Universität Koblenz-Landau
Institut für Informatik
Dr. Bernhard Beckert
`www.uni-koblenz.de/~beckert`

WS '02/03

Praktikum

Entwicklung objektorientierter Software mit formalen Methoden

Aufgabenblatt 3

Aufgabe 1

Installieren Sie die (neueste) „Source code"-Version des KeY-Systems.

- Stellen Sie zunächst sicher, daß TogetherCC und die KeY-Libraries zur Verfügung stehen.
- Laden Sie von der Seite `http://i12www.ira.uka.de/~key/download.htm` die neueste Version der „Java source code version" des KeY-Systems herunter.
- Entpacken Sie die Dateien in das Verzeichnis `key`.
- Wechseln Sie in des Verzeichnis `key/system`.
- Editieren Sie die Date `Makefile`: Setzen Sie

  ```
  JAVAC=javac
  ```

  damit `javac` (und nicht wie voreingestellt `jikes`) verwendet wird. Achten Sie zudem darauf, daß

  ```
  INSTALLSCRIPT=dist_extras/install/install_dev.sh
  ```

  gesesetzt ist.
- Starten Sie die Installation mit dem Kommando „`make all`".

  Der lokale Pfad von Together lautet `/usr/local/Together`.

  Sollten Sie das System re-compilieren wollen, löschen Sie zunächst alle Binärdateien mit „`make clean`".

  Treten bei der Installation Probleme oder Unklarheiten auf, dann notieren Sie diese bitte (auch solche, die Sie selbst beheben konnten), damit die Installationsroutine bzw. -anleitung korrigiert werden kann.

Aufgabe 2

Laden Sie sich das `tar`-Archiv mit den Problemdateien zu dieser Aufgabe von der Praktikums-Webseite herunter. Legen Sie ein Verzeichnis für die Aufgaben an (sinnvoller Weise als Unterordner von `key/system`), legen Sie die `tar`-Datei dort hinein, und entpacken Sie sie mit dem Befehl:

```
gtar zxf aufgaben.tgz
```

Sie erhalten 7 Dateien mit Namen `p1.key` bis `p7.key`. Jede dieser Dateien enthält eine Beweisaufgabe, die Sie mit dem KeY-Beweiser lösen sollen.

Starten Sie den (Standalone-)KeY-Beweiser ohne TogetherCC, indem Sie vom `system`-Verzeichnis aus `runProver` aufrufen.

Die Probleme sollen zunächst ohne Anwendung von Heuristiken gelöst werden. Wählen Sie den Menüpunkt `Options | Heuristics` und stellen Sie sicher, daß sich in der rechten Liste (aktive Heuristiken) keine Einträge befinden. Etwaige Einträge können mit einem Doppelklick in die linke Liste (deaktivierte Heuristiken) bewegt werden.

Problemdateien können jetzt mit dem Menüpunkt `File | Load` geladen werden. 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 $(p \to q) \leftrightarrow (\neg q \to \neg p)$ 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. 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 Instantiierungsdialog 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 $(\exists x.\forall y.p(x,y)) \quad \to \quad (\forall v.\exists u.p(u,v))$. 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. Um einen $\forall$-Quantor links bzw. einen $\exists$-Quantor rechts mit einem bereits in der Sequenz vorhandenen Term t zu instantiieren benutzt man am besten die `inst_all`- bzw. die `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. Die Terme `[eps x...]` etc. entsprechen neu eingeführten Konstanten. Man kann ihnen mit `Name Term` beliebige Namen zuweisen.

`p4.key` Hier ist zu zeigen, daß $a = b \wedge b = c \to a = c$, 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 `inst_all`.

`p6.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. Um bequemer mit Zahlen umgehen zu können sollten sie mit dem Menüpunkt `Options | Heuristics` die beiden Heuristiken `simplify` und `simplify_int` aktivieren. Ferner empfiehlt es sich, den Menüeintrag `View | Pretty&Untrue` zu selektieren, da sonst arithmetische Ausdrücke in etwas ungewohnter Form dargestellt werden. Selektieren Sie auch den Knopf „Autoresume Heuristics“, damit die Heuristiken nach jeder Regelanwendung ausgeführt werden. Die `simplify` Heuristik sorgt übrigens auch dafür, daß viele der Standardregeln für Prädikatenlogik automatisch angewandt werden, und `make_insert_eq` automatisch für jede Gleichung durchgeführt wird. Im linken Teil des Beweiserfensters können Sie jeweils verfolgen, welche Regeln von den Heuristiken angewandt wurden.

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.

`p7.key` Es ist zu zeigen, daß nach Ausführung von

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

die Variable `i` den Wert 12 hat. Arbeiten Sie das Programm wieder der Reihe nach ab. `throw` und `if` Anweisungen mit komplexen Ausdrücken müssen erst mit den Regeln `evaluate throw` bzw. `if_else_eval` vereinfacht werden. Wenn die Argumente dann nur noch Variablen sind, kann man die `try_throw` und `if_else_split` anwenden. Dabei ist zu bemerken, daß die `if_else_split`-Regel, im Gegensatz zu den meisten anderen Regeln, nicht für die Programmformel, sondern für selbige inklusive den davor stehenden Updates angeboten wird. (Mit gutem Grund—Warum?) Man muß also mit der Maus auf das vorderste Updates zeigen, um die Regel anzuwenden. Wenn Sie einen Zweig mit der Formel `TRUE=FALSE` auf der linken Seite bekommen, können Sie diesen mit der Regel `boolean_not_equal` schließen.