Fachhochschule Münster
FB Elektrotechnik und Informatik
Labor für Software Verifikation

Prof. Dr. Moritz Sinn
Julian Winter M.Sc.

WS 25
Aufgabenblatt 1

Praktikum
Zuverlässige Softwaresysteme

Allgemeiner Hinweis: In diesem Praktikum verwenden wir *Dafny*[1]. Um das Praktikum zu bestehen, müssen Sie mindestens 3 der 4 Aufgaben erfolgreich bearbeiten.

Vor dem Praktikumstermin I: Stellen Sie sicher, dass Sie eine funktionierende Installation von *Dafny* zur Verfügung haben: *Dafny* ist bequem zu installieren. Sie können auch die Laborrechner verwenden.
Vor dem Praktikumstermin II: Machen Sie sich mit der Programmiersprache *Dafny* und den Direktiven für die Verifikation vertraut: Eine Einführung in *Dafny* findet sich unter `https://dafny.org/latest/OnlineTutorial/guide`.

Aufgabe 1 MaxSum

Gegeben ist die Signatur einer Methode, welche die Summe s der beiden Parameter x und y sowie das Maximum m von x und y berechnen soll:

```
method MaxSum(x: int, y: int) returns (s: int, m: int)
```

(a) Spezifizieren Sie die Nachbedingung dieser Methode.

(b) Schreiben Sie eine Methode, welche `MaxSum` mit den Argumenten 4235 und 5 aufruft. Fügen Sie unter diesen Aufruf eine Zusicherung (assertion) ein, welche besagt, was das erwartete Resultat des Aufrufs ist. Falls die IDE diese Assertion nicht verifizieren kann, gehen Sie zurück zum ersten Aufgabenteil und verbessern Sie Ihre Spezifikation. Auf diese Art können Sie Ihre Spezifikation „testen“. Beachten Sie, dass Sie die Spezifikation mittels Verifizierung in der IDE „testen“, nicht durch Ausführung des Programms.

(c) Schreiben Sie eine Implementierung von `MaxSum`.

Aufgabe 2 MaxSum II

Die Methode

```
method ReconstructFromMaxSum(s: int, m: int) returns (x: int, y: int)
        ensures s = x + y
        ensures x ≤ m ∧ y ≤ m
```

[1] `https://dafny.org`

soll die Umkehrung der Methode `MaxSum` implementieren.

(a) Versuchen Sie eine Implementierung für die Methode `ReconstructFromMaxSum` zu schreiben. Sie werden feststellen, dass die IDE Ihre Implementierung für die angegebene Spezifikation nicht akzeptiert. Ergänzen Sie eine Vorbedingung, so dass die Spezifikation implementiert werden kann.

(b) Verwenden Sie den folgenden Testharness um Ihre Spezifikation zu prüfen:

```
method TestReconstructFromMaxSum(x: int, y: int) {
        var s, m := MaxSum(x, y);
        var xx, yy := ReconstructFromMaxSum(s, m);
        assert (xx = x ∧ yy = y) ∨ (xx = y ∧ yy = x);
}
```

Falls die angegebenen Assertion von der IDE nicht verifiziert werden kann, ergänzen Sie die Spezifikation der Methode `ReconstructFromMaxSum` entsprechend, um eine Verifizierung zu ermöglichen.

Aufgabe 3 Triple

Die Methode Triple wurde in der Vorlesung wie folgt spezifiziert

```
method Triple(x: int) returns (r: int)
ensures r = 3 * x
```

Eine Alternative Spezifikation der Methode ist wie folgt gegeben:

```
method Triple'(x : int) returns (r: int)
    ensures (r + 3*x) / 2 = 3 * x
```

(a) Die Spezifikation von $Triple'$ ist nicht äquivalent zur Spezifikation von $Triple$. Schreiben Sie eine Implementierung von $Triple'$, die keine Implementierung von $Triple$ ist.

(b) Fügen Sie eine minimale Nachbedingung zur Spezifikation von $Triple'$ hinzu, so dass die Spezifikation von $Triple'$ äquivalent ist zur Spezifikation von $Triple$.

(c) Wie können Sie überprüfen, dass die Spezifikationen von $Triple$ und die Spezifikation von $Triple'$ übereinstimmen?

Aufgabe 4 Zeichenkette finden

Es ist folgende Spezifikation einer Java Methode:

```
public static boolean contains(java.lang.String string,
                               java.langString substring)
```

Gibt `true` zurück genau dann wenn `string` die Zeichenkette `substring` enthält. Anderenfalls wird `false` zurückgegeben.

Parameter:

`string`	Die Zeichenkette in der gesucht wird
`substring`	Die Zeichenkette welche gesucht wird

Rückgabe

`true` falls `substring` in `string` enthalten ist
`false` anderenfalls

(a) Formulieren Sie sinnvolle Vorbedingungen für die genannte Methode in Form von assert-Statements in Java-Syntax.

(b) Formulieren Sie die Nachbedingung für die genannte Methode in Form von assert-Statements in Java-Syntax. Sie können hierfür auf die Methode `java.lang.String.contains` zurückgreifen.

(c) Betrachten Sie folgende Implementierung der Methode:

```
public static boolean contains(String string, String substring) {
        int i = 0;
        int j = 0;
        while (i < string.length() && j < substring.length()) {
                if (string.charAt(i) == substring.charAt(j))
                        j++;
                else {
                        j = 0;
                }
                i++;
        }
        return j == substring.length();
}
```

Ist diese Implementierung korrekt? Fügen Sie Ihre assert-Statements für Vor- und Nachbedingung an passender Stelle ein und Testen Sie die Implementierung. Ihre Testfälle müssen die Vorbedingung erfüllen. Verletzt einer der Testfälle die Nachbedingung?

(d) Verwenden Sie das Fuzzing-Tool Jazzer[2]. Hierzu müssen Sie die assert-Statements, welche Sie als Vorbedingung angeben haben, durch Code ersetzen, so dass ungültige Eingaben abgefangen werden und in solchen Fällen bspw. `false` zurück gegeben wird. Damit Sie das Gegenbeispiel, welches Jazzer findet, gut nachvollziehen können, sollten Sie die Zeichen auf den Bereich bspw. $A - Z$ beschränken. Auch dies können Sie erreichen, in dem Sie Eingaben, welche andere Zeichen enthalten wie beschrieben abfangen. Die assert-Statements, welche die Nachbedingung überprüfen, müssen Sie selbstverständlich im Code belassen. Falls Jazzer eine Eingabe findet, welche die Nachbedingung verletzt, wird Jazzer dies anzeigen und in einer Datei die entsprechende Eingabe hinterlegen.

(e) Analysieren Sie das gefundene Gegenbeispiel und beheben Sie den Fehler in der Implementierung. Prüfen Sie nach Behebung des Fehlers erneut mit dem Fuzzer Jazzer.

(f) Aus der Vorlesung Algorithmen und Datenstrukturen ist Ihnen möglicherweise noch der Knuth-Morris-Pratt-Algorithmus bekannt. Welchen Vorteil hat dieser gegenüber der gegebenen, von Ihnen korrigierten Implementierung? Kann die Imple-

[2] `https://github.com/CodeIntelligenceTesting/jazzer/releases`

mentierung durch den Knuth-Morris-Pratt-Algorithmus ersetzt werden, ohne die von Ihnen in Aufgabe a) und b) angegebene Spezifikation zu verletzen?