Karlsruher Institut für Technologie

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

Theorembeweiserpraktikum – SS 2016

http://pp.ipd.kit.edu/lehre/SS2016/tba

Lösung 5: Induktive Prädikate

Abgabe: 23. Mai 2016, 12:00 Uhr
Besprechung: 24. Mai 2016

1 Regeln ohne Basisfall

Zeigen Sie, dass folgende Definition

```
inductive evenempty :: "nat ⇒ bool"
where Add2Ie: "evenempty n ⟹ evenempty (Suc (Suc n))"
```

die leere Menge definiert.

```
lemma evenempty_empty: "evenempty x ⟹ False"
by (induction rule: evenempty.induct)
```

2 Euklid'scher Algorithmus – induktiv

Definieren Sie induktiv das Prädikat *ggT*, welches den größten gemeinsamen Teiler zweier natürlicher Zahlen beschreibt:

```
inductive ggT :: "nat ⇒ nat ⇒ nat ⇒ bool"
where ZeroLeft: "ggT 0 b b"
  | ZeroRight: "ggT a 0 a"
  | SubLeft: "a ≥ b ⟹ ggT (a-b) b c ⟹ ggT a b c"
  | SubRight: "a < b ⟹ ggT (b-a) a c ⟹ ggT a b c"
```

ggT a b g bedeutet, dass *g* der ggT von *a* und *b* ist. Die Definition sollte so nahe wie möglich am Euklid'schen Algorithmus sein: man zieht solange die kleinere von der größeren Zahl ab, bis eine der beiden Zahlen 0 ist; dann ist die andere Zahl der ggT.

Berechnen Sie nun den ggT von 15 und 10.

```
lemma "ggT 15 10 5"
by (fastforce intro: ggT.intros)
```

Wie sieht es bei Ihrem Algorithmus mit Spezialfällen wie dem Folgenden aus?

```
lemma "ggT 0 0 0"
by (rule ZeroLeft)
```

Zeigen Sie, dass der ggT wirklich ein Teiler ist. Sie werden für den Beweis ein oder mehrere geeignete Hilfslemmas brauchen. Suchen Sie entweder geeignete Lemmas in der Bibliothek, oder beweisen Sie sie selbst:

```
lemma ggT_divides: assumes "ggT a b g" shows "g dvd a ∧ g dvd b"
using assms
by (induction rule: ggT.induct) (auto dest: dvd_diffD)
```

Zeigen Sie, dass der ggT der größte gemeinsame Teiler ist:

```
lemma ggT_greatest:
assumes "ggT a b g"
  and "0 < a ∨ 0 < b"
  and "d dvd a" and "d dvd b"
shows "d ≤ g"
using assms
by (induction rule: ggT.induct) (auto intro: dvd_imp_le)
```

Auch hier werden Sie ein Hilfslemma benötigen. Wie verhalten sich `op dvd` und `op ≤`?

Bisher haben wir nur gezeigt, dass `ggT` korrekt ist, aber es könnte sein, dass Ihr Algorithmus nicht für alle `a`,`b` ein Ergebnis berechnet. Zeigen Sie also die Vollständigkeit des Algorithmus:

```
lemma ggT_defined: "∃g. ggT a b g"
proof (induction "a + b" arbitrary: a b rule: nat_less_induct)
  case 1
  show ?case
  proof (cases "b ≤ a")
    case True
    show ?thesis
    proof (cases "b = 0")
      case False
      with ‹b ≤ a› have "(a - b) + b < a + b" by simp
      with 1 have "∃g. ggT (a - b) b g" by blast
      with ‹b ≤ a› show ?thesis by (auto intro: SubLeft)
    qed (auto intro: ZeroRight)
  next
    case False
    show ?thesis
    proof (cases "a = 0")
      case False
      with ‹¬ b ≤ a› have "(b - a) + a < a + b" by simp
      with 1 have "∃g. ggT (b - a) a g" by blast
      with ‹¬ b ≤ a› show ?thesis by (fastforce intro: SubRight)
    qed (auto intro: ZeroLeft)
  qed
qed
```

Dieses Lemma lässt sich per Induktion über die natürlichen Zahlen beweisen. Allerdings funktioniert es nicht, die Induktion einfach über `a` oder `b` zu machen.
Die Idee ist, zu zeigen, dass `ggT` eine Lösung für alle `a`,`b` hat, falls man weiß, dass `ggT` eine Lösung für alle `a`,`b` hat, deren Summe kleiner ist als `a + b`.
Sie können daher wahlweise ein Hilfslemma beweisen, welches die Aussage für alle `a`,`b` beweist, deren Summe kleiner als ein beliebiges `n` ist, oder Sie verwenden starke Induktion über die natürlichen Zahlen (Lemma `nat_less_induct: (⋀n. ∀m<n. ?P m ⟹ ?P n) ⟹ ?P ?n`) und führen die Induktion direkt über `a + b` (`induction "a + b" arbitrary: a b`).
Um das Lemma dann zu beweisen, wenden Sie Fallunterscheidung entsprechend der verschiedenen Fälle des Algorithmus an und zeigen Sie, wie man die Berechnung des `ggT` für `a`,`b` auf die Berechnung des `ggT` für geeignete kleinere `a'`,`b'` reduzieren kann.

Hinweise:

- Bei Hilfslemmas, die nur mit `div` und $\leq$ arbeiten, muss man explizit angeben, dass man auf den natürlichen Zahlen arbeitet. Dafür geben Sie einfach einer Variable explizit den Typ `nat`, z.B. `(a::nat) div b`.
- Übersicht über Lemmas, die Sie evt. brauchen könnten:

```
      dvdI:  ?a = ?b * ?k ⟹ ?b dvd ?a
      dvdE:  ?b dvd ?a ⟹ (⋀k. ?a = ?b * k ⟹ ?P) ⟹ ?P
  dvd_diffD: ?k dvd ?m - ?n ⟹ ?k dvd ?n ⟹ ?n ≤ ?m ⟹ ?k dvd ?m
 dvd_imp_le: ?k dvd ?n ⟹ 0 < ?n ⟹ ?k ≤ ?n
    dvd_def: (?b dvd ?a) = (∃k. ?a = ?b * k)
```

Bonusaufgabe:

Zeigen Sie, dass das Prädikat `ggT` rechtseindeutig ist. D.h. für gegebene `a` und `b` gibt es maximal ein `g`, so dass `ggT a b g` wahr ist. Oder umformuliert: Zeigen Sie, wenn `ggT a b g` und `ggT a b g'` gelten, so ist `g = g'`.

Hinweis: Verwenden Sie die auf diesem Blatt gezeigten Lemmas `ggT_divides` und `ggT_greatest`. Eventuell müssen Sie auch noch Hilfslemmas für die Basisfälle zeigen.

```
lemma ggT_inj:
  assumes "ggT a b g"
  and "ggT a b g'"
  shows "g = g'"
proof (cases "a = 0 ∧ b = 0")
  case True
  with ‹ggT a b g› have "g = 0" by (induction a b g rule: ggT.induct) auto
  also
  from ‹ggT a b g'› True have "0 = g'" by (induction a b g' rule: ggT.induct) auto
  finally show ?thesis .
next
  case False
  then have "a > 0 ∨ b > 0" by simp
  from ‹ggT a b g› have "g dvd a" and "g dvd b" by (blast dest: ggT_divides)+
  from ggT_divides [OF ‹ggT a b g'›] have "g' dvd a" and "g' dvd b" by simp_all
  from ‹ggT a b g› ‹a > 0 ∨ b > 0› ‹g' dvd a› ‹g' dvd b›
  have "g' ≤ g"
    by (rule ggT_greatest)
  moreover
  from ‹ggT a b g'› ‹a > 0 ∨ b > 0› ‹g dvd a› ‹g dvd b›
  have "g ≤ g'"
    by (rule ggT_greatest)
  ultimately show "g = g'" by simp
qed
```