7. Aufgabe 2 a)

von Charlotte Heep,
cheep01@student.ethz.ch

```
public int compute(int n) {
    // Precondition:  n >= 0
    int x;
    int res;

     x = 0;
    res = x;

    // Loop Invariante:
    while (x <= n) {
        res = res + x;
        x = x + 1;
    }
    // Postcondition:  res == ((n + 1) * n) / 2
    return res;
}
```

?

(A) Invariante finden

(B) Invariante verifizieren

(A) INVARIANTE FINDEN

1. Tabelle erstellen mit allen Variablen, die im loop geschrieben werden

i	res	alternative Notation res	x	
0	0	0	0	// initiale Werte vor dem loop
1	0	0+0	1	// Werte nach der 1. Iteration
2	1	0+0+1	2	
3	3	0+0+1+2	3	
4	6	0+0+1+2+3	4	
5	10	0+0+1+2+3+4	5	
6	15	0+0+1+2+3+4+5	6	
⋮				

↳ häufig gibt diese Schreibweise mehr Aufschluss

2. Allgemeine Form abhängig von i finden

$$res = \sum_{k=0}^{i-1} = \frac{(i-1)\cdot i}{2} \quad \text{(Gauss)}$$

$$x = i$$

3. Umformen, sodass i eliminiert wird

$$res = \frac{(i-1)\cdot i}{2} \overset{x=i}{=} \boxed{\underbrace{\frac{(x-1)\cdot x}{2}}_{\text{partielle Invariante}} = res} \quad (1)$$

hier sehr einfacher Fall, im allgemeinen eine Gleichung nach i umformen und einsetzen

4. Loop-Condition um Termination erweitern

Loop-Condition: $x \leq n$

Wert von x bei Termination: $x = n+1$

→ Erweiterung $x \leq n \;||\; x = n+1$

$$\Leftrightarrow \boxed{x \leq n+1} \quad (2)$$

5. Alle anderen Annahmen, die wir treffen können, sammeln

Precondition ⇒ $\boxed{n \geq 0}$ (3)

Loop-Invariante: (1) && (2) && (3)

$$\{ res = \frac{(x-1)\cdot x}{2} \;\&\&\; x \leq n+1 \;\&\&\; n \geq 0 \}$$

! Wenn genug Zeit ist prüfen, ob nicht noch Conditions hinzugefügt werden müssen

B INVARIANTE VERIFIZIEREN

```
public int compute(int n) {
    // Precondition:  n >= 0
    int x;
    int res;

     x = 0;
    res = x;

    // Loop Invariante:
    while (x <= n) {
        res = res + x;
        x = x + 1;
    }
    // Postcondition:  res == ((n + 1) * n) / 2
    return res;
}
```

ANFORDERUNGEN:

1. Precondition ⇒ Invariante
2. { Condition ∧ Invariante }
   Body;
   { Invariante } ist ein valides Tripel.
3. ¬ Condition ∧ Invariante ⇒ Postcondition

unsere Lösung: $\{ res == \frac{(x-1)\cdot x}{2} \;\&\&\; x \leq n+1 \;\&\&\; n \geq 0 \}$

1. 1. Precondition ⇒ Invariante

Wir dürfen die Precondition und alle Statements aus dem Code vor dem Loop annehmen

Dann gilt

$n \geq 0$

$x = 0$

$res = 0$

nun prüfen wir, ob die LI dann auch stimmt

i) $res = \frac{(x-1)\cdot x}{2}$ | $res = 0,\ x = 0$

$\Leftrightarrow 0 = \frac{(-1)\cdot 0}{2}$ ✓

ii) $x \leq n+1$ | $x = 0$

$\Leftrightarrow 0 \leq n+1$ | $0 \leq n$

$\Leftrightarrow 0 \leq n \leq n+1$ ✓

iii) $0 \leq n$ gilt ohnehin per Annahme ✓

alle Teile der LI folgen aus unseren Annahmen

1) ✓ Wenn Precondition gilt, gilt auch LI

2.:

2. { Condition ∧ Invariante }
   Body;
   { Invariante } ist ein valides Tripel.

Wir dürfen Condition und LI annehmen

Also:

$x \leq n$	$x \leq n+1$
$res = \frac{(x-1)\cdot x}{2}$	$n \geq 0$

Dann führen wir den Code aus:

$$res' = res + x$$
$$x' = x + 1$$

Ich bezeichne hier die upgedateten Variablen zur besseren Unterscheidung mit res' und x'

Nun prüfen wir die LI für die upgedateten Werte

$$res = \frac{(x-1)\ x}{2}$$

$$= res' - x = \frac{(x-1)\cdot(x)}{2} \quad | \ res = res' - x$$

$$\Leftrightarrow \quad res' = \frac{(x-1)\cdot(x) + 2x}{2} \quad | \ + x$$

$$= \frac{(x+1)\cdot x}{2}$$

$$= res' = \frac{(x')\cdot(x'-1)}{2} \quad | \ x' = x+1$$

Anm.: Ich habe hier einen Vorwärtsschluss gemacht, aber weil alle Beziehungen Gleichheit oder doppelte Implikation sind könnte man auch genauso Rückwärtsschließen

LI

außerdem $x \leq n \Leftrightarrow x+1 = x' \leq n+1$

n bleibt gleich, also gilt $n \geq 0$ weiter

⇒ LI gilt auch für upgedatete Werte

2) ✓ Das Tripel ist gültig

3.:

3. ¬ Condition ∧ Invariante ⇒ Postcondition

Wir dürfen ¬Condition und Invariante annehmen

Also gilt:

$!(x \leq n)$ $\quad$ $x \leq n+1$

$res = \frac{x \cdot (x-1)}{2}$ $\quad$ $n \geq 0$

$!(x \leq n) \land x \leq n+1$

$\Leftrightarrow \quad x > n \land x \leq n+1$

$\Leftrightarrow \quad \boxed{x = n+1}$ weitere Annahme, die wir nutzen dürfen

$res = \frac{x \cdot (x-1)}{2} \land x = n+1$

$\Rightarrow \quad res = \underbrace{\frac{(n+1) \cdot (n)}{2}}_{\text{Postcondition}}$

3) ✓ Wenn ¬Condition und LI gelten, gilt auch PC

Alle 3 Bedingungen stimmen, die LI ist also gültig

JETZT WO WIR DAS KLAR HABEN:

Problem von letzter Woche

```java
public static int factorial(int n) {
    // Precondition: n >= 0
    int counter;
    int result;

    counter = n;
    result = 1;
    // Loop-Invariante: ??
    while (counter > 0) {
        result = result * counter;
        counter = counter - 1;
    }

    // Postcondition: result = n!
    return result;
}
```

Wir hatten

LI : result = n! / counter ! && counter ≥ 0

gefunden

Wir wollen nun die Korrektheit prüfen:

1. Precondition ⇒ Invariante
2. { Condition ∧ Invariante }
   Body;
   { Invariante } ist ein valides Tripel.
3. ¬ Condition ∧ Invariante ⇒ Postcondition

1. : 1. Precondition ⇒ Invariante

Wir dürfen die PC und alle Statements im Code vor dem Loop annehmen

Es gilt also:

$n \geq 0$

counter = n

result = 1

$$\Rightarrow \frac{n!}{counter!}$$

$$= \frac{n!}{n!} \quad | \ counter = n$$

$$= 1$$

$$= result \quad | \ result = 1$$

$$\Rightarrow \frac{n!}{counter!} = result$$

außerdem folgt aus den Annahmen counter = n ≥ 0

gemeinsam ist das

LI = n! / counter && counter ≥ 0

1) ✓ Precondition ⇒ Invariante

2.

2. { Condition ∧ Invariante }
   Body;
   { Invariante } ist ein valides Tripel.

Wir können die Loop-Condition und LI annehmen

Es gilt also:

$$counter > 0$$

$$result = \frac{n!}{counter!}$$

$$counter \geq 0$$

Wenn wir nun den Code ausführen, kommen wir auf die upgedateten Werte

$$result' = result * counter$$

$$counter' = counter - 1$$

$$\Rightarrow result' = result * counter$$

$$= \frac{n!}{counter!} * counter \quad | \; result = \frac{n!}{counter!}$$

$$= \frac{n! * counter}{counter \cdot (counter - 1)!} \quad | \; \text{Def. } !$$

$$= \frac{n!}{(counter - 1)!}$$

$$= \frac{n!}{(counter')!}$$

$$\Rightarrow result' = \frac{n!}{counter'!}$$

Hier gilt auch $n' = n$, weil n im Loop nicht geschrieben wird

LI für upgedatete values

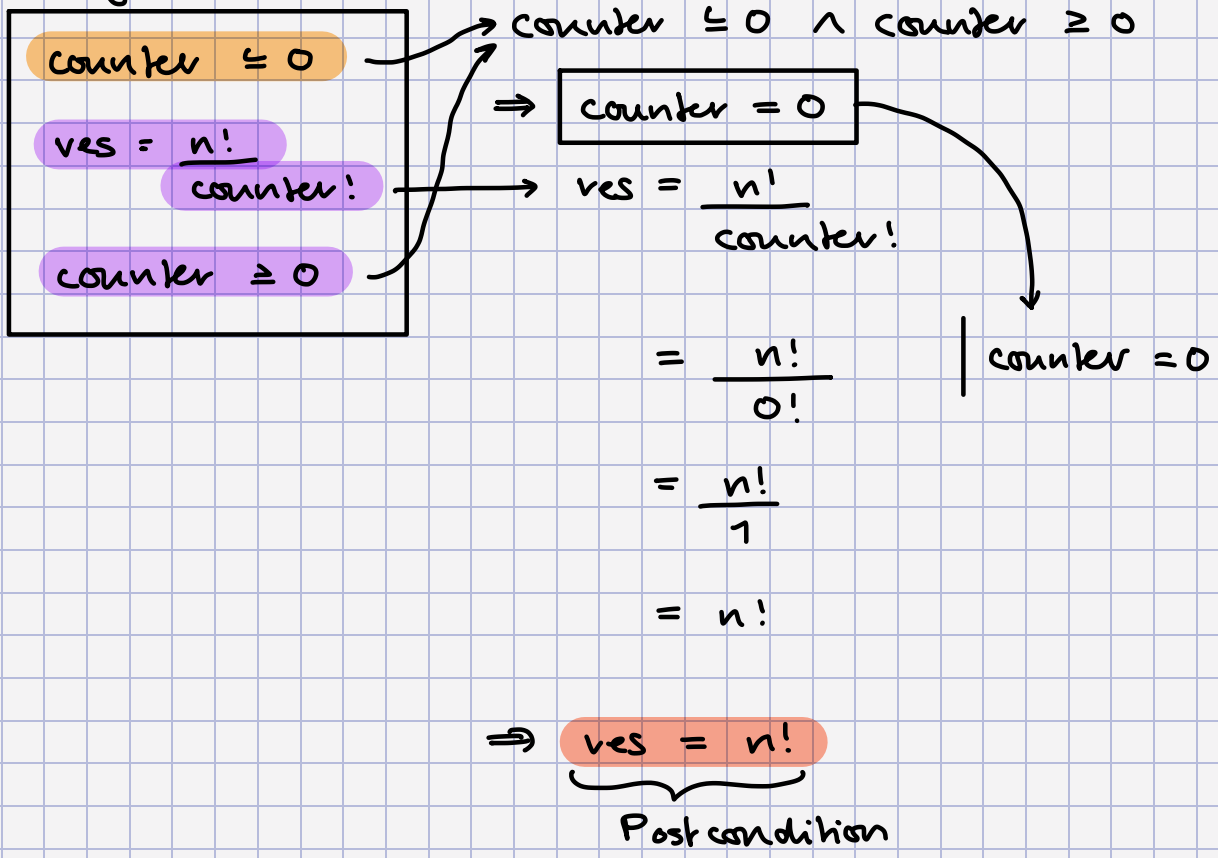
2) ✓ LI ∧ condition ⇒ LI

3.

3. ¬ Condition ∧ Invariante ⇒ Postcondition

Gib die ¬ condition und LI annehmen

Also gilt:



3) ✓ ¬ condition && LI ⇒ PC

Alle 3 Bedingungen sind erfüllt, die Invariante ist korrekt