Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Dr. Mattias Ulbrich, Mihai Herda, Alexander Weigl

Formale Systeme, WS 2018/2019

Praxisaufgabe 3: Theorembeweiser

Abgabe der Lösung bis zum 24.02.2019 über die ILIAS-Seite zur Vorlesung
`https://ilias.studium.kit.edu/goto.php?target=crs_875048`

Hinweis: Bitte beachten Sie, dass die Praxisaufgabe in Einzelarbeit und selbständig zu bearbeiten ist. Wir behalten uns vor, die selbständige Bearbeitung dadurch stichprobenartig zu prüfen, dass wir uns die eingereichte Lösung erklären lassen.

Für die vollständige Lösung dieser Praxisaufgabe erhalten Sie **10 Übungspunkte**. Bitte beachten Sie die Erläuterung zu Übungspunkten auf der Webseite zur Vorlesung. Für unvollständige Lösung werden die Übungspunkte anteilig vergeben.

A Informationen zum KeY-System

Was ist KeY? Zusammen mit unseren Partnern, unter anderem an der Technische Universität Darmstadt, wird an unserem Institut das KeY-System entwickelt. Es ist ein Softwareverifikationswerkzeug, mit dem die Übereinstimmung von Java-Software und ihrer Spezifikation formal bewiesen werden kann.

Literatur zu KeY Ein Buch [ABB+16] über das KeY-System steht zur Verfügung. Das Buch ist über Springerlink[1] im Uni-Netz verfügbar. Besonders empfehlenswert für diese Praxisaufgabe sind Kapitel 15 „Using the KeY Prover“ und 16 „Formal Verification with KeY: A Tutorial“.

Installation Das KeY-System besitzt eine graphische Benutzeroberfläche und ist komplett in Java geschrieben, so dass es auf jeder Plattform, für die eine virtuelle Maschine für Java zur Verfügung steht, lauffähig ist. Laden Sie KeY 2.6, die Binary Version[2] , von der Webseite des Tools[3] herunter. Nachdem Sie die zip Datei entpacken, können Sie KeY mit dem Kommando `java -jar KeY.jar` starten.

Für die Aufgaben dieses Blattes empfiehlt es sich, die Standardeinstellungen des KeY-Systemes zu belassen, wie sie zum Zeitpunkt des Systemstarts sind.

[1] http://link.springer.com/book/10.1007%2F978-3-319-49812-6
[2] Alternativ können Sie den Quellcode herunterladen und compilieren. Anweisungen dazu gibt es in den Readme-Dateien.
[3] http://key-project.org/download/

B Aufgabe: Spezifikation und Verifikation eines Java-Programms

Gegeben sei folgende Implementierung von Selectionsort[4]:

```
public class SelectionSort{

   public void sort(int[] a){
      for(int i = 0; i < a.length; i++){
         int m = min(a,i);
         swap(a,m,i);
      }
   }

   public void swap(int[] a, int i, int j){
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }

   public int min(int[] a, int start){
      int res = start;
      for(int i = start; i < a.length; i++){
         if(a[i] < a[res]){
            res = i;
         }
      }
      return res;
   }
}
```

[4] https://de.wikipedia.org/wiki/SelectionSort

Aufgabe 1 Die Methode `sort` sortiert das Array `a`, d. h., jedes Element von `a` im Nachzustand ist kleiner oder gleich seinem rechtem Nachbarn (falls ein solcher Nachbar existiert). Die Methode `swap` tauscht die Elemente von der Positionen `i` und `j` im Array `a`. Die Methode `min` liefert die Position des Minimums der Elementen aus Array `a` ab Position `start`.

Spezifizieren Sie das Verhalten der Methoden `sort`, `swap` und `min`, indem Sie die vorgegebenen Methodenverträge vervollständigen.

Beweisen Sie die Korrektheit der Implementierung gegenüber den Methodenverträgen. Vervollständigen Sie dazu die Schleifeninvarianten und -varianten (`decreases`-Klauseln).

Beachten Sie:

- Es genügt zu zeigen, dass das Array im Nachzustand sortiert ist. Sie müssen **nicht** zeigen, dass das Array zudem auch eine Permutation des Arrays im Vorzustand ist.
- Für den Beweis der Methode `sort` ist der richtige Methodenvertrag für die Methoden `swap` und `min` notwendig – aber auch hinreichend. Sie müssen **nicht** zeigen, dass `swap` und `min` einen stärkeren Vertrag erfüllen als der, der zum Beweis der Korrektheit von `sort` erfolderlich ist.

Abgabe Bitte laden Sie die von Ihnen vervollständigte Datei `SelectionSort.java` mit dem annotierten Programm mittels der dafür vorgesehenen Formularfelder in ILIAS hoch.

Lassen Sie die vorgegebene Implementierung unverändert!

Geben Sie Ihre Matrikelnummer als Kommentar an!

Weitere Hinweise Sollte die Implementierung und Ihre Spezifikation konsistent sein, und die notwendigen Zusatzannotationen hinreichend, so funktionieren die Beweise der Methoden `sort` und `shiftRight` automatisch (z.B. durch Betätigen des grünen „Play“-Knopfes).

Erfahrungsgemäß klappt das Beweisen aber nicht auf Anhieb, da die Spezifikation und die Zusatzannotationen Fehler enthalten können. Zum „Debuggen“ empfiehlt es sich, den „Beweis-Autopiloten“ zu nutzen. Wählen Sie dafür nach Laden der Beweisverpflichtung den Punkt „Full Auto Pilot“ aus dem Kontextmenü „Strategy Macros“, das in der Sequenzen- oder der Beweisbaumansicht verfügbar ist. Der Autopilot strukturiert den Beweis stärker anhand der zu beweisenden Aussage: jeder Einzelteil der zu beweisenden Aussage wird als separates Beweisziel behandelt. An den offenen Zielen kann man in diesem Fall besser erkennen, welcher Teil der Korrektheitsaussage nicht (automatisch) bewiesen werden konnte.

KeY wird mit einigen Beispielen ausgeliefert, diese können unter „Load Example“ im File-Menü aufgerufen werden. Als Einstieg sind die Beispiele „Sum and Max“, „Binary Search“, und „Remove Duplicates“ gut geeignet.

Literatur

[ABB+16] Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Mattias Ulbrich, editors. *Deductive Software Verification - The KeY Book: From Theory to Practice*, volume 10001 of *Lecture Notes in Computer Science*. Springer, 2016.