Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. B. Beckert
Thorsten Bormer

Formale Systeme, WS 2010/2011

Lösungen zu Übungsblatt 11

Dieses Blatt wurde in der Übung am 11.02.2011 besprochen.

Dieses Übungsblatt beschäftigt sich mit dem folgenden Klassendiagramm:

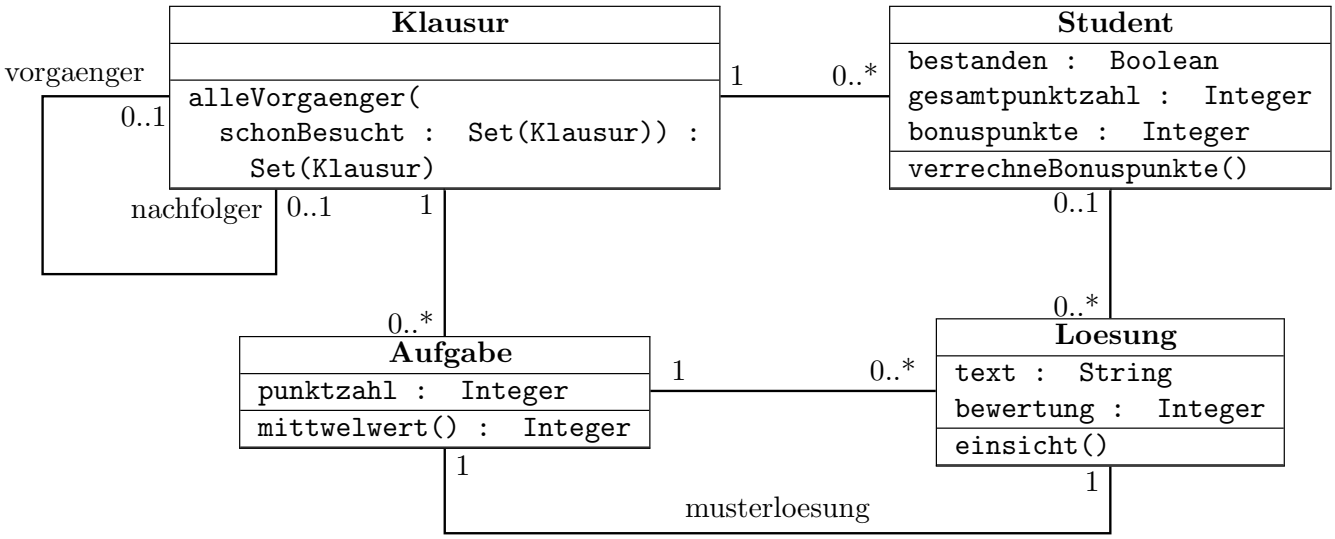

Abbildung 1: UML-Klassendiagramm (`nachfolger` und `vorgaenger` bezeichnen die einer Klausur direkt vorausgehende bzw. nachfolgende Klausur zur selben Vorlesung).

Zu Aufgabe 1

Geben Sie OCL-Bedingungen (Invarianten) an, die die folgenden Sachverhalte modellieren:

(a) Die Lösung jedes Studenten muss mindestens so lang sein wie die der Musterlösung, wenn der Student für diese Aufgabe die volle Punktzahl bekommen hat.

Lösung:

```
context Student
inv: loesung->forAll(l | l.bewertung = l.aufgabe.punktzahl
                            implies l.text.size()
                                >= l.aufgabe.musterloesung.text.size()
                        )
```

(b) In jeder Klausur gibt es (mindestens) einen Studenten, der bestanden hat, aber nicht zu jeder Aufgabe eine Lösung abgibt.

Folgende Invariante können Sie als gegeben annehmen:

```
context Student
inv: loesung->forAll(l1, l2 | l1.aufgabe = l2.aufgabe implies l1 = l2)
```

Lösung:

```
context Klausur
inv: student->exists(s | s.bestanden and aufgabe->size() > s.loesung->size())
```

(c) Unter allen Klausuren gibt es eine Klausur, bei der weniger als die Hälfte der teilnehmenden Studenten bestanden hat.

Lösung:

```
context Klausur
inv: Klausur.allInstances->exists(
            student->select(s | s.bestanden)->size() * 2 < student->size()
)
```

Zu Aufgabe 2

Spezifizieren Sie die folgenden Operationen mittels Vor-/Nachbedingungspaaren.

(a) `mittelwert()` berechnet die Durchschnittsbewertung für die Lösungen einer Aufgabe.

Lösung:

```
context Aufgabe::mittelwert : Integer
pre:    loesung->size()>0
post:   result = loesung.bewertung->sum() / loesung->size()
```

(b) Die Methode `verrechneBonuspunkte()` addiert die Bonuspunkte (≥ 0) eines Studenten zur Gesamtpunktzahl, aber nur, falls der Student die Klausur bestanden hat. Zusätzlich wird die Zahl der Bonuspunkte durch die Methode auf 0 gesetzt.

```
context Student::verrechneBonuspunkte() : void
pre:      bonuspunkte >= 0
post:     if bestanden then
            gesamtpunktzahl = gesamtpunktzahl@pre + bonuspunkte@pre
          else
            gesamtpunktzahl = gesamtpunktzahl@pre
          endif
        and
          bonuspunkte = 0
```

Zu Aufgabe 3

Die Spezifikation der Funktion `alleVorgaenger`, die den transitiven Abschluss bzgl. der `vorgaenger`-Relation berechnet, lautet wie folgt:

```
context Klausur::alleVorgaenger(s: Set(Klausur)) : Set(Klausur)
post: if s->includesAll(s.vorgaenger->asSet())
          then result = s
          else result = alleVorgaenger(s->union(s.vorgaenger->asSet()))
      endif
```

Diese rekursive Definition funktioniert (und funktioniert nur), weil per Definition in OCL nur endliche Objektmengen betrachtet werden.

Mit Hilfe dieser Funktion kann nun spezifiziert werden, dass die `vorgaenger`-Relation (und damit auch `nachfolger`) nicht zyklisch ist:

```
context Klausur
inv: alleVorgaenger(Set{self})->excludes(self)
```

Unglücklicherweise ist durch die Einschränkung auf terminierende Rekursion die syntaktische Korrektheit eines OCL Ausdrucks unentscheidbar.

Zu Aufgabe 4

`source->reject(iterator | body)` kann ausgedrückt werden als:

```
source->iterate(iterator; acc:Set(T)=source |
  if body then acc->excluding(iterator) else acc)
```

Eine Akkumulation ausgehend von der leeren Menge ist natürlich ebenso möglich.