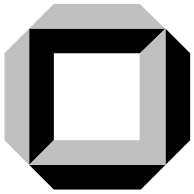

Universität Karlsruhe
Institut für Theoretische Informatik
Prof. Dr. P. H. Schmitt
C. Engel, B. Weiß

10. Dezember 2008

Praktikum
Formale Entwicklung objektorientierter Software
Übungsblatt 7

Aufgabe 15

Laden Sie sich das Archiv `keyex2.tgz` von der Praktikums-Webseite herunter, und entpacken Sie es mit dem Befehl:

```
tar xzvf keyex2.tgz
```

Wie schon in Aufgabe 13 auf Blatt 6 erhalten Sie sieben Dateien mit Namen `p1.key` bis `p7.key`, von denen jede eine Beweisaufgabe enthält, die Sie mit dem KeY-Beweiser lösen sollen. Verfahren Sie zum Laden, Beweisen und Abgeben der Probleme wie in Aufgabe 13 beschrieben. Benutzen Sie insbesondere die automatische Beweissuche (Strategien) nur dann, wenn es in den untenstehenden Erläuterungen explizit angegeben ist.

Stellen Sie vor dem Beweisen sicher, dass unter Options — Default Taclet Options — intRules die Einstellung intRules:arithmeticSemanticsIgnoringOF ausgewählt ist. Mit dieser Einstellung ignoriert der KeY-Beweiser die Möglichkeit von Integer-Überläufen, was uns hier im Praktikum das Arbeiten erleichtern wird.

`p1.key` Hier soll gezeigt werden, dass der Wert einer Integer-Variablen `i` nach Ausführung der Anweisung `i++` um eins höher ist als vorher.

Dazu arbeitet man schrittweise die Programmformeln ab ("symbolische Ausführung"). Befindet sich der Mauszeiger über einer Programmformel, wird die jeweils nächste zu behandelnde Anweisung grau unterlegt. Die Regel zur symbolischen Ausführung dieser Anweisung ist normalerweise die oberste aus der Liste der anwendbaren Regeln, etwa `postincrement`, `cast`, `variableDeclaration` etc.

Nach Abarbeitung des Programms bleibt unter anderem ein Term mit dem Funktionssymbol `javaAddInt` übrig. Dieses Funktionssymbol kann (aufgrund der ausgewählten Behandlung von Integerüberläufen) mit Hilfe der Regel `translateJavaAdd` in eine mathematische Addition auf $\mathbb{Z}$ umgewandelt werden.

`p2.key` Hier sollen Sie zeigen, dass nach Ausführung des Programmstücks mit den lokalen Variablen `i` und `k`:

```
if (k==0) {
  i=k;
}
else {
  i=0;
}
```

`i` den Wert 0 hat. Führen Sie wieder das Programm symbolisch aus. Sie werden Fallunterscheidungen machen, die Sie beim "intuitiven Überprüfen" des Programms auch machen würden (z.B. ob `k` den Wert 0 hat oder nicht). Denken Sie daran, dass eine Sequenz und eine Implikation auch dann wahr werden, wenn Sie deren linke Seite zu *false* vereinfachen können.

`p3.key` Bevor Sie die Problemdatei in den KeY-Beweiser laden, öffnen Sie die Datei in einen Editor Ihrer Wahl und betrachten Sie die DL-Formel, die innerhalb von `problem{ ... }` steht. Sie enthält folgendes Java-Programm in einem Diamond:

```
int i=0;
try {
   throw e;
   i=i+1;
} catch (RuntimeException e1) {
   i=i+4;
} finally {
   i=i+8;
}
```

Hierbei ist `e` ein Objekt (nicht mit `null` belegt) vom Typ `IllegalArgumentException`. `IllegalArgumentException` ist eine Unterklasse von `RuntimeException`. Ersetzen Sie in der hinter dem Diamond stehenden Teilformel `i = XXX` das `XXX` durch eine solche Zahl, dass die DL-Formel allgemeingültig ist. Falls Ihnen die Java-Semantik bezüglich der Behandlung von Exceptions nicht mehr geläufig ist, schlagen Sie sie in der Java Language Specification[1] nach.

Laden Sie dann die so veränderte Problemdatei in den KeY-Beweiser und weisen Sie die Allgemeingültigkeit nach.

(a) Machen Sie zu Beginn jede Regelanwendung ohne Zuhilfename der Strategien, bis einschließlich der Anwendung von `tryCatchFinallyThrow`. Beschreiben Sie den Effekt dieser Regel.

(b) Führen Sie den Rest des Beweises, indem Sie die Strategie `Java DL` einschalten und auf den Knopf ▷ klicken. Der Beweis läuft dann automatisch ab. Im Proof-Tab können Sie nachvollziehen, welche Regeln angewandt wurden.

`p4.key` Zu dieser Aufgabe gehört die Java-Klasse `MyClass.java`, die Sie im Unterverzeichnis `classes` finden. Laden Sie erneut die .key-Datei in einen Editor und überlegen Sie sich, durch was Sie `XXX` ersetzen müssen, damit die Eingabeformel allgemeingültig ist. Natürlich sollte nicht schon die Teilformel hinter dem Diamond allein eine Tautologie sein. Beweisen Sie die Allgemeingültigkeit der veränderten Formel mit KeY.

`p5.key` Nun sollen Sie sich um das Ergebnis des folgenden Programmstücks kümmern:

```
i=i0;
if (((j=i+=i++)+ ++i)== i=++i)
  i=i++;
```

Überlegen Sie sich, wie die Ausdrücke ausgewertet werden. Ersetzen Sie danach die beiden `XXX` durch die richtigen Ausdrücke, die nur aus `+`, `i0` und Zahlenliteralen bestehen dürfen. Beweisen Sie die veränderte Eingabedatei. Sie können dazu die `Java DL`-Strategie verwenden.

[1] `http://java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html`

`p6.key` Die folgende Regel (`loopUnwind`)

$$\frac{\Gamma \Longrightarrow \langle \texttt{if (b) \{l1 : \{l2 : \{}\alpha'\texttt{\}} while (b)\{}\alpha\texttt{\}\}\}}\rangle\Phi,\ \Delta}{\Gamma \Longrightarrow \langle \texttt{while (b)\{}\alpha\texttt{\}}\rangle\Phi,\ \Delta}$$

führt genau einen Schleifendurchlauf aus. Dabei stimmt α' mit α überein, außer dass jedes in α vorkommende `break;` durch ein `break l1;` und jedes `continue;` durch ein `break l2;` ersetzt ist.

(a) Laden Sie das Problem in Ihren Editor. Im Abschnitt `problem` finden Sie ein Java-Programm. Überlegen Sie sich wieder den Wert der Variablen `i` nach Ausführung des Programms und tragen Sie ihn anstelle von `XXX` ein.

(b) Laden Sie Ihre modifizierte Beweisereingabe `p6.key` in den Beweiser. Stellen Sie die Anzahl der maximalen automatischen Regelanwendungen im Proof Search Strategy-Tab auf mindestens 100 und wählen Sie wie üblich die Strategie `Java DL` aus. Stellen Sie unter Java DL Options — Loop treatment den Wert None ein. Mit dieser Einstellung stoppt die Strategie, wenn die symbolische Ausführung eine Schleife erreicht hat. Lassen Sie die Strategie laufen.

(c) In der resultierenden Situation kann die Regel `loopUnwind` ausgeführt werden. Schauen Sie sich die Sequenz vor Ausführung der Regel und diejenige danach an. Erklären Sie die Funktionsweise der `loopUnwind` Regel.

(d) Führen Sie den Beweis mithilfe der eingestellten Strategie zu Ende.

`p7.key` Betrachten Sie erneut die Klasse `MyClass.java` und ihre Unterklassen und überlegen Sie sich, was die Methode `p` zurückliefert. Ersetzen Sie `XXX` durch den Rückgabewert einer `MyClass2`-Instanz beim Aufruf ihrer `p`-Methode. Beweisen Sie wie zuvor Ihre Behauptung unter Zuhilfenahme der Strategien.

Abgabe bis 16.12.
Es braucht pro Gruppe nur *eine* Lösung abgegeben werden.
Die Abgabe der Übungsblätter erfolgt mit dem SVN System. Dazu legen Sie die abzugebenden Dateien im SVN ab und kopieren sie mit SVN in den Unterordner *abgabe/<nr>* wie in Aufgabe 2 auf Blatt 1 beschrieben.
Einige Aufgaben verlangen eine schriftliche Bearbeitung, diese ist dann je nach Komplexität als ASCII, html, ps- oder pdf-Dokument abzugeben. Auf *keinen* Fall im MS Word `doc`-Format.

Praktikums-Webseite: `http://lfm.iti.uni-karlsruhe.de/keyprakt0809.php`
Christian Engel: R. 106, Tel. 608-4338, E-Mail: `engelc@ira.uka.de`
Benjamin Weiß: R. 309, Tel. 608-4324, E-Mail: `bweiss@ira.uka.de`