Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. B. Beckert
Dr. Vladimir Klebanov

Formale Systeme, WS 2009/2010

Lösungen zum Übungsblatt 11

Dieses Blatt wurde in der Übung am 05.02.2010 besprochen.

Zu Aufgabe 1

(a)
```
context Klausur
inv L1: self.aufgabe->intersection(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))
```

Invarianten (L1) bis (L3) stellen drei verschiedene Lösungsmöglichkeiten dar.

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

(b)
```
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.

(c)
```
context Loesung
inv:    student.nachbar->collect( n | n.loesung )->
            exists( l | l.text = self.text )
        implies not student.bestanden
```

Wichtig ist hier, dass der Text der Lösungen verglichen wird und nicht die Lösungsobjekte.

Zu Aufgabe 2

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

(b)
```
context Loesung::einsicht() : void
post:   bewertung >= bewertung@pre
```

Zu Aufgabe 3

Wir spezifizieren hierzu eine Hilfsfunktion `abschlussVorgaenger`, die eine Menge von Klausuren als Argument hat, und den transitiven Abschluss bzgl. der `vorgaenger`-Relation berechnet:

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

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

Diese kann nun verwendet werden, um `alleVorgaenger` zu spezifizieren:

```
context Klausur::alleVorgaenger() : Set(Klausur)
post : result = abschlussVorgaenger(Set{self})
```

Schließlich kann man nun spezifizieren, dass die `vorgaenger`-Relation (und damit auch `nachfolger` nicht zyklisch ist:

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

Alternativ könnte man auch auf die Hilfsfunktion verzichten, wenn man die Signatur der Funktion `alleVorgaenger` veränderte und ihr eine Menge von Klausuren als Argument gäbe:

```
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

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.