RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Übungsblatt 5
25.11.2013

Übung zur Vorlesung Programmierung

Aufgabe T10

Gegeben sei folgendes Java-Programm P:

$\langle\varphi\rangle$ (Vorbedingung)

```
z = 0;
while (i < x){
  i = i + 1;
  j = 0;
  while (j < x){
    j = j + 1;
    z = z + x;
  }
}
```

$\langle\psi\rangle$ (Nachbedingung)

Als Vorbedingung φ für das oben aufgeführte Programm P gelte $x \geq 0$ und als Nachbedingung ψ gelte $z = x^3$. Beweisen Sie die Gültigkeit des Hoare-Tripels $\langle\varphi\rangle\, \mathsf{P}\, \langle\psi\rangle$ unter Verwendung des Hoare-Kalküls (inklusive Terminierung) oder widerlegen Sie die Gültigkeit durch Angabe eines geeigneten Gegenbeispiels.

Lösungsvorschlag

Das Programm ist tatsächlich falsch. Hier ein Beispiel wie ein Korrektheitsbeweisversuch aussehen könnte:

Partielle Korrektheit:

$\langle x \geq 0 \wedge i = 0 \rangle$
$\langle x \geq 0 \wedge i = 0 \wedge 0 = 0 \rangle$

```
z = 0;
```

$\langle x \geq 0 \wedge i = 0 \wedge z = 0 \rangle$
$\langle x \geq 0 \wedge z = i \cdot x^2 \wedge i \leq x \rangle$

```
while (i < x){
```

$\langle x \geq 0 \wedge z = i \cdot x^2 \wedge i \leq x \wedge i < x \rangle$
$\langle x \geq 0 \wedge z = (i + 1 - 1) \cdot x^2 \wedge i + 1 \leq x \rangle$

```
   i = i + 1;
```

$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 \wedge i \leq x \rangle$
$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 \wedge i \leq x \wedge 0 = 0 \rangle$

```
   j = 0;
```

$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 \wedge i \leq x \wedge j = 0 \rangle$
$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 + j \cdot x \wedge i \leq x \wedge j \leq x \rangle$

```
   while (j < x){
```

$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 + j \cdot x \wedge i \leq x \wedge j \leq x \wedge j < x \rangle$
$\langle x \geq 0 \wedge z + x = (i - 1) \cdot x^2 + (j + 1) \cdot x \wedge i \leq x \wedge j + 1 \leq x \rangle$

```
      j = j + 1;
```

$\langle x \geq 0 \wedge z + x = (i - 1) \cdot x^2 + (j) \cdot x \wedge i \leq x \wedge j \leq x \rangle$
$\langle x \geq 0 \wedge z + x = (i - 1) \cdot x^2 + j \cdot x \wedge i \leq x \wedge j \leq x \rangle$

```
      z = z + x;
```

$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 + j \cdot x \wedge i \leq x \wedge j \leq x \rangle$

```
   }
```

$\langle x \geq 0 \wedge z = (i - 1) \cdot x^2 + j \cdot x \wedge i \leq x \wedge j \leq x \wedge \neg j < x \rangle$
$\langle x \geq 0 \wedge z = i \cdot x^2 \wedge i \leq x \rangle$

```
}
```

$\langle x \geq 0 \wedge z = i \cdot x^2 \wedge i \leq x \wedge \neg i < x \rangle$
$\langle z = x^3 \rangle$

Gegenbeispiel:

Könnten wir einen korrekten Beweis schreiben, müsste gleichzeitig folgendes wahr sein, was nicht stimmt.

$\langle x = 1 \wedge i = 1 \rangle$

```
P
```

$\langle z = 0 \rangle$

Terminierung:

Wir wählen als Variante für die äußere Schleife $V = x - i$. Hiermit lässt sich die Terminierung dieser Schleife (mit Schleifenbedingung $B = i < x$) beweisen, denn es gilt:

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow i < x \Leftrightarrow x - i > 0 \Leftrightarrow V > 0$ und

- $\langle x - i = m \wedge i < x \rangle$
  $\langle x - (i+1) = m - 1 \rangle$

```
i = i + 1;
```

  $\langle x - (i) = m - 1 \rangle$
  $\langle x - i < m \rangle$

```
j = 0;
```

  $\langle x - i < m \rangle$

```
while(j < x){
```

  $\langle x - i < m \wedge j < x \rangle$
  $\langle x - i < m \rangle$

```
   j = j + 1;
```

  $\langle x - i < m \rangle$

```
   z = z + x;
```

  $\langle x - i < m \rangle$

```
}
```

  $\langle x - i < m \wedge \neg j < n \rangle$
  $\langle x - i < m \rangle$

Wir wählen als Variante für die innere Schleife $V = x - j$. Hiermit lässt sich die Terminierung dieser Schleife (mit Schleifenbedingung $B = j < x$) beweisen, denn es gilt:

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow j < x \Leftrightarrow x - j > 0 \Leftrightarrow V > 0$ und
- $\langle x - j = m \wedge j < x \rangle$
  $\langle x - (j+1) = m - 1 \rangle$

```
j = j + 1;
```

  $\langle x - (j) = m - 1 \rangle$
  $\langle x - j < m \rangle$

```
z = z + x;
```

  $\langle x - j < m \rangle$

Aufgabe T11

Bestimmen Sie mit der Monte-Carlo-Methode das Volumen einer Kugel mir Radius 6 cm, aus der ein Zylinder mit Radius 3 cm und endloser Länge so ausgeschnitten wurde, daß die Mittelachse des Zylinders genau 3 cm am Mittelpunkte der Kugel vorbeigeht.

Lösungsvorschlag

Wir nehmen an, daß der Mittelpunkt der Kugel im Ursprung liegt und die Achse des Zylinders parallel zur z-Achse verläuft. Weiter soll die Achse die xy-Ebene im Punkt $(3, 0, 0)$ schneiden. Der folgende Code approximiert per Monte-Carlo-Methode das Volumen des oben beschriebenen Körpers (dabei ersetze man 'mod' durch '%'):

```
public static void main(String[ ] args) {
  int inCounter = 0;
  int outCounter = 0;
  int samples = 1000000;
  double radius = 6.0;
```

```
  double unitVolume = 8 * radius * radius * radius;
  for (int i = 1; i <= samples; i++) {
    if(i mod 1000 == 0) {
      double fraction = inCounter/(double)(inCounter + outCounter);
      double volume = fraction * unitVolume;
      volume = Math.round(volume * 100)/100.0;
      System.out.println("Volume after " + i + " samples:" + volume + " cm3");
    }
    double x = Math.random() * 2 * radius - radius;
    double y = Math.random() * 2 * radius - radius;
    double z = Math.random() * 2 * radius - radius;
    if(x * x + y * y + z * z > radius * radius) {
      // Not in sphere
      outCounter++;
      continue;
    }
    // Assume cylinder is aligned to the z axis and
    // hits the point (radius/2, 0, 0)
    x -= radius/2;
    if(x * x + y * y < radius * radius/4.0) {
      // In sphere, but also inside cylinder
      outCounter++;
      continue;
    }
    inCounter++;
  }
}
```

Aufgabe H10 (13 Punkte)

Gegeben sei folgendes Java-Programm P:

$\langle\varphi\rangle$ (Vorbedingung)

```
i = 0;
max = 0;
while (i < n){
  if (a[i] > max){
    max = a[i];
  }
  i = i + 1;
}
```

$\langle\psi\rangle$ (Nachbedingung)

Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\mathtt{n} \geq 0$ und als Nachbedingung ψ gelte $\mathtt{max} = max(\bigcup_{k=0}^{n-1}\{a[k]\} \cup \{0\})$. Beweisen Sie die Gültigkeit des Hoare-Tripels $\langle\varphi\rangle\, \mathtt{P}\, \langle\psi\rangle$ unter Verwendung des Hoare-Kalküls (inklusive Terminierung) oder widerlegen Sie die Gültigkeit durch Angabe eines geeigneten Gegenbeispiels.

Lösungsvorschlag
Partielle Korrektheit:

$\langle \texttt{n} \geq 0 \rangle$
$\langle \texttt{n} \geq 0 \wedge 0 = 0 \wedge 0 = 0 \rangle$
`i = 0;`
$\langle \texttt{n} \geq 0 \wedge \texttt{i} = 0 \wedge 0 = 0 \rangle$
`max = 0;`
$\langle \texttt{n} \geq 0 \wedge \texttt{i} = 0 \wedge \texttt{max} = 0 \rangle$
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i} \leq \texttt{n} \rangle$
`while (i < n){`
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{i} < \texttt{n} \rangle$
`if (max < a[i]){`
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{i} < \texttt{n} \wedge \texttt{max} < \texttt{a}[\texttt{i}] \rangle$
$\langle \texttt{a}[\texttt{i}] = max(\bigcup_{k=0}^{\texttt{i}+1-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i}+1 \leq \texttt{n} \rangle$
`max = a[i];`
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}+1-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i}+1 \leq \texttt{n} \rangle$
`}`
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}+1-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i}+1 \leq \texttt{n} \rangle$
`i = i + 1;`
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i} \leq \texttt{n} \rangle$
`}`
$\langle \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i} \leq \texttt{n} \wedge \neg(\texttt{i} < \texttt{n}) \rangle$
$\langle \texttt{max} = max(\bigcup_{k=0}^{n-1}\{a[k]\} \cup \{0\}) \rangle$

Zusätzliche Implikation für If-Regel:
$\texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{i} < \texttt{n} \wedge \neg(\texttt{max} < \texttt{a}[\texttt{i}])$
$\implies \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}+1-1}\{a[k]\} \cup \{0\}) \wedge \texttt{i}+1 \leq \texttt{n}$
ist wahr, da
$\texttt{max} = max(\bigcup_{k=0}^{\texttt{i}-1}\{a[k]\} \cup \{0\}) \wedge \neg(\texttt{max} < \texttt{a}[\texttt{i}]) \implies \texttt{max} = max(\bigcup_{k=0}^{\texttt{i}+1-1}\{a[k]\} \cup \{0\})$
und
$\texttt{i} < \texttt{n} \implies \texttt{i}+1 \leq \texttt{n}$.

Terminierung:

Wir wählen als Variante $V = \texttt{n} - \texttt{i}$. Hiermit lässt sich die Terminierung von P beweisen, denn für die einzige Schleife im Programm (mit Schleifenbedingung $B = \texttt{i} < \texttt{n}$) gilt:

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow \texttt{i} < \texttt{n} \Leftrightarrow \texttt{n} - \texttt{i} > 0 \Leftrightarrow V > 0$ und
- $\langle \texttt{n} - \texttt{i} = m \wedge \texttt{i} < \texttt{n} \rangle$
  `if (max < a[i]){`
  $\langle \texttt{n} - \texttt{i} = m \wedge \texttt{i} < \texttt{n} \wedge \texttt{max} < \texttt{a}[\texttt{i}] \rangle$
  $\langle \texttt{n} - (\texttt{i}+1) < m \rangle$
  `max = a[i];`
  $\langle \texttt{n} - (\texttt{i}+1) < m \rangle$
  `}`
  $\langle \texttt{n} - (\texttt{i}+1) < m \rangle$

```
i = i + 1;
```
$\langle \mathtt{n} - \mathtt{i} < m \rangle$

Zusätzliche Implikation für If-Regel:
$\mathtt{n} - \mathtt{i} = m \wedge \mathtt{i} < \mathtt{n} \wedge \neg(\mathtt{max} < \mathtt{a[i]}) \implies \mathtt{n} - (\mathtt{i} + 1) < m$
ist wahr, da
$\mathtt{n} - \mathtt{i} = m \implies \mathtt{n} - (\mathtt{i} + 1) < m$.

Aufgabe H11 (3+5+3+2+2 Punkte)[1]

Gegeben ist der Rumpf einer zirkulär doppelt-verketteten Liste. In so einer Liste kann von jedem Element auf das vorherige und das nachfolgende Element der Liste zugegriffen werden. Die Elemente formen einen Kreis, die Liste hat also kein Anfang oder Ende.

1. Implementieren Sie die Methode **public int** *size*(), welche die Anzahl an Elementen in der Liste zurückgibt.
2. Implementieren Sie die Methode **public** *CircularDoubleLinkedList insert*(**int** *value*). Die Methode soll ein neues Element zur Liste hinzufügen und eine Referenz auf dieses Element zurückgeben.
3. Implementieren Sie die Methode **public boolean** *checkStructure*() um folgende Invariante der Liste zu prüfen: ein Listenelement hat ein Vorgängerelement *previous* und ein Nachfolgerelement *next*, beide dürfen nicht *null* sein. Weiterhin gilt für jedes Listenelement, daß der Vorgänger des Nachfolgers das Element selbst ist; genauso verhält es sich mit dem Nachfolger des Vorgängers. Die Methode *checkStructure*() soll diese Invariante für das Element, für welches sie aufgerufen wird, überprüfen und genau dann *true* zurückgeben, wenn diese Invariante gegeben ist.[2]

   Rufen sie *checkStructure*() in einem **assert** vor und nach dem Ausführen jeder **public** Methode für jedes geänderte Element sowie jedes Element, daß eine Referenz auf ein geändertes Element besitzt, auf. Prüfen Sie mit einem solchen **assert** auch, ob die Invariante am Ende jedes Konstruktors für das gerade erzeugte Element gilt.

   (a) Stellt diese Invariante sicher, daß die Liste immer kreisförmig ist?

   (b) Stellt diese Invariante sicher, daß die Liste, welche von *insert* zurückgegeben wird, auch alle Elemente der ursprünglichen Liste beinhaltet?

```
class CircularDoubleLinkedList {
  public CircularDoubleLinkedList previous, next;
  public int value;
  public CircularDoubleLinkedList(int value){
    this.value = value;
    this.previous = this;
    this.next = this;
    assert(this.checkStructure());
  }
  private CircularDoubleLinkedList(int value,
        CircularDoubleLinkedList previous, CircularDoubleLinkedList next){
    this.value = value;
    this.previous = previous;
```

```
    this.next = next;
  }
  public CircularDoubleLinkedList insert(int value) {//TODO }
  public int size() {//TODO }
  public boolean checkStructure() {//TODO }
}
```

Lösungsvorschlag

```
class CircularDoubleLinkedList {
  public CircularDoubleLinkedList previous, next;
  public int value;
  public CircularDoubleLinkedList(int value){
    this.value = value;
    this.previous = this;
    this.next = this;
    assert(this.checkStructure());
  }
  private CircularDoubleLinkedList(int value,
        CircularDoubleLinkedList previous, CircularDoubleLinkedList next){
    this.value = value;
    this.previous = previous;
    this.next = next;
  }
  public CircularDoubleLinkedList insert(int value) {
    CircularDoubleLinkedList oldNext = this.next;
    CircularDoubleLinkedList oldNextNext = this.next.next;
    CircularDoubleLinkedList newValue =
        new CircularDoubleLinkedList(value, this, this.next);
    this.next.previous = newValue;
    this.next = newValue;
    assert(this.previous.checkStructure());
    assert(this.checkStructure());
    assert(newValue.checkStructure());
    assert(oldNext.checkStructure());
    assert(oldNextNext.checkStructure());
    return newValue;
  }
  public int size() {
    return this.size(this);
  }
  private int size(CircularDoubleLinkedList first) {
    if (this.next != first) {
      return 1 + this.next.size(first);
    } else {
      return 1;
    }
  }
```

```
  public boolean checkStructure() {
    if (this.previous != null
          && this.next != null
          && this.previous.next != null
          && this.next.previous != null
          && this.previous.next == this
          && this.next.previous == this) {
      return true;
    } else {
      return false;
    }
  }
}
```

3a) Ja, wenn diese Invariante im Konstruktor und jedes Mal auf allen geänderten Elementen sowie auf allen Elementen mit einer Referenz auf ein geändertes Element aufgerufen wird, garantiert sie, dass die Liste kreisförmig ist.

3b) Die Invariante stellt dies nicht sicher. Beispielsweise könnte *insert*(**int** *value*) das Element nicht einfügen, sondern schlicht ein neues Element erzeugen, dessen Vor- und Nachfolger es selbst ist. Keines der obigen *asserts* würde daraufhin fehlschlagen. Nichtsdestotrotz vermeidet das Prüfen der Invariante viele mögliche Fehler.

[1] Bitte Quelltext für die Abgabe ausdrucken und zusätzlich per E-mail an den jeweiligen Tutor senden.
[2] Korrigiert: *checkStructure* wird nicht mehr im privaten Konstruktor aufgerufen.