Karlsruher Institut für Technologie

Institut für Programmstrukturen und Datenorganisation
Lehrstuhl Programmierparadigmen
Am Fasanengarten 5
76131 Karlsruhe
http://pp.ipd.kit.edu/

Theorembeweiserpraktikum – SS 2013

http://pp.ipd.kit.edu/lehre/SS2013/tba

Blatt 3: Datentypen und Rekursion

Abgabe: 6. Mai 2013
Besprechung: 7. Mai 2013

1 Rätsel: Der reiche Großvater

Wir beginnen zum Aufwärmen damit, nicht-triviale Aussagen zu beweisen. Zeigen Sie, dass gilt:

Wenn jeder arme Mann einen reichen Vater hat,
dann gibt es einen reichen Mann mit einem reichen Großvater.

theorem `"(∀x. ¬ rich x ⟶ rich (father x)) ⟶ (∃y. rich y ∧ rich(father(father y)))"`
⟨*solution*⟩

Hinweise:

- Gibt es überhaupt einen reichen Mann?
- Überlegen Sie sich den Beweis erst auf Papier und versuchen Sie ihn dann möglichst analog in Isar zu formulieren!
- Sie werden in jedem Fall Fallunterscheidungen brauchen.

2 Cantors Theorem

Sie sollen nun Cantors Theorem beweisen; dieses sagt aus, dass es keine surjektive Funktion von einer Menge auf ihre Potenzmenge geben kann. Formalisiert:

theorem `"∃S. S ∉ range (f :: 'a ⇒ 'a set)"`
⟨*solution*⟩

Dabei bezeichnet `range f` die Wertemenge einer Funktion.
Hinweise:

- Der Knackpunkt des Beweises ist das Finden der richtigen Menge `S`. Versuchen Sie es erstmal alleine, erinnern Sie sich (falls bekannt) an das sogenannte *Cantor'sche Diagonalverfahren*. Ansonsten versuchen Sie ihr Glück im Internet, der Name der Übung sollte Hinweis genug sein. ;-)
- Auch hier sollten Sie sich Ihren Beweis erst auf Papier überlegen und dann möglichst analog in Isar übertragen.
- Falls Sie eine Aussage wie `b ∈ range f` haben, lässt sich daraus unmittelbar ein `x` auswählen ("obtainen"), so dass `b = f x` gilt, da die Regel `rangeE: ⟦b ∈ range f; ⋀x. b = f x ⟹ P⟧ ⟹ P` als Eliminationsregel in allen Taktiken des automatischen Schließens existiert.

3 Rekursive Datenstrukturen

In dieser Übung soll eine rekursive Datenstruktur für Binärbäume erstellt werden. Außerdem sollen Funktionen über Binärbäume definiert und Aussagen darüber gezeigt werden. Dazu dürfen Sie auch automatische Taktiken, z.B. `auto` verwenden. Und denken Sie daran: *Recursion is proved by induction!*.
Zuerst definieren Sie den Datentypen für (nichtleere) Binärbäume. Sowohl Blätter (ohne Nachfolger) als auch innere Knoten (mit genau 2 Nachfolgern) speichern Information. Der Typ der Information soll beliebig sein, also arbeiten sie mit Typparameter `'a`.

datatype *'a tree = ...*

Definieren Sie jetzt die Funktionen *preOrder*, *postOrder* und *inOrder*, welche einen *'a tree* in der entsprechenden Ordnung durchlaufen:

fun *preOrder :: "'a tree ⇒ 'a list"*
  where —
fun *postOrder :: "'a tree ⇒ 'a list"*
  where —
fun *inOrder :: "'a tree ⇒ 'a list"*
  where —

Definieren Sie nun eine Funktion *mirror*, welche das Spiegelbild eines *'a tree* zurückgibt.

fun *mirror :: "'a tree ⇒ 'a tree"*
  where —

Seien *xOrder* und *yOrder*, beliebig ausgewählt aus *preOrder*, *postOrder* und *inOrder*. Formulieren und zeigen Sie alle gültigen Eigenschaften der Art:

lemma *"xOrder (mirror xt) = rev (yOrder xt)"*

Definieren Sie die Funktionen *root*, *leftmost* und *rightmost*, welche die Wurzel, das äußerst links bzw. das äußerst rechts gelegene Element zurückgeben.

fun *root :: "'a tree ⇒ 'a"*
  where —
fun *leftmost :: "'a tree ⇒ 'a"*
  where —
fun *rightmost :: "'a tree ⇒ 'a"*
  where —

Beweisen Sie folgende Theoreme oder zeigen Sie ein Gegenbeispiel (dazu kann man u.a. *quickcheck* verwenden). Es kann nötig sein, erst bestimmte Hilfslemmas zu beweisen.

theorem *"hd (preOrder xt) = last (postOrder xt)"* ⟨*solution*⟩
theorem *"hd (preOrder xt) = root xt"* ⟨*solution*⟩
theorem *"hd (inOrder xt) = root xt"* ⟨*solution*⟩
theorem *"last (postOrder xt) = root xt"* ⟨*solution*⟩
theorem *"hd (inOrder xt) = leftmost xt"* ⟨*solution*⟩
theorem *"last (inOrder xt) = rightmost xt"* ⟨*solution*⟩

Und hier noch ein etwas komplizierteres Theorem.

lemma *"(mirror xt = mirror xt') = (xt = xt')"*
⟨*solution*⟩