Lehrstuhl für Software and Computational Systems
Prof. Dr. Gidon Ernst, Marian Lingsch-Rosenfeld

Übungen zur Vorlesung
Formale Spezifikation und Verifikation

Wintersemester 2024/25

Übungsblatt 06

Bekanntgabe am 02.12.2024

Für diese Aufgabe werden Aufgaben 1 und 2 in den Tutorien vorgerechnet.

Aufgabe 3 ist zur Bearbeitung in Präsenz in den Tutorien vorgesehen.

1 Array-Vergleich `arraycompare`

Am Anfang der Datei `blatt06_arraycmp.dfy` finden Sie die Methode `arraycopy`, welche wir bereits in der Vorlesung besprochen haben, mit einer vollständigen Spezifikation und Verifikation.

Implementieren Sie analog dazu die Methode `arraycompare`, welche zwei Arrays in einem Bereich vergleichen soll.

Aufgaben:

- Implementieren Sie die Methode `arraycompare`
- Fügen Sie geeignete Vor- und Nachbedingungen hinzu
- Fügen Sie notwendige Invarianten hinzu, sodass der Beweis erfolgreich ist

Hinweise

- Arbeiten Sie nur an einer Stelle gleichzeitig—wenn die Verifikation für eine Methode gerade fehlschlägt dann kann das die Verifikation von anderen Methoden gegebenfalls beeinträchtigen (z.B. dass Dafny festhängt).
- Wenn Dafny festhängt, dann können Sie es neu starten: `CTRL+SHIFT+P` bzw. `STRG+⇑+P` und dann "Dafny: Restart Server" eintippen und bestätigen.
- Nutzen Sie `assume` um sich in der Verifikation temporär auf bestimmte Fälle zu konzentrieren und `assert` um an Programmstellen zu testen, welche Eigenschaften gelten bzw. bewiesen werden können.
- Versuchen Sie, alle `assume`s für die Abgabe wieder zu entfernen, nur dann ist die Verifikation vollständig.
- Es könnte sein das sie die Invariante leicht modifiziert als `assert` vor einem `return` hinzufügen.

Installation von Dafny

- Website `https://dafny.org`
- DotNet Runtime `https://dotnet.microsoft.com/`
- VS Code `https://code.visualstudio.com/` (oder Codium)

- Dafny Plugin: Installation via Marketplace direkt in VS Code
  `https://marketplace.visualstudio.com/items?itemName=correctnessLab.dafny-vscode`
- Dafny auf GitHub `https://github.com/dafny-lang/dafny`
  → schöne Tutorial Videos und mehr Beispiele

2 Objektorientierte Programme: Bankkonto

Gegeben ist eine abstrakte Klasse für Bankkonten:

```
1 abstract class Account {
2     public Account();
3     public int getBalance();
4     public void deposit(int amount);
5     public void withdraw(int amount);
6 }
```

Nachfolgend die Beschreibung der Methoden:

- Der Konstruktor `Account()` für die Klasse ist so initialisieren, dass der Kontostand 0 ist
- Die Methode `getBalance()` gibt den aktuellen Kontostand zurück
- Mit der Methode `deposit` kann die durch `amount` angegebene Geldmenge eingezahlt werden. Das Ergebnis von `getBalance()` ändert sich entsprechend
- Mit der Methode `withdraw` kann analog die durch `amount` angegebene Geldmenge abgehoben werden

Aufgaben

a) Überlegen Sie sich, wie der interne Zustand der Klasse möglichst abstrakt und einfach repräsentiert werden kann (Anm.: die offensichtliche Lösung ist die richtige). Geben Sie dazu die Modellvariable(n) und deren Typ an

b) Formulieren Sie Kontrakte für den Konstruktor und alle Methoden, die die jeweilige Änderung des internen Zustands mit der/den Modellvariable(n) nach außen hin abstrakt beschreiben

c) Schreiben sie die von Ihnen erstellten Kontrakte in Dafny auf, bedienen sie sich dazu von der Datei `blatt06_account.dfy`

3 Präsenzaufgabe: Girokonto (Bankkonto mit separat gelisteter Einzahlung/Auszahlung)

Gegeben ist die Klasse `CheckingAccount` als Unterklasse von `Account`. Diese verwaltet mittels zwei Variablen separat die Geldmenge, welche seit dem Anlegen des Kontos eingezahlt wurde sowie jene Geldmenge, die insgesamt abgehoben wurde. Die Klasse `CheckingAccount` erlaubt darüber hinaus auch negative Kontostände (im Gegensatz zu `SavingsAccount` aus der vorigen Teilaufgabe).

Aufgaben

a) Implementieren Sie (auf Papier) die Methoden in folgender Klassendefinition, ohne zusätzliche Attribute hinzuzufügen:

```
1  class CheckingAccount extends Account {
2      // Attribute
3      int deposited;
4      int withdrawn;
5
6      // Konstruktor
7      public CheckingAccount()         { ... }
8
9      // Methoden
10     public int getBalance()          { ... }
11     public void deposit(int amount)  { ... }
12     public void withdraw(int amount) { ... }
13 }
```

b) Definieren Sie eine korrekte Klasseninvariante, welche die Attribute `deposited` und `withdrawn` mit der/den Modellvariable(n) in Bezug setzt

c) Verifizieren Sie (auf Papier) für den Konstruktor und alle Methoden dass

- Die Klasseninvariante nach den Aufrufen (wieder) etabliert ist.
- Dass die von `Account` geerbten Nachbedingungen gelten
- Geben Sie auch an, welchen neuen Wert die Modellvariable(n) am Ende der Methoden bekommen.