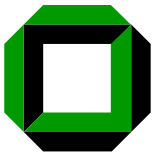

UNIVERSITÄT KARLSRUHE (TH)
Institut für Theoretische Informatik
Prof. Dr. B. Beckert
M. Ulbrich

Formale Systeme, WS 2008/2009

Lösungen zum Übungsblatt 10

Dieses Blatt wurde in der Übung am 23.1.2009 besprochen.

Zu Aufgabe 1

(a) Eine Lösung (zu einer Aufgabe) kann höchstens mit der Punktzahl der zugeordneten Aufgabe bewertet werden, wird aber nie negativ bewertet. *oder:* Die Bewertung einer Lösung ist größer oder gleich null, aber kleiner oder gleich der Punktzahl der Aufgabe.

(b) Die Zahl der Lösungen, die ein Student in einer Klausur erarbeitet, ist höchstens die Zahl der gestellten Aufgaben der Klausur.

(c) Alle Aufgaben, für die von Studenten Lösungen eingereicht werden, müssen Aufgaben sein, die in der Klausur gestellt werden.

Zu Aufgabe 2

(a)
```
context Klausur
inv L1: self.nachklausur->isEmpty() or
        self.aufgabe->intersect(self.nachklausur.aufgabe)->isEmpty()
inv L2: aufgabe->select(a | nachklausur->collect(aufgabe)->includes(a))->isEmpty()
inv L3: aufgabe->forAll(a | not nachklausur->collect(aufgabe)->includes(a))
```

Invariante (L1) bis (L3) stellen drei Lösungsmöglichkeiten dar, natürlich ist eine hinreichend.

Der Schnitt der Aufgaben einer Klausur mit den Aufgaben der Nachklausur muss leer sein.

Gibt es keine Nachklausur, so kann `nachklausur` als die leere Menge und `nachklausur.aufgabe` als Abkürzung für `self.nachklausur->collect(aufgabe)` ebenso leer interpretiert werden.

Aber gibt es keine Nachklausur, könnte `self.nachklausur` auch als undefiniert interpretiert werden (ist ja kein Element), womit dann `self.nachklausur.aufgabe` auch undefiniert ist.

Diese Problematik kann umgangangen werden, indem der Spezifikation hinzufügt wird, dass die Nachklausur auch nicht gesetzt kein.

`X->isEmpty()` ist eine Abkürzung für `X->size() = 0`.

(b)
```
context Klausur
inv:    self.nachklausur->size() + self.vorklausur->size() = 1
```

Die Möchtigkeiten dieser beiden Assoziationen muss in der Summe eins ergeben. Damit hat jede Klausur entweder eine nachfolgende oder eine vorangehende Klausur, aber nie beides.

In der Spezifikation der Aufgabenstellung steht „einer *anderen* Klausur“. Die Möglichkeit, dass eine Klausur etwa Nachklausur von sich selbst ist, ist aber auch von diesem Contraint ausgeschlossen, weil aus `self.nachklausur=self` schon `self.vorklausur=self` folgen würde, also die Summe 2 wäre.

(c)
```
context Student
inv:    self.bestanden implies loesung.bewertung->sum()
                                        >= klausur.mindestPunktzahl
```

Das **sum**-Konstrukt addiert die Bewertungspunkte für alle eingereichten Lösungen auf. Diese Summe wird verglichen mit der Mindestpunktzahl der Klausur.

Zu Aufgabe 3

Lösung:

```
post:   result = self->iterate(elem; acc:Set(T) = s | acc->including(elem))
```

Erläuterung: Benutze die Iterationsvariable `elem` und die Akkumulationsvariable `acc`. Der Ausdruck `acc->including(elem)` wird nun für jedes Element aus der Menge ausgewertet. Dabei enthält `elem` das aktuelle Element der Iteration und `acc` das Ergebnis des vorangegangenen Schrittes. Das Ergebnis der Auswertung wird wieder `acc` zugewiesen und steht im nächsten Iterationsschritt zur Verfügung. Der initiale Wert für `acc` wird explizit angegeben.

Hier ist der initiale Wert gerade die zweite Menge `s`, und die Akkumulationsvariable wird in jedem Durchlauf zu der (kleinsten) Menge verändert, die `acc` und das Laufelement `elem` enthält.

Am Schluss entsteht damit eine Menge, die alle Elemente aus `s` enthält (`acc` wird von `s` ausgehend nur erweitert) und auch die aus `self` enthält, darüber hinaus aber keine Element. Diese Menge soll das Ergebnis der Operation sein.

Zu Aufgabe 4

```
context Student::zurEinsichtGehen() : void
pre:    self.loesung->size() >= 1
post:   loesung->forAll(l | l.bewertung >= l.bewertung@pre)
```

Die Vorbedingung, damit dieser Vertrag angewendet werden kann, ist, dass ein Student mindestens eine Aufgabe bearbeitet hat.

Ist diese erfüllt, so gilt nach Ausführung der Methode "zurEinsichtGehen" für jede eingereichte Lösung, dass die Bewertung *nach* der Einsicht (`l.bewertung`) nicht kleiner (also größer oder gleich) der Bewertung *vor* der Einsicht (`l.bewertung@pre`) ist.