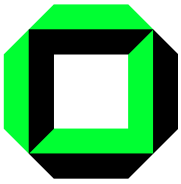

Universität Karlsruhe
Intitut für Theoretische Informatik
Prof. Dr. P. H. Schmitt
C. Engel, F. Werner

15. November 2006

Praktikum
Formale Entwicklung objektorientierter Software
Übungsblatt 3

Aufgabe 6

Geben Sie analog zu den Folien die linke und die rechte Regel für die Äquivalenz $\leftrightarrow$ an.

Aufgabe 7

Laden Sie sich das Archiv mit den Problemdateien von der Praktikums-Webseite herunter. Entpacken Sie die Datei mit dem Befehl:

```
tar xzvf aufg7.tgz
```

Sie erhalten im Verzeichnis `aufg7/` 6 Dateien mit Namen `p1.key` bis `p6.key`. Jede dieser Dateien enthält eine Beweisaufgabe, die Sie mit dem KeY-Beweiser lösen sollen.

Starten Sie den KeY-Beweiser, indem Sie (wenn Sie die Source-Code-Version intalliert haben) im `bin/` Verzeichnis aus `runProver` aufrufen oder (wenn Sie die Byte-Code-Version intalliert haben) im `bin/` Verzeichnis `startProver` aufrufen.

Die Probleme sollen zunächst *ohne* Anwendung von Strategien gelöst werden. Versuchen Sie sich bei jedem Schritt darber klar zu werden, was Sie gerade machen!

Problemdateien können jetzt mit dem Menüpunkt `File | Load` geladen werden. **Zur Abgabe** speichern Sie die Aufgaben mit `File | Save` als `p1.proof` bis `p6.proof`. Sie sollten auch in der Lage sein, Ihre Lösung vorzuführen. Es folgen einige Hinweise zu den einzelnen Problemen:

`p1.key` Zu beweisen ist die triviale Aussage `true`. Klicken Sie auf die Formel `true`. Es erscheint ein Menü mit möglichen Regelanwendungen. `close_by_true` schließt den Ast. Es erscheint ein Dialog mit der Meldung „Proved“, d.h. der Beweis ist fertig. Das geschlossene Ziel ist in der linken Hälfte des Beweiserfensters grün dargestellt.

`p2.key` Hier ist zu zeigen, daß $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$ eine (aussagenlogische) Tautologie ist. Wenden Sie wieder Regeln an, indem Sie auf die Operatoren (`->`, `<->`, `!`) klicken. Die für Sie interessanten Regeln für die Aussagenlogik haben Namen, die sich aus dem Operator und der Seite der Sequenz zusammensetzen, also etwa `equiv_right` für eine Äquivalenz rechts vom Sequenzenpfeil und `imp_left` für eine Implikation links. Wenn Sie mit der Mauszeiger etwas länger über dem Regelnamen verweilen, wird ein “Tooltip” angezeigt, der sagt, was eine Regelanwendung machen wird. Einige der Regelanwendungen werden zu Fallunterscheidungen („Case 1/2“ im linken Teil des Fensters) führen. Durch Auswahl der Ziele im linken Teil können Sie auswählen, an welchem Sie arbeiten wollen. Letztlich müssen alle geschlossen werden. Wenn eine Formel links *und* rechts vom Sequenzenpfeil vorkommt, kann ein Ast geschlossen

werden. Dazu wendet man bei der rechten Formel `close_goal` an. Es wird ein Instanziierungsdialog angezeigt, der aber in diesen Aufgaben einfach mit dem „Apply“ Knopf quittiert werden kann. Insgesamt entstehen vier zu schließende Äste.

`p3.key` Es geht um die prädikatenlogische Tautologie $(\exists x.\forall y.p(x,y)) \rightarrow (\forall v.\exists u.p(u,v))$. Überlegen Sie sich zunächst, warum diese Aussage gilt, und wie man sie beweisen könnte. Zerlegen Sie wieder die Formel, indem Sie bei den äußeren Operatoren anfangen. Wenn Sie einen $\exists$-Quantor links bzw. einen $\forall$-Quantor rechts abarbeiten, werden Sie nach einem Namen für die einzuführende *Skolemkonstante* gefragt. Diesen können Sie in einem Instanziierungsdialog eingeben; wenn Sie stets mit dem Standardnamen zufrieden sind, können Sie auch Options | Minimize Interaction auswählen und werden nicht mehr gefragt. Um einen $\forall$-Quantor links bzw. einen $\exists$-Quantor rechts mit einem bereits in der Sequenz vorhandenen Term t zu instanziieren benutzt man am besten die `inst_all` resp. `inst_ex` Regel. Dazu klickt man *nicht* auf die quantifizierte Formel, sondern auf den Term t. Solange nur eine $\forall/\exists$ Formel für die Instantiierung in Frage kommt, kann man den erscheinenden Dialog wieder mit „Apply“ quittieren. Als dritte Alternative können Sie einfach den Term t auf die zu instanziierende Variable ziehen.

`p4.key` Hier ist zu zeigen, daß $a = b \wedge b = c \rightarrow a = c$, wobei a, b, c irgendwelche Konstanten sind. Zerlegen Sie die Formel wieder mit `imp_right` und `and_left`. Sie können nun Gleichungen auf Terme anwenden. Um etwa die Gleichung `a=b` irgendwo anzuwenden, wählen Sie die Regel `make_insert_eq` auf dieser Gleichung. Es existiert nun eine Regel `insert_eq` mit für jedes Vorkommen des Terms `a` angeboten wird, und die diesen in `b` überführt. Probieren Sie es aus! Wenn Sie eine Gleichung in der anderen Richtung, also von rechts nach links anwenden wollen, können Sie sie mit `commute_eq` umdrehen, um dann nochmals `make_insert_eq` anzuwenden.

`p5.key` Das Problem lautet $((\forall x.f(g(x)) = g(x)) \wedge (g(a) = b)) \rightarrow (f(b) = g(a))$. Das Problem kann mit den bisher vorgestellten Techniken gelöst werden. überlegen Sie sich vorher, welche Instanz der all-quantifizierten Formel sie benötigen, und besorgen Sie sich diese mit einer der oben erwähnten Möglichkeiten, einen Allquantor zu instanziieren.

`p6.key` Argumentieren Sie zuerst, weshalb folgendes eine Tautologie ist

$$\forall x.\forall y.\forall z.(((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)) \wedge \neg P(x,x)) \rightarrow \forall x.\forall y.(P(x,y) \rightarrow \neg P(y,x))$$

Beweisen Sie danach im KeY-Beweiser.

`p7.key` Lesen Sie sich folgendes Rätsel aus dem Lufthansa-Magazin 11/2002 durch:

> Genau zwei der Personen A, B, C, D und E lügen. Welche?
> A: “B lügt dann und nur dann, wenn D die Wahrheit sagt!”
> B: “Sollte C ein wahrheitsliebender Mensch sein, so ist entweder A oder D ein Lügner.”
> C: “ E kann man auf keinen Fall trauen, und auch A oder B hält bzw. halten es mit der Wahrheit nicht so genau!”
> D: “Würde B die Wahrheit sagen, dann könnte man A oder C vertrauen!”
> E: “Unter den Personen A, C und D befindet sich mindestens ein Lügner!”

Überlegen Sie sich die Lösung des Problems und formalisieren Sie die Aufgabenstellung und ihre Lösung als logische Formel. Beweisen Sie diese dann mit Hilfe des KeY-Beweisers. Sie können hierzu die Strategie *Java DL* verwenden!

Hinweis: Die zu beweisende Formel geben Sie im `\problem` Abschnitt von `p7.key` an. Die anderen Abschnitte in `p7.key` dienen der Deklaration der benötigten Signatur.

Aufgabe 8

Gegeben sei wieder das Authentifikationssystem aus Aufgabe 5 (2. Übungsblatt). Folgende Eigenschaft sei gefordert:

> Jede Chipkarte besitzt eine eindeutige Identifikationsnummer.

Formalisieren Sie die obige Eigenschaft

- in JML
- in getypter (sortierter) Logik erster Stufe (FOL). Geben Sie hier auch die zugehörige FOL Signatur explizit an.

Aufgabe 9

Laden Sie sich das Archiv mit den Problemdateien von der Praktikums-Webseite herunter. Legen Sie ein Verzeichnis für die Aufgaben an, legen Sie die Datei dort hinein, und entpacken Sie sie mit dem Befehl:

```
tar xzvf aufg9.tgz
```

Sie erhalten 7 Dateien mit Namen `p9_1.key` bis `p9_7.key`. Wie schon in Aufgabe 7 enthält jede dieser Dateien eine Beweisaufgabe, die Sie mit dem KeY-Beweiser lösen sollen. Verfahren Sie zum Laden und Beweisen der Probleme wie in Aufgabe 7 beschrieben. Zur Abgabe gehen Sie ebenfalls wie in Aufgabe 7 vor.

`p9_1.key` Hier soll gezeigt werden daß der Wert einer `int` Variablen `i` in einem Java Programm nach Ausführung der Anweisung `i++` um eins höher ist als vorher.

Ansonsten geht es hier darum, schrittweise die Programmformeln abzuarbeiten. Befindet sich der Mauszeiger über einer Programmformel, wird die jeweils nächste zu behandelnde Anweisung grau unterlegt. Meist ist nur eine Regel anwendbar, etwa `eliminate_variable_declaration`, `assignment`, etc.

`p9_2.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 alle Schritte manuell ohne Hilfe der automatischen Beweissuche (Strategien) 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.

`p9_3.key` Bevor Sie die Problemdatei in den KeY-Beweiser laden, öffnen Sie die Datei in einen Editor Ihrer Wahl (z.B. xemacs) und betrachten Sie sich 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) von `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. Wenn Ihnen die Java-Semantik bezüglich der Behandlung von Exceptions nicht mehr geläufig ist, so schlagen 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.

(b) Führen Sie noch die Regel `try_throw` aus und beschreiben Sie, was diese Regel gemacht hat.

(c) 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 linken Teil des Beweiserfensters können Sie verfolgen, welche Regeln angewandt wurden.

`p9_4.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.

`p9_5.key` Nun sollen Sie sich um das Ergebnis des folgenden Programmstücks (relativ zu `i0`) kümmern:

```
i=i0;
if (((j=i+=1)+ ++i)== i=++i)
  i=i++;
```

Überlegen Sie sich, wie die Ausdrücke ausgewertet werden. Ersetzen Sie danach wieder `XXX` durch den richtigen Ausdruck, der nur aus `+`, `i0` und Zahlenliteralen bestehen darf. Beweisen Sie die veränderte Eingabedatei.

`p9_6.key` Die folgende Regel (`unwind_while`)

$$\frac{\Gamma \vdash < \texttt{if (b) } \{\texttt{l1} : \{\texttt{l2} : \{\alpha'\} \texttt{ while (b)}\{\alpha\}\}\} > \Phi,\ \Delta}{\Gamma \vdash < \texttt{while (b)}\{\alpha\} > \Phi,\ \Delta}$$

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

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

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

(b) Laden Sie das Problem in den Beweiser. Stellen Sie die Anzahl der maximalen automatischen Regelanwendungen auf 100 und wählen Sie die Strategie `Java DL Loop treatment - None` aus. Diese Strategie stoppt vor der *symbolischen Ausführung* einer Schleife. In solch einer Situation kann die Regel `unwind_while` ausgeführt werden.

Führen Sie mithilfe der angegebenen Strategie den Beweis bis sie stoppt und führen Sie dann `unwind_while` aus. Schauen Sie sich die Sequenz vor Ausführung der Regel und diejenige danach an. Erklären Sie die Funktionsweise der `unwind_while` Regel.

(c) Beweisen Sie Ihre modifizierte Beweisereingabe `p9_6.key` mithilfe der angegebenen Strategie zu Ende und speichern Sie die Lösung ab.

`p9_7.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.

Aufgabe 10 (Updatevereinfachung)

Vereinfachen Sie die unten stehenden Terme, soweit wie möglich, gemäß den im Vortrag vorgestellten Regeln. Geben Sie dabei die einzelnen Schritte, also nicht nur das Endergebnis, an.

(a) $\{x := 3\}(x \doteq 3 \rightarrow < \mathtt{x} = \mathtt{x} + \mathtt{1}; > x \doteq 4)$

(b) $\{o.a := o\}o.a.a.a.b$

Abgabe bis 29.11.

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 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://i12www.ira.uka.de/~engelc/lehre/keypraktWS0607/`
Christian Engel: Email: `engelc@ira.uka.de`
Frank Werner: Zi. 308, Tel. 608-7322, Email: `werner@iti.uni-karlsruhe.de`