Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Dr. Thorsten Bormer, Dr. Vladimir Klebanov, Dr. Mattias Ulbrich

Formale Systeme, WS 2015/2016

Lösungen zu Übungsblatt 11

Dieses Übungsblatt wurde in der Übung am 22.01.2016 besprochen.

Aufgabe 1

Es sei an die Java-Klasse `Person` zur Implementierung einer Personendatenbank vom letzten Übungsblatt erinnert. Ihre Deklaration (ohne die Methoden) lautete:

```
class Person {
   int age;
   boolean isFemale;
   Person father;
   Person mother;
   Person[] children;

   // Methoden: s. Blatt 10
}
```

(a) Formulieren Sie für die folgenden Aussagen Klassen-Invarianten in JML:

(i) *Der Vater einer Person ist männlich und älter als die Person selbst.*

(ii) *Alle Kinder einer Person sind echt jünger als die Person selbst.*

(iii) *Jede Person ist Kind ihrer Mutter.*

(iv) *Wenn eine Person weiblich ist, dann ist sie Mutter all ihrer Kinder.*

(b) Die JML-Semantik gestattet es nicht, dass eine Person ohne Vater/Mutter angelegt wird. Wie muss die Spezifikation geändert werden, wenn Personen ohne Angabe ihrer Eltern angelegt werden können sollen?

Lösung zu Aufgabe 1

(a) Die folgenden Klasseninvarianten können an Stellen in der Klassendefinition von `Person` stehen, an der Felder oder Methoden deklariert werden können.

Invarianten gelten für das durch `this` referenzierte Objekt. Die Aussage ist implizit allquantifiziert, da wir annehmen, dass Invarianten für *alle* Objekte gelten. Es genügt also für Aussagen die Eigenschaften für "alle Personen" fordern, diese für das durch `this` referenzierte Objekt zu formalisieren.

(i) `//@ invariant !father.isFemale && father.age > this.age;`

(ii) `//@ invariant (\forall int i; 0<=i && i<children.length; children[i].age<age);`

(iii)
```
/*@ invariant (\exists int i; 0<=i && i<mother.children.length;
@                mother.children[i] == this);
@*/
```

(iv)
```
/*@ invariant isFemale ==>
@     (\forall int i; 0<=i && i<children.length;
@          children[i].mother == this);
@*/
```

(b) JML hat eine Standardeinstellung: Wenn die Deklaration eines Feldes, eines Methodenparameters oder eines Rückgabewertes nicht weiter annotiert ist, so gilt er als `nonnull`: An der entsprechenden Stelle im Speicher darf der Wert `null` *nie* stehen.

Um diesen Standard zu deaktivieren, muss das Schlüsselwort `nullable` als Modifikator verwendet werden:

```
/*@ nullable */ Person father;
/*@ nullable */ Person mother;
```

Mit dieser modifizierten Deklaration, sollten nur auch die Spezifikationen angepasst werden und den weiteren Fall in Betracht ziehen. So sollte z.B. die Invariante aus (a.i) folgendermaßen lauten:

```
//@ invariant father!=null ==> (!father.isFemale && father.age > this.age);
```

Aufgabe 2

Gegeben sei die folgende Java-Klasse `C`, die die in JML spezifizierte Methode `m()` enthält:

```
class C {
  int[] a;
  int[] b;
  int[] c;

  /*@ public normal_behaviour
    @  requires a.length > 0 && b.length > 0;
    @  assignable a[*], b
    @*/
  void m() {
    a[0] = 1;             /* (1) */
    b[0] = 1;             /* (2) */

    // swapping a and b
    int[] x = a;          /* (3) */
    a = b;                /* (4) */
    b = x;                /* (5) */

    b[0] = 1;             /* (6) */
    c = new int[42];      /* (7) */
    c[0] = 1;             /* (8) */
  }
}
```

Welche der Zuweisungen (1)–(8) verstoßen gegen die `assignable`-Klausel des Methodenvertrages, welche nicht? Warum?

Lösung zu Aufgabe 2

(1) Die Zuweisung `a[0] = 1` ist erlaubt, da `a[*]` bedeutet, dass alle Stellen des Arrays `a`geschrieben werden dürfen. Außerdem ist die Länge von `a`wenigstens 1 wegen der Vorbedingung.

(2) Falls die beiden Referenzen `a` und `b` auf dasselbe Array zeigen, ist die Zuweisung `b[0] = 1` durch die `assignable`-Klausel `a[*]` gedeckt. Die Zuweisung verstößt gegen die Spezifikation, falls die beiden Referenzen `a`und `b`nicht auf dasselbe Array verweisen. Da diese Gleichheit (der Effekt wird auch *Aliasing* genannt) im Allgemeinen nicht der Fall ist, verstößt diese Zeile gegen den Vertrag. Die Klausel sagt aus, dass die Referenz `b` verändert werden darf (das Array darf ausgetauscht werden), der Inhalt von `b` ist davon aber nicht betroffen.

(3) Die Zuweisung `x = a` ist erlaubt, da der Methodenvertrag keine Einschränkung auf die (von außen nicht sichtbaren) lokalen Variablen hat.

(4) Die Zuweisung `a = b` ist nicht erlaubt. In der Spezifikation wird zwar `a[*]` als zuweisbar ausgewiesen, das bedeutet jedoch nur, dass der Inhalt des durch `a` referenzierten Arrays modifiziert werden darf. Das Feld `a` selbst darf nicht modifiziert werden.

(5) Die Zuweisung `b = x` wiederum ist erlaubt, da die Spezifikation eine Zuweisung auf das Feld `this.b` explizit erlaubt.

(6) Die Zuweisung `b[0] = 1` an dieser Stelle der Methode ist erlaubt. Gleichwohl war die gleichlautende Zuweisung in (2) verboten gewesen. Wie kann das sein? Dies hängt damit zusammen, dass die assignable-Klausel am Anfang einer Methode ausgeführt wird und dass Felder und Variablen ihre Werte ändern können. Aufgrund der Zuweisungen (3) und (5) zeigt `b` in dieser Zeile auf das Array, auf das `a` ursprünglich verwiesen hat. (Mittlerweile zeigt `a` auf ein anderes Array, aber das spielt keine Rolle.) Der Inhalt des ursprünglichen Arrays `a` darf aber nach Spezifikation modifiziert werden.

(7) Die Zuweisung `c = new ...` ist sicher nicht zulässig, da die Speicherstelle `this.c` nicht in der Klausel aufgelistet ist.

(8) Die letzte Zuweisung `c[0] = 1` ist wiederum zulässig, obwohl `c[*]` nicht in der Klausel gelistet ist. Das liegt daran, dass in (7) der Referenz `c` ein neues Objekt zugewiesen wurde. Der Inhalt von neuen Objekten darf (ungeachtet der assignable Klauseln) beliebig modifiiziert werden.