	Universität Karlsruhe (TH) **Lehrstuhl für Programmierparadigmen** Theorembeweiser und ihre Anwendungen SS 2009 http://pp.info.uni-karlsruhe.de/ Übungsleiter: Daniel Wasserrab wasserra@ipd.info.uni-karlsruhe.de
	Übungsblatt 5 Besprechung: 26.05.2009

1. Regeln ohne Basisfall

Zeigen Sie, dass folgende Definition

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

ein leeres Prädikat definiert

lemma *evenempty_empty: "evenempty x ⟹ False"*
oops

2. Euklidischer Algorithmus - induktiv

Definieren Sie induktiv die Menge *ggT*, welche den größten gemeinsamen Teiler zweier natürlicher Zahlen beschreibt:

consts
  ggT :: "nat ⇒ nat ⇒ nat ⇒ bool"

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 durch Einzelschritte. Verwenden Sie dazu nur *simp* und die von ihnen oben definierten Introduktionsregeln. Wozu wird *?g* unifiziert?

lemma *"ggT 15 10 ?g"*
oops

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

lemma *"ggT 0 0 ?g"*
oops

Zeigen Sie, dass der ggT wirklich ein Teiler ist (Sie werden für den Beweis ein oder mehrere geeignete Hilfslemmas brauchen):

lemma *ggT_divides: "ggT a b g ⟹ g dvd a ∧ g dvd b"*
oops

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

lemma *ggT_greatest: "⟦ggT a b g; 0 < a ∨ 0 < b; d dvd a; d dvd b⟧ ⟹ d ≤ g"*
oops

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

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: "∀ a b. ∃ g. ggT a b g"*
oops

Das folgende Lemma, bewiesen durch starke Induktion über *n* (*nat_less_induct*), könnte hilfreich sein. Warum hilft die übliche Induktion über natürliche Zahlen hier nicht weiter?

thm *nat_less_induct*

lemma *ggT_defined_aux: "(a + b) ≤ n ⟹ ∃ g. ggT a b g"*
oops

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*.

Um dieses Lemma 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:

- Fallunterscheidung über Variablen, die ⋀-quantifiziert sind, mittels *case_tac* statt *case*.
- Bei Hilfslemmas, die mit *div* und ≤ arbeiten, muss man manchmal 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*.
- Taktik arith löst mathematische Aussagen, liefert auch ein Gegenbeispiel, falls sie subgoal nicht lösen kann
- Übersicht über Lemmas, die Sie brauchen könnten:
  add_le_imp_le_right: a + c ≤ b + c ⟹ a ≤ b
  add_le_mono1: i ≤ j ⟹ i + k ≤ j + k
  *dvd_def: (b dvd a) = (∃k. a = b * k)*
  *add_mult_distrib: (m + n) * k = m * k + n * k*
  *add_mult_distrib2: k * (m + n) = k * m + k * n*
  *diff_mult_distrib: (m - n) * k = m * k - n * k*
  *diff_mult_distrib2: k * (m - n) = k * m - k * n*