Formale Modellierung SoSe 15 Christoph Lüth

4. Übungsblatt

Ausgabe: 12.06.15
Abgabe: 25.06.15

In diesem Aufgabenblatt gehen wir den Schritt zur Logik höheren Stufe in Isabelle, kurz Isabelle/HOL. Um HOL zu benutzen, importieren wir in Isabelle einfach die Theorie `Main`:

```
theory MyThy
imports Main
begin
```

HOL hat algebraische Datentypen und rekursive Funktionsdefinitionen, fast wie in Haskell, aber erlaubt auch Beweise über diesen. Der Slogan ist

HOL = Functional Programming + Logic

4.1 *Bäume* *10 Punkte*

Bäume sind neben Listen einer der populärsten Datentypen der Informatik. In Isabelle/HOL können Binäre Bäume als algebraischer Datentyp definiert werden (genau wie in Haskell):

```
datatype 'a tree = Node 'a "'a tree" "'a tree"
                 | Leaf
```

(i) Definieren Sie für diesen Datentypen `tree` eine Funktion `map` (mit der offensichtlichen Semantik), und zeigen Sie das sogenannte *map fusion lemma* (wobei `o` die Komposition zweier Funktionen darstellt)

```
map (f o g) = map f o map g.
```

(ii) Definieren Sie eine Funktion

```
fun inorder :: "'a tree => 'a list"
```

welche die Liste der in dem Baum enthaltenen Knoten (in Inorder-Traversion) zurückgibt.

Jetzt wollen wir zeigen, dass diese Liste die gleichen Knoten wie der Baum enthält. Dazu definieren wir zwei Funktionen

```
tcount :: "'a tree => 'a=> nat"
lcount :: "'a list => 'a=> nat"
```

welche zählen, wie oft ein Element in einem Baum bzw. einer Liste auftritt (mit anderen Worten, sie berechnen die Multimenge der Elemente).

Formulieren und beweisen Sie mit Hilfe dieser Funktionen, dass die durch `inorder` erzeugte Liste dieselben Knoten (Multimenge der Elemente wie oben) enthält wie der ursprüngliche Baum.

Hinweis: Sie werden ein Hilfslemma über das Verhältnis von `lcount` und der Listenkonkatenation (Infix-Operator `@`) benötigen.

(iii) Beschreibt diese Spezifikation vollständig die `inorder`-Funktion? Wie könnte eine vollständige Spezifikation aussehen? Welche anderen Funktionen erfüllen diese Spezifikation?

4.2 *Graphen* *10 Punkte*

Jetz definieren wir Graphen, wie vom letzten Übungsblatt, aber in HOL. Formulieren Sie dazu einen Datentypen

```
datatype ('v, 'e) graph = ...
datatype ('v, 'e) path  = ...
```

eines über dem Knoten- und Kantentyp parametrisierten Graphen, sowie einen Typ für Pfade in solch einem Graphen. Der Typ für Pfade ist zweckmäßigerweise rekursiv: ein Pfad besteht entweder aus nur einem Knoten (dann ist es ein leerer Pfad), oder einem Knoten, einer Kanten, und einem weiteren Pfad. Wir schreiben dafür $p = \langle v_1, e_1, v_2, \ldots, v_n \rangle$

Definieren Sie eine Prädikat `valid_path`, welches bestimmt ob ein Pfad p wie oben in einem Graphen G gültig ist, d.h. alle Knoten v_i sind in dem Graphen enthalten, und für alle Kanten e_i gilt, dass e_i in G eine Kanate von v_i nach v_{i+1} ist (geschrieben $G \models e_i : v_i \Longrightarrow v_{i+1}$).

Definieren Sie ferner ein Prädikat `vertex_cover`, welches bestimmt ob ein Pfad p alle Knoten eines Graphen G besucht.

Jetzt können wir eine Funktion `find_cover` definieren, welche für einen gegebenen Graphen einen Pfad berechnet, der alle Knoten des Graphen besucht. Definieren Sie `find_cover`, und zeigen Sie, dass die Funktion der Spezifikation genügt.

Hinweise: Die Funktion wird notwendigerweise partiell sein. Es ist nicht nötig, den kürzesten solchen Pfad zu berechnen, oder die Berechnung effizient zu gestalten. Sie werden mehrere Hilfsfunktionen und Lemmata benötigen.