RWTH Aachen **WS 2013/14**
Lehrgebiet Theoretische Informatik Übungsblatt 3
Frohn–Reidl–Rossmanith–Sánchez–Ströder 11.11.2013

Übung zur Vorlesung Programmierung

Aufgabe T6

Schreiben Sie ein Codesegment P, das eine Integer-Variable x auf folgende Art und Weise verändert:

- Wenn $x = 0$ gilt, dann bleibt x unverändert.
- Wenn $x < 0$ gilt, dann wird x dekrementiert.
- Wenn $x > 0$ gilt, dann wird x inkrementiert.

Verifizieren Sie anschließend mit Hilfe des Hoare-Kalküls, dass folgende Aussage gilt:
$\langle x = x' \rangle$
P
$\langle (x' = 0 \wedge x = x') \vee (x' < 0 \wedge x = x' - 1) \vee (x' > 0 \wedge x = x' + 1) \rangle$

Lösungsvorschlag

$\langle x = x' \rangle$
if $(x < 0)$ $x = x - 1$;
if $(x > 0)$ $x = x + 1$;
$\langle (x' = 0 \wedge x = x') \vee (x' < 0 \wedge x = x' - 1) \vee (x' > 0 \wedge x = x' + 1) \rangle$

(1) Bedingungsregel

$$\left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle \textbf{if}(x > 0) x = x + 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' + 1 \end{array} \right\rangle$$

(1.1) Vorbedingung 1

$$\left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right. \left. \begin{array}{l} \\ \wedge x > 0 \\ \\ \end{array} \right\rangle x = x + 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' + 1 \end{array} \right\rangle$$

(1.1.1) Konsequenzregel 1

$$\langle x = x' \wedge x > 0 \rangle \; x = x + 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' + 1 \end{array} \right\rangle$$

(1.1.2) Konsequenzregel 1

$$\langle x' > 0 \wedge x + 1 = x' + 1 \rangle \; x = x + 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' + 1 \end{array} \right\rangle$$

(1.1.3) Konsequenzregel 2

$\langle x' > 0 \wedge x + 1 = x' + 1 \rangle \; x = x + 1 \; \langle x' > 0 \wedge x = x' + 1 \rangle$

$\Longrightarrow$**gilt nach Zuweisungsregel!**

(1.2) Vorbedingung 2

$$\left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle \wedge \neg x > 0 \Longrightarrow \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' + 1 \end{array} \right\rangle$$

$\Longrightarrow$**gültige Implikation!**

(2) Bedingungsregel

$$\left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle \textbf{if}(x < 0) x = x - 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle$$

(2.1) Vorbedingung 1

$$\left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle \wedge x < 0 \; x = x - 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle$$

(2.1.1) Konsequenzregel 1

$$\langle x' < 0 \wedge x - 1 = x' - 1 \rangle \; x = x - 1 \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle$$

(2.1.2) Konsequenzregel 2

$\langle x' < 0 \wedge x - 1 = x' - 1 \rangle \; x = x - 1 \; \langle x' < 0 \wedge x = x' - 1 \rangle$
$\Longrightarrow$**gilt nach Zuweisungsregel!**

(2.2) Vorbedingung 2

$$\left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle \wedge \neg x < 0 \Longrightarrow \left\langle \begin{array}{ll} & x' = 0 \wedge x = x' \\ \vee & x' < 0 \wedge x = x' - 1 \\ \vee & x' > 0 \wedge x = x' \end{array} \right\rangle$$

$\Longrightarrow$**gültige Implikation!**

(3) Sequenzregel aus (1) und (2)

$\langle (x' = 0 \wedge x = x') \vee (x' < 0 \wedge x = x') \vee (x > 0 \wedge x = x') \rangle$
if $(x < 0)$ $x = x - 1$;
if $(x > 0)$ $x = x + 1$;
$\langle (x' = 0 \wedge x = x') \vee (x' < 0 \wedge x = x' - 1) \vee (x' > 0 \wedge x = x' + 1) \rangle$

(4) Konsequenzregel 1

$\langle x = x' \rangle$
if $(x < 0)$ $x = x - 1$;
if $(x > 0)$ $x = x + 1$;
$\langle (x' = 0 \wedge x = x') \vee (x' < 0 \wedge x = x' - 1) \vee (x' > 0 \wedge x = x' + 1) \rangle$

Aufgabe T7

Implementieren Sie eine Klasse *Set*, deren Instanzen Mengen natürlicher Zahlen darstellen. Die Klasse *Set* soll die Methoden *add*(**int** x), *contains*(**int** x) und *remove*(**int** x) zur

Verfügung stellen. Dabei sollen die Attribute jeder Instanz der Klasse *Set* unveränderbar sein, nachdem sie im Konstruktor gesetzt wurden. Die Menge wird dafür durch eine Liste von *Set* Objekten repräsentiert. Jedes *Set* Objekt enthält eine Operation und eine Referenz zu einem anderen *Set* Objekt (oder *null*, wenn es das letzte Element der Liste ist). Jede Operation soll entweder eine Einfüge– oder eine Löschoperation sein. Die Methoden *add*(**int** *x*) und *remove*(**int** *x*) geben ein neues *Set* Objekt zurück, welches der neue Kopf der Liste ist und eine Einfüge– oder Löschoperation repräsentiert. Die Methode *contains*(**int** *x*) entscheidet, ob eine Element in der Menge ist oder nicht, indem sie die Einfüge– und Löschoperationen, die bereits ausgeführt wurden, durchläuft.

Lösungsvorschlag

```
class Set {
  private int value;
  private boolean present;
  private Set child;
  public Set (int value, boolean present) {
    this(value, present, null);
  }
  private Set (int value, boolean present, Set child) {
    this.value = value;
    this.present = present
    this.child = child;
  }
  public Set add(int x) {
    return new Set(x, true, this);
  }
  public Set remove(int x) {
    return new Set(x, false, this);
  }
  public boolean contains(int x) {
    if (this.value == x && this.present) {
      return true;
    } else if (this.value == x &&!this.present) {
      return false;
    } else if (this.child == null) {
      return false;
    } else {
      return this.child.contains(x);
    }
  }
}
```

Aufgabe H6 (10 Punkte)
Implementieren Sie ein Codesegment P, das eine Integer-Variable x derart verändert, dass sie nach Ausführung von P den Betrag jener Zahl enthält, die vor Ausführung von P in x gespeichert war. Verifizieren Sie anschließend mithilfe des Hoare-Kalküls, dass folgende Aussage gilt:
$\langle x = x' \rangle\ P\ \langle x = |x'| \rangle$

Lösungsvorschlag
$\langle x = x' \rangle\ \textbf{if}(x < 0)\ x = -x;\ \langle x = |x'| \rangle$

(1) Bedingungsregel
$\langle x = x' \rangle\ \textbf{if}(x < 0)\ x = -x;\ \langle x = |x'| \rangle$

(1.1) Vorbedingung 1
$\langle x = x' \wedge x < 0 \rangle\ x = -x;\ \langle x = |x'| \rangle$

(1.1.1) Konsequenzregel 2
$\langle x = x' \wedge x < 0 \rangle\ x = -x;\ \langle (x' \geq 0 \wedge x = x') \vee (x' < 0 \wedge x = -x') \rangle$

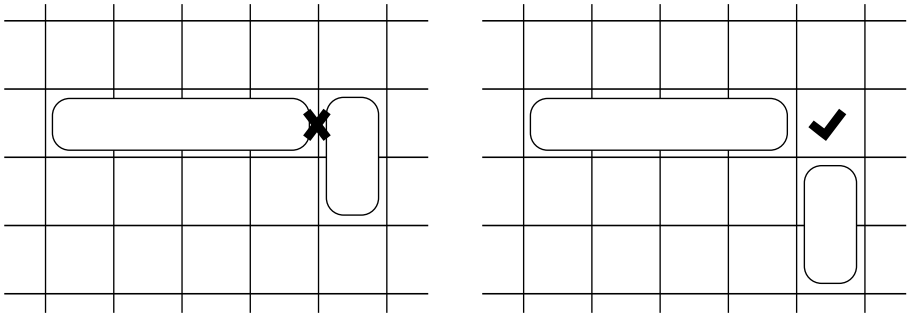
(1.1.2) Konsequenzregel 2
$\langle x = x' \wedge x < 0 \rangle\ x = -x;\ \langle x' < 0 \wedge x = -x' \rangle$

(1.1.3) Konsequenzregel 1
$\langle x' < 0 \wedge -x = -x' \rangle\ x = -x;\ \langle x' < 0 \wedge x = -x' \rangle$
$\Longrightarrow$ **Gilt nach Zuweisungsregel!**

(1.2) Vorbedingung 2
$\langle x = x' \wedge \neg x < 0 \rangle\ \Longrightarrow\ \langle x = |x'| \rangle$
$\Longrightarrow$ **Gültige Implikation!**

Aufgabe H7 (10 Punkte)[1]
Auf der Homepage zur Vorlesung finden Sie eine Archivdatei `battleship.zip`. Darin befindet sich eine fast vollständige Implementierung des Spiels "Schiffe versenken": lediglich in der Klasse *lufti.battleship.Ship* fehlen sinnvolle Implementierung zweier Methoden. Zum Starten des Spiels kompilieren sie die Klasse *main.BattleshipMain* und starten diese (starten Sie das Spiel ruhig einmal, um die fehlende Funktionalität zu sehen).
Implementieren Sie folgenden Methoden der Klasse *Ship*, um das Spiel zu vervollständigen:

- *isHitBy*(*Point shot*): Diese Methode soll *true* zurückgeben, wenn die Koordinate *shot* von diesem Schiff belegt wird. Es ist ratsam, dazu die Methode *getCoordinates*() zu Hilfe zu nehmen.
- *toCloseTo*(*Ship otherShip*): Diese Methode soll *true* zurückgeben, wenn dieses Schiff und das übergebene Schiff *otherShip* zu nahe beeinander liegen. Dies ist dann der Fall, wenn nicht mindestens ein Feld zwischen beiden Schiffen liegt; allerdings dürfen sich die Schiffe "über Eck" berühren:



Für die schriftliche Abgabe reicht es aus, wenn Sie die beiden obigen Methoden angeben. Senden Sie Ihrem Tutor bitte lediglich die von Ihnen modifizierte Klasse *Ship*.

Lösungsvorschlag

Die folgenden Implementation leisten das gewünschte:

```
public boolean toCloseTo(Ship otherShip) {
  int left = position.x;
  int right = position.x;
  int top = position.y;
  int bottom = position.y;
  if (horizontal) {
    right += size - 1;
  } else {
    bottom += size - 1;
  }
  Point[ ] coords = otherShip.getCoordinates();
  for (int i = 0; i < coords.length; i++) {
    Point point = coords[i];
    if ((point.x >= left - 1 && point.x <= right + 1
          && point.y >= top && point.y <= bottom)
        || (point.x >= left && point.x <= right
         && point.y >= top - 1 && point.y <= bottom + 1)) {
      return true;
    }
  }
  return false;
}

public boolean isHitBy(Point shot) {
  Point[ ] coords = getCoordinates();
  for (int i = 0; i < coords.length; i++) {
    Point point = coords[i];
    if (point.equals(shot)) {
      return true;
    }
  }
  return false;
```

```
}
```

Abgabe zum 19.11.2013

[1] Bitte Quelltext für die Abgabe ausdrucken und zusätzlich per E-mail an den jeweiligen Tutor senden.