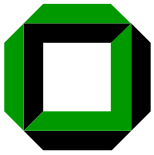

UNIVERSITÄT KARLSRUHE (TH)
Institut für Theoretische Informatik
Prof. Dr. B. Beckert
M. Ulbrich

Korrigierte Fassung:
s. Gleichung (6) auf Seite 3
n statt m

Formale Systeme, WS 2008/2009

Praxisaufgabe 2: Theorembeweiser KeY

Abgabe der Lösungen bis zum 15.02.09 über die Webseite der Vorlesung
`http://i12www.ira.uka.de/~beckert/Lehre/Formale-Systeme/login/`

Für die vollständige Lösung dieser Praxisaufgabe – sie besteht aus den beiden Teilaufgaben 1 und 2 – erhalten Sie 1,5 Bonuspunkte für die Abschlussklausur (bitte beachten Sie die Erläuterung zu Bonuspunkten auf der Webseite zur Vorlesung).

A Informationen zum KeY-System

A.1 Das KeY-System

Zusammen mit unseren Partnern an der Universität Koblenz und an der Chalmers University of Technology in Göteborg wird an unseren Institut das KeY-System entwickelt. Es ist ein Softwareverifikationwerkzeug, mit dem die Übereinstimmung von Java Card-Software und ihrer Spezifikation formal bewiesen werden kann.

Die Logik, auf der das KeY-System basiert, ist eine dynamische Prädikatenlogik. In dieser dynamischen Logik ist die in der Vorlesung eingeführte Prädikatenlogik vollständig enthalten. Deshalb können wir KeY auch benutzen, um rein prädikatenlogische Probleme zu formulieren und zu beweisen.

A.2 Weitere Informationen und Literatur zu KeY

Auf der Internetseite der Vorlesung steht eine Einführung zu Beweisen von prädikatenlogischen Formeln mit KeY (`KeYIntro.pdf`). Bitte erarbeiten Sie sich diese Einführung.

Für weitergehende Fragen bietet sich die Lektüre des KeY-Buches [BHS07] an und darin vor allem Kapitel 10, das eine tiefere Einführunng in KeY bietet. Kapitel 2 des Buches erklärt fundiert die prädikatenlogischen Grundlagen, auf denen KeY basiert.

A.3 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.

Version 1.2 des Werkzeuges ist die zuletzt veröffentlichte Version und kann über die KeY-Homepage[1] bezogen werden.

Wenn Sie die Software "Java WebStart" installiert haben, können Sie KeY direkt aus dem Internetbrowser[2] heraus starten, ohne etwas installieren zu müssen.

Auf den Studentenrechnern im Poolraum der ATIS ist diese Version ebenfalls installiert und kann durch den Aufruf

[1] `http://www.key-project.org`
[2] über die Adresse `http://www.key-project.org/download/releases/webstart/KeY.jnlp`

```
startkey
```

gestartet werden, wenn – wie auf dem ersten Praxisblatt beschrieben – die entsprechende Konfigurationsdatei geladen worden ist. Alternativ dazu können Sie KeY für diese Aufgaben auch auf jedem anderen Rechner benutzen.

A.4 Hinweis zur Konfiguration von KeY

Die automatische Beweisbarkeit hängt bei KeY bisweilen sehr von den verwendeten Einstellungen ab.

Für Teilaufgabe 1 empfiehlt es sich, im Bereich unten links in KeY auf der Registerkarte "Proof Search Strategy" die Einstellung **FOL** zu wählen.

Für Teilaufgabe 2 wählen Sie am besten **Java DL** und belassen alle weiteren Schalter in ihrer Standardstellung.

A.5 KeY-Problemdateien zu den Teilaufgaben

Für Teilaufgabe 1 erstellen Sie bitte selbst eine KeY-Problemdatei.

Für Teilaufgabe 2 finden Sie auf der Webseite der Vorlesung die Datei `list.key` zum Herunterladen vor. In diese müssen Sie noch die zu beweisende Behauptung eintragen (wie unten zu Teilaufgabe 2 beschrieben).

A.6 Abgabe der Lösungen

Bitte speichern Sie die beiden Beweise in KeY als „.key.proof"-Dateien ab. Auf der Webseite der Vorlesung steht Ihnen ein Formular zur Verfügung, um die beiden Lösungen einzureichen.

Für unvollständige Beweise werden Punkte auch anteilig vergeben.

B Teilaufgabe 1: Der Fall „Tante Agatha" wird aufgerollt

Eine bereits bekannte Aufgabe, die noch auf ihre Lösung wartet, zu Beginn: Auf dem 3. Übungsblatt wurde in Aufgabe 8 ein logisches Puzzle vorgestellt, das Teil einer großen Benchmarkbibliothek für Beweiser ist. Über der dort eingeführten Signatur lassen sich die gegebenen Indizien folgendermaßen formalisieren[3]:

$$
\begin{aligned}
& \forall x(x \doteq a \vee x \doteq b \vee x \doteq c) \\
\wedge\ & \exists x(kills(x,a)) \\
\wedge\ & \forall x\ \forall y\ (kills(x,y) \rightarrow hates(x,y)) \\
\wedge\ & \forall x\ (hates(a,x) \rightarrow \neg hates(c,x)) \\
\wedge\ & \forall x\ \forall y\ (kills(x,y) \rightarrow \neg richer(x,y)) \\
\wedge\ & \forall x\ ((\neg richer(x,a) \vee hates(a,x)) \rightarrow hates(b,x)) \\
\wedge\ & \forall x\ \exists y\ \neg hates(x,y) \\
\wedge\ & \forall x\ (\neg x \doteq b \rightarrow hates(a,x)) \\
\wedge\ & \neg a \doteq b
\end{aligned}
$$

- Formalisieren Sie nun auch noch die Aussage „Tante Agathe hat sich selbst umgebracht.",
- schreiben Sie eine Problembeschreibungsdatei für den Beweiser KeY und
- beweisen Sie mit KeY, dass aus den Indizien folgt, dass Tante Agatha sich selbst umgebracht haben muss.

[3]Die letzten beiden Aussagen waren in Aufgabe 3.8 nicht enthalten, sind aber notwendig, um die Behauptung schließen zu können

C Teilaufgabe 2: Abstrakte Datentypen

Wir betrachten die Definition eines termerzeugten Datentyps (Liste von Zahlen), und einer Operation darauf (Einfügen eines Elements). Aufgabe ist es, zu beweisen, dass die Operation eine Eigenschaft (Sortiertheit) erhält.[4]

Wir betrachten zwei Sorten von Objekten: ganze Zahlen (Sorte `int`) und Listen von ganzen Zahlen (Sorte `lst`). In der Signatur gibt es die folgenden drei Funktionen

$$\begin{aligned} \texttt{nil} &: \to \texttt{lst} \\ \texttt{cons} &: \texttt{int} \times \texttt{lst} \to \texttt{lst} \\ \texttt{insert} &: \texttt{int} \times \texttt{lst} \to \texttt{lst} \end{aligned}$$

und das einstellige Prädikat `sorted`, das als Argument einen Term der Sorte `lst` benötigt.

Wir wollen als Interpretationen der Sorten nur das "kanonische" Universum betrachten, in dem die Sorte `int` der Menge $\mathbb{Z}$ der ganzen Zahlen und `lst` der Menge aller (endlichen) Listen von ganzen Zahlen entspricht. Die Menge der Listen wird von den beiden Funktionen `nil` und `cons` erzeugt.

Da wir uns auf bestimmte Interpretationen beschränken, können wir – wie schon in Aufgabe 3 auf dem 7. Übungsblatt – weitergehende korrekte Regelschemata formulieren. Um Aussagen über Listen beweisen zu können, nehmen wir das Regelschema für die strukturelle Induktion über Listen hinzu:

$$\frac{\begin{array}{ll} \Gamma & \Rightarrow \{l/\texttt{nil}\}\varphi,\ \Delta \\ \Gamma & \Rightarrow \forall n \forall l(\varphi \to \{l/\texttt{cons}(n,l)\}\varphi),\ \Delta \end{array}}{\begin{array}{ll} \Gamma & \Rightarrow \forall l\ \varphi,\ \Delta \end{array}}\ (\mathsf{listInduction})$$

Diese Regel ist korrekt, da die Sorte `lst` von den beiden Konstruktoren `nil` und `cons` termerzeugt wird, also jedes Element der Sorte als Term über diesen Funktionen dargestellt werden kann.

Die Funktionen und das Prädikat sind durch folgende Axiome (partiell) festgelegt:

$$\begin{array}{rlcll} & \texttt{sorted}(\texttt{nil}) & & & (1) \\ \forall n & \texttt{sorted}(\texttt{cons}(n,\texttt{nil})) & & & (2) \\ \forall n \forall m \forall l & \big(\texttt{sorted}(\texttt{cons}(n,\texttt{cons}(m,l))) & \leftrightarrow & n \le m \wedge \texttt{sorted}(\texttt{cons}(m,l))\big) & (3) \\ & & & & \\ \forall n & \texttt{insert}(n,\texttt{nil}) & \doteq & \texttt{cons}(n,\texttt{nil}) & (4) \\ \forall n \forall m \forall l & n \le m \to \texttt{insert}(n,\texttt{cons}(m,l))) & \doteq & \texttt{cons}(n,\texttt{cons}(m,l)) & (5) \\ \forall n \forall m \forall l & n > m \to \texttt{insert}(n,\texttt{cons}(m,l))) & \doteq & \texttt{cons}(m,\texttt{insert}(\underline{n},l)) & (6) \end{array}$$

Hier und im Folgenden sind die Variablen n und m vom Typ `int` und l vom Typ `lst`.

C.1 Die zu beweisende Aussage

Beweisen Sie Folgendes mit KeY:

> Wenn eine (beliebige) Liste von ganzen Zahlen sortiert ist, dann ist auch die Liste, die durch Einfügen eines beliebigen weiteren Wertes mittels `insert` entsteht, sortiert.

C.2 Aufgabendatei in KeY

Auf der Seite zur Vorlesung finden Sie eine KeY-Datei, in der die Axiome (1) bis (6) formalisiert sind. Sie sind darin als Sequenzenkalkülregeln formuliert (s. Tabelle 1), die z.T. automatisch angewendet werden.

Am Ende der Datei müssen Sie in den `\problem`-Abschnitt Ihre Formalisierung der zu beweisenden Aussage eintragen. Formulieren Sie diese Aussage so, dass die resultierende Formel eine Struktur hat, die es erlaubt, das Regelschemas listInduction darauf anzuwenden.

[4] Derartige Fragestellungen treten in der Programmverifikation durchaus auf. Man kann sich z.B. vorstellen, dass diese Formalisierung von einer Definition einer Funktion in einer funktionalen Programmiersprache (z.B. SML) herrührt. Von dieser Definition möchte man formal beweisen, dass sie die Sortierung einer Liste erhält.

Regelname	Axiom
sortedNil	(1)
sortedConsNil	(2)
sortedConsCons	(3)
insertNil	(4)
insertCons	(5) und (6)

Tabelle 1: Regeln zu den Axiomen

C.3 Hilfreiche Lemmata

Mit einer einzigen Induktion kann die Aussage leider nicht bewiesen werden. Sie werden im Laufe des Beweises an Punkte kommen, an denen Lemmata benötigt werden, die selbst wieder durch Induktion zu beweisen sind.

Wir benutzen die Regel (Cut), die in Aufgabe 2 auf dem 7. Übungsblatt vorgestellt wurde, um ein Lemma einzufügen:

$$\frac{\overbrace{\Gamma, \varphi \Rightarrow \Delta}^{(A)} \qquad \overbrace{\Gamma \Rightarrow \varphi, \Delta}^{(B)}}{\Gamma \Rightarrow \Delta}\ (\mathsf{Cut})$$

(Cut) teilt den Beweis in zwei Äste auf:

Teil (A) Hier steht die eingefügte Formel φ *links* des Sequenzenpfeiles.

Damit steht sie auf diesem Ast als weitere Voraussetzung zur Verfügung und kann benutzt werden, um das ursprüngliche Ziel zu schließen (Verwenden des Lemmas).

Teil (B) Hier steht φ *rechts* des Sequenzenpfeiles.

Auf diesem Ast ist φ also das Beweisziel und muss gezeigt werden (Beweis des Lemmas).

Wenden Sie also, wenn Sie ein Lemma einfügen wollen, die Cut-Regel an. Im folgenden sind zwei sehr hilfreiche Lemmata beschrieben.

C.3.1 Lemma 1

Folgende Formel stellt ein hilfreiches Lemma dar:

$$\forall l \forall n \forall m \big(\mathtt{sorted}(\mathtt{cons}(m, l)) \land m \leq n \rightarrow \mathtt{sorted}(\mathtt{cons}(m, \mathtt{insert}(n, l)))\big)$$

C.3.2 Lemma 2

Außerdem ist folgende Aussage (die Sie selbst noch formalisieren müssen) ein hilfreiches Lemma:

Für jede nicht-leere sortierte Liste gilt, dass die Liste, die entsteht, wenn man das erste Element weglässt, ebenfalls sortiert ist.

Literatur

[BHS07] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach.* LNCS 4334. Springer-Verlag, 2007.