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

WS 2013/14
Übungsblatt 4
18.11.2013

Übung zur Vorlesung Programmierung

Aufgabe T8

Ein *Stack* ist eine Datenstruktur, die stets nur Zugriff auf das zuletzt eingefügte Element ermöglicht. Folglich stellt ein Stack in der Regel Funktionen zum Ablegen eines Elementes (*push*) und zum Zugriff auf das zuletzt abgelegte Element (*pop*, *peek*) zur Verfügung.
Implementieren Sie eine Klasse `Stack`, die eine veränderbare[1] Implementierung der Datenstruktur Stack für nicht negative ganze Zahlen mit folgenden Methoden zur Verfügung stellt:

- **void** *push*(**int** x): Legt das Element x auf dem Stack ab. Wenn x negativ ist, dann bleibt der Stack unverändert.
- **int** *pop*(): Entfernt das zuletzt abgelegte Element vom Stack und liefert dieses zurück. Falls der Stack leer ist, wird -1 zurückgegeben.
- **int** *peek*(): Liefert das zuletzt abgelegte Element zurück, ohne es zu entfernen. Falls der Stack leer ist, wird -1 zurückgegeben.

Eine *Queue* ist eine Datenstruktur, die stets nur Zugriff auf jenes Element ermöglicht, das am längsten Teil der Queue ist. Folglich stellt eine Queue in der Regel Funktionen zum Einfügen eines neuen Elements (*enqueue*) und zum Zugriff auf das älteste Element (*dequeue*) zur Verfügung.
Implementieren Sie eine Klasse `Queue`, die eine veränderbare Implementierung der Datenstruktur Queue für nicht negative ganze Zahlen mit folgenden Methoden zur Verfügung stellt:

- **void** *enqueue*(**int** x): Fügt das Element x zu der Queue hinzu. Wenn x negativ ist, dann bleibt die Queue unverändert.
- **int** *dequeue*(): Entfernt das älteste Element aus der Queue und liefert dieses zurück. Falls die Queue leer ist, wird -1 zurückgegeben.

[1] Das heißt Methoden, die den Stack verändern, manipulieren die `Stack`-Instanz, auf der die Methode aufgerufen wurde, statt neue Instanzen zu erzeugen.

Lösungsvorschlag

```
class Stack {
    private Entry head;
    public boolean isEmpty() {
        return this.head == null;
    }
    public void push(int i) {
        if (i < 0) return;
        this.head = new Entry(i, this.head);
    }
    public int pop() {
        if (this.isEmpty()) return -1;
        int res = this.head.getValue();
        this.head = this.head.getNext();
        return res;
    }
    public int peek() {
        if (this.isEmpty()) return -1;
        return this.head.getValue();
    }
}
class Entry {
    private int value;
    private Entry next;
    Entry(int v, Entry n) {
        this.value = v;
        this.next = n;
    }
    int getValue() {
        return this.value;
    }
    Entry getNext() {
        return this.next;
    }
    void setNext(Entry n) {
        this.next = n;
    }
}
class Queue {
    private Entry head;
    private Entry last;
    public void enqueue(int i) {
        if (i < 0) return;
        Entry oldHead = this.head;
        this.head = new Entry(i, null);
        if (oldHead != null) oldHead.setNext(this.head);
        else this.last = this.head;
    }
```

```
    public boolean isEmpty() {
        return this.last == null;
    }
    public int dequeue() {
        if (this.isEmpty()) return -1;
        int res = this.last.getValue();
        this.last = this.last.getNext();
        if (this.last == null) this.head = null;
        return res;
    }
}
```

Aufgabe T9

Gegeben sei folgendes Java-Programm P:

$\langle\varphi\rangle$ (Vorbedingung)

```
r = 1;
while (r * r < x){
  r = r + 1;
}
```

$\langle\psi\rangle$ (Nachbedingung)

a) Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\mathtt{x} > 0$ und als Nachbedingung ψ gelte[2] $\mathtt{r} = \lceil\sqrt{\mathtt{x}}\,\rceil$. Vervollständigen Sie die folgende Verifikation des Algorithmus, indem Sie die leeren Zeilen durch korrekte Zusicherungen entsprechend des Hoare-Kalküls ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

$\langle \mathtt{x} > 0 \rangle$

$\langle$ _____ $\rangle$

```
r = 1;
```

$\langle$ _____ $\rangle$

$\langle$ _____ $\rangle$

```
while (r * r < x){
```

$\langle$ _____ $\rangle$

[2] hierbei ist $\lceil x \rceil$ die ganzzahlige Aufrundung von x; siehe auch http://de.wikipedia.org/wiki/Abrundungsfunktion_und_Aufrundungsfunktion

$\langle$ _____ $\rangle$

```
   r = r + 1;
```

$\langle$ _____ $\rangle$

```
}
```

$\langle$ _____ $\rangle$

$\langle \texttt{r} = \lceil \sqrt{\texttt{x}} \rceil \rangle$

Hinweise:

- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x + 1 = y + 1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und mit Hilfe des Hoare-Kalküls die Terminierung bewiesen werden.

Hinweis: Sie dürfen für den Terminierungsbeweis zusätzlich zur Schleifenbedingung auch $\texttt{r} > 0$ *zu Beginn jedes Schleifendurchlaufs annehmen.*

Lösungsvorschlag

a)

$\langle \texttt{x} > 0 \rangle$

$\langle \texttt{x} > 0 \wedge 1 = 1 \rangle$

```
r = 1;
```

$\langle \texttt{x} > 0 \wedge \texttt{r} = 1 \rangle$

$\langle (\texttt{r} - 1)^2 < \texttt{x} \wedge \texttt{r} > 0 \rangle$

```
while (r * r < x){
```

$\langle (\texttt{r} - 1)^2 < \texttt{x} \wedge \texttt{r} > 0 \wedge \texttt{r}^2 < \texttt{x} \rangle$

$\langle (\texttt{r} + 1 - 1)^2 < \texttt{x} \wedge \texttt{r} + 1 > 0 \rangle$

```
   r = r + 1;
```

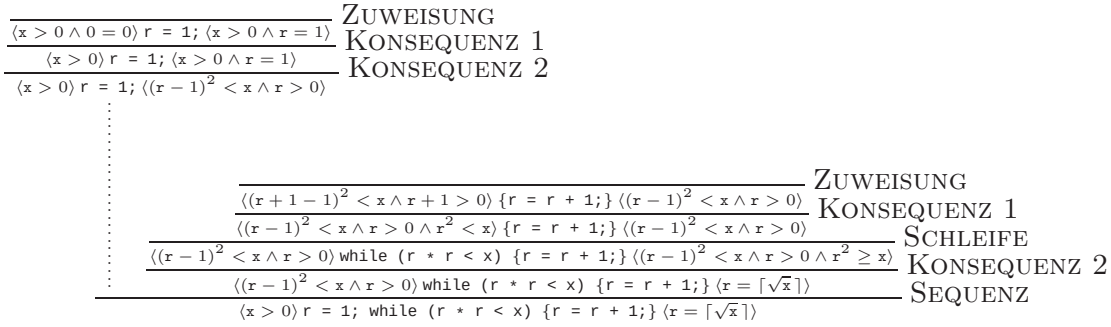
$\langle (\texttt{r} - 1)^2 < \texttt{x} \wedge \texttt{r} > 0 \rangle$

```
}
```

$\langle (\texttt{r} - 1)^2 < \texttt{x} \wedge \texttt{r} > 0 \wedge \neg(\texttt{r}^2 < \texttt{x}) \rangle$

$\langle \texttt{r} = \lceil \sqrt{\texttt{x}} \rceil \rangle$

Der folgende Teil dient lediglich zur Erklärung und ist für die Lösung der Aufgabe nicht erforderlich. Er zeigt genau, welche einzelnen Regeln des Kalküls in der obigen Lösung verwendet wurden, um die Korrektheit von P zu zeigen.



b) Wir wählen als Variante $V = \mathtt{x} - \mathtt{r}^2$. Hiermit lässt sich die Terminierung von P beweisen, denn für die einzige Schleife im Programm (mit Schleifenbedingung $B = \mathtt{r}^2 < \mathtt{x}$ und Schleifenkörper `r = r + 1;`) gilt:

- $B \Rightarrow V \geq 0$, denn $B \Leftrightarrow \mathtt{r}^2 < \mathtt{x} \Leftrightarrow \mathtt{x} - \mathtt{r}^2 > 0 \Leftrightarrow V > 0$ und
- $\langle \mathtt{x} - \mathtt{r}^2 = m \wedge \mathtt{r}^2 < \mathtt{x} \wedge \mathtt{r} > 0\rangle$
  $\langle \mathtt{x} - (\mathtt{r}+1)^2 < m\rangle$
  `r = r + 1;`
  $\langle \mathtt{x} - \mathtt{r}^2 < m\rangle$

Aufgabe H8 (8+2 Punkte)

Gegeben sei folgendes Java-Programm P:

$\langle\varphi\rangle$ (Vorbedingung)

```
i = 0;
res = 0;
while (i < n){
  i = i + 1;
  res = res + i * i * i;
}
```

$\langle\psi\rangle$ (Nachbedingung)

a) Als Vorbedingung φ für das oben aufgeführte Programm P gelte $\mathtt{n} \geq 0$ und als Nachbedingung ψ gelte $\mathtt{res} = (\frac{\mathtt{n}\cdot(\mathtt{n}+1)}{2})^2$. Vervollständigen Sie die folgende Verifikation des Algorithmus, indem Sie die leeren Zeilen durch korrekte Zusicherungen entsprechend des Hoare-Kalküls ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

$\langle \mathtt{n} \geq 0\rangle$

$\langle$_____$\rangle$

`i = 0;`

$\langle$_____$\rangle$

```
res = 0;
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

```
while (i < n){
```

$\langle$_____$\rangle$

$\langle$_____$\rangle$

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

$\langle$_____$\rangle$

```
   res = res + i * i * i;
```

$\langle$_____$\rangle$

```
}
```

$\langle$_____$\rangle$

$\langle \texttt{res} = (\frac{\texttt{n}\cdot(\texttt{n}+1)}{2})^2 \rangle$

Hinweise:

- Sie dürfen beliebig viele Zusicherungs-Zeilen ergänzen oder streichen. In der Musterlösung werden allerdings genau die angegebenen Zusicherungen benutzt.
- Bedenken Sie, dass die Regeln des Kalküls syntaktisch sind, weshalb Sie semantische Änderungen (beispielsweise von $x + 1 = y + 1$ zu $x = y$) nur unter Zuhilfenahme der Konsequenzregeln vornehmen dürfen.
- Es gilt $(\frac{x\cdot(x+1)}{2})^2 + (x + 1)^3 = (\frac{(x+1)\cdot(x+2)}{2})^2$ für alle $x \in \mathbb{R}$.

b) Untersuchen Sie den Algorithmus P auf seine Terminierung. Für einen Beweis der Terminierung muss eine Variante angegeben werden und mit Hilfe des Hoare-Kalküls die Terminierung bewiesen werden.

Lösungsvorschlag

a)
$\langle \texttt{n} \geq 0 \rangle$
$\langle \texttt{n} \geq 0 \wedge 0 = 0 \wedge 0 = 0 \rangle$
`i = 0;`
$\langle \texttt{n} \geq 0 \wedge \texttt{i} = 0 \wedge 0 = 0 \rangle$
`res = 0;`
$\langle \texttt{n} \geq 0 \wedge \texttt{i} = 0 \wedge \texttt{res} = 0 \rangle$
$\langle \texttt{res} = (\frac{\texttt{i}\cdot(\texttt{i}+1)}{2})^2 \wedge \texttt{i} \leq \texttt{n} \rangle$
`while (i < n){`
$\langle \texttt{res} = (\frac{\texttt{i}\cdot(\texttt{i}+1)}{2})^2 \wedge \texttt{i} \leq \texttt{n} \wedge \texttt{i} < \texttt{n} \rangle$
$\langle \texttt{res} + (\texttt{i}+1)\cdot(\texttt{i}+1)\cdot(\texttt{i}+1) = (\frac{(\texttt{i}+1)\cdot((\texttt{i}+1)+1)}{2})^2 \wedge (\texttt{i}+1) \leq \texttt{n} \rangle$
`i = i + 1;`
$\langle \texttt{res} + \texttt{i}\cdot\texttt{i}\cdot\texttt{i} = (\frac{\texttt{i}\cdot(\texttt{i}+1)}{2})^2 \wedge \texttt{i} \leq \texttt{n} \rangle$
`res = res + i * i * i;`
$\langle \texttt{res} = (\frac{\texttt{i}\cdot(\texttt{i}+1)}{2})^2 \wedge \texttt{i} \leq \texttt{n} \rangle$
`}`
$\langle \texttt{res} = (\frac{\texttt{i}\cdot(\texttt{i}+1)}{2})^2 \wedge \texttt{i} \leq \texttt{n} \wedge \neg(\texttt{i} < \texttt{n}) \rangle$
$\langle \texttt{res} = (\frac{\texttt{n}\cdot(\texttt{n}+1)}{2})^2 \rangle$

Korrekturhinweise:

- korrekte Anwendung der Zuweisungsregeln: jeweils 0,5 Punkte
- korrekte Anwendung der Schleifenregel (Schleifenbedingung bzw. ihre Negation kommt zu Beginn bzw. unmittelbar nach der Schleife hinzu und sonst ändert sich nichts): 1 Punkt
- korrekte Anwendung der Konsequenzregeln, sodass alle direkt untereinander stehenden Zusicherungen gefolgert werden können: insgesamt 2 Punkte
- unmittelbar vor und am Ende der Schleife steht die gleiche nicht-leere Zusicherung: 1 Punkt
- an einer der beiden gerade genannten Stellen steht eine Schleifeninvariante: 1 Punkt
- an einer der beiden gerade genannten Stellen steht eine Zusicherung, aus der gemeinsam mit der negierten Schleifenbedingung die Nachbedingung folgt: 1 Punkt
- die entsprechenden Punkte gibt es nur, wenn an der jeweiligen Stelle keine sonstigen Fehler gemacht werden (beliebt: semantische Änderung ohne Konsequenzregel)
- es gibt keinen Punktabzug für eigentlich verbotene kleinere Änderungen wie:
  - Tausch der Reihenfolge innerhalb einer Konjunktion
  - Umwandlung von $k \not\geq 0$ zu $k < 0$
  - Zusammenfassen von $k \geq 0 \wedge k > 0$ zu $k > 0$

  (die entsprechenden Fehler sollten aber kommentiert werden)
- wenn Fehler mehrfach gemacht werden, aber der eigentliche Lösungsweg richtig ist, wird nur beim ersten Fehler etwas abgezogen (Beispiel: semantische Änderung ohne Konsequenzregel)

b) 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
-
$\langle \texttt{n} - \texttt{i} = m \wedge \texttt{i} < \texttt{n} \rangle$
$\langle \texttt{n} - (\texttt{i} + 1) < m \rangle$
`i = i + 1;`
$\langle \texttt{n} - \texttt{i} < m \rangle$
`res = res + i * i * i;`
$\langle \texttt{n} - \texttt{i} < m \rangle$

Korrekturhinweise:

- Angabe einer sinnvollen Variante: 0,5 Punkte
- Begründung, warum es sich um eine gültige Variante ($V \geq 0$) handelt: 0,5 Punkte
- Beweis der Terminierung mittels des Hoare-Kalküls: 1 Punkt

Aufgabe H9 (5+5 Punkte)[3]

Erweitern sie die Klasses *Set* aus Aufgabe T7 indem sie zu der Klasse die Methoden *intersection*(*Set other*) und *union*(*Set other*) hinzufügen. Die Methode *intersection*(*Set other*) soll ein *Set* zurückgeben was nur die Elemente enthält, die sowohl in der Menge sind worauf die Methode aufgerufen wurde, als auch in der Menge *other*. Die Methode *union*(*Set other*) soll ein *Set* zurückgeben was alle Elemente aus der Menge worauf die Methode aufgerufen wuerde enthält, als auch die Element aus der Menge *other*. Dabei sollen die Attribute jeder Instanz der Klasse *Set*, wie davor, unveränderbar sein, nachdem sie im Konstruktor gesetzt wurden. Eine leere Menge kann dabei auch durch den Wert *null* representiert werden.

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);
    }
  }
  public Set intersection(Set other){
    return this.intersection(this, other);
  }
  public Set intersection(Set topElement, Set other){
    Set rest = null;
    if (this.child != null){
      rest = this.child.intersection(topElement, other);
    }
    if (topElement.contains(this.value)
        && other.contains(this.value)){
      return new Set(this.value, true, rest);
    } else {
      return rest;
    }
  }
  public Set union(Set other) {
    return union(this, other, other);
  }
  private Set union(Set left, Set right, Set topRight) {
    if (right == null) {
      return left;
    }
    Set res = left;
    if (right.child != null) {
      res = union(left, right.child, topRight);
    }
    if (topRight.contains(right.value)) {
      res.add(right.value);
    }
    return res;
  }
}
```

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