Fachhochschule Münster
FB Elektrotechnik und Informatik
Labor für Software Verifikation

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

WS 25/26
Aufgabenblatt 3

Praktikum
Zuverlässige Softwaresysteme

Allgemeiner Hinweis: In diesem Praktikum verwenden wir *Dafny*[1]. Um das Praktikum zu bestehen, müssen Sie mindestens 4 der 5 Aufgaben erfolgreich bearbeiten.

Vor dem Praktikumstermin: Gehen Sie die in der Vorlesung behandelten Beispiele durch (siehe Buch *Program Proofs*, Kapitel 5, die Beispiele finden sich als Code im Ilias) und stellen Sie sicher, dass Sie verstehen, wie die Beweise für diese Beispiele funktionieren

Aufgabe 1 Untere Schranke für `Reduce`

In der Vorlesung wurde die Funktion

```
function Reduce(m: nat, x: int): int {
    if m = 0 then x else Reduce(m/2, x+1) - m
}
```

vorgestellt und gezeigt, dass

$$x - 2m \leq \texttt{Reduce}(m, x)$$

gilt. Gehen Sie den Beweis noch einmal durch.
Warum kann

$$x - 2m < \texttt{Reduce}(m, x)$$

nicht gezeigt werden?

Aufgabe 2 `ReverseColors` ist eine Involution

Gegeben ist die in der Vorlesung vorgestellte Datenstruktur `BlueYellowTree`.

```
datatype BYTree = BlueLeaf | YellowLeaf | Node(left: BYTree, right: BYTree)
```

(a) Implementieren Sie eine Funktion

```
function ReverseColors(t: BYTree) : BYTree
```

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

welche die Farben Blau und Gelb im Baum vertauscht, d.h. jeden blauen Knoten in einen gelben Konten verwandelt und umgekehrt.

(b) „Testen“ Sie Ihre Funktion mit der folgenden Methode:

```
method TestReverseColors() {
 var a := Node(BlueLeaf, Node(BlueLeaf, YellowLeaf));
 var b := Node(YellowLeaf, Node(YellowLeaf, BlueLeaf));
 assert ReverseColors(a) = b;
 assert ReverseColors(b) = a;
}
```

Der Verifier sollte die Assertions direkt nachweisen können. Falls dies nicht der Fall ist, korrigieren Sie Ihre Implementierung.

(c) Weisen Sie nach, dass die Funktion `ReverseColors` ihre eigene Inverse ist, d.h. dass `ReverseColors(ReverseColors(`t`))` = t. Sie können sich dafür an den in der Vorlesung gegebenen Beweisen orientieren. Führen Sie den Beweis **ohne** automatische Induktion, d.h. füllen Sie den Body des folgenden Lemmas aus:

```
lemma{:induction false} ReverseColorsIsAnInvolution(t: BYTree)
    ensures ReverseColors(ReverseColors(t)) = t
{
//...
}
```

(d) Geben Sie auch einen Beweis unter Verwendung von *calc*-Statements an. Gehen Sie kleinschrittig vor um Schritt für Schritt die o.g. Gleichung (Nachbedingung des Lemmas) zu zeigen.

Aufgabe 3 `Oceanize` kann die Anzahl der blauen Knoten nur erhöhen

Gegeben ist die in der Vorlesung vorgestellte Datenstruktur `BlueYellowTree`.

```
datatype BYTree = BlueLeaf | YellowLeaf | Node(left: BYTree, right: BYTree)
```

(a) Implementieren Sie eine Funktion

```
function Oceanize(t: BYTree) : BYTree
```

welche alle Knoten Blau färbt.

(b) Weisen Sie nach, dass die Funktion `Oceanize` die Anzahl der blauen Knoten nur erhöhen kann, d.h. dass `blueCount(Oceanize(`t`))` >= `blueCount(`t`)` gilt. Führen Sie den Beweis **ohne** automatische Induktion, d.h. füllen Sie den Body des folgenden Lemmas aus:

```
lemma{:induction false} OceanizeCanOnlyIncreaseBlueCount(t: BYTree)
    ensures BlueCount(Oceanize(t)) ≥ BlueCount(t)
{
//...
}
```

(c) Geben Sie auch einen Beweis unter Verwendung von *calc*-Statements an. Gehen Sie kleinschrittig vor um Schritt für Schritt die o.g. Gleichung (Nachbedingung des Lemmas) zu zeigen.

Aufgabe 4 EvalEnvDefault

In *Ilias* finden Sie das Lemma *EvalEnvDefault*. Beweisen Sie dieses in *Dafny*. Hierfür ist es wichtig zunächst die angegebenen Definitionen und Funktionen zu verstehen um ein intuitives Verständnis für die Korrektheit der Aussage des Lemmas zu gewinnen.

Aufgabe 5 PreEvalCorrect

In *Ilias* finden Sie das Lemma *PreEvalCorrect* und einen lückenhaften Beweis dieses Lemmas. Füllen Sie die Lücken aus um einen vollständigen Beweis zu erhalten. Hierfür ist es wichtig zunächst die angegebenen Definitionen und Funktionen zu verstehen um ein intuitives Verständnis für die Korrektheit der Aussage des Lemmas zu gewinnen. Verwenden Sie *Dafny*.