Isabelle/HOL Exercises
Projects

The Euclidean Algorithm – Inductively

Rules without base case

Show that the following

inductive_set *evenempty :: "nat set"* **where**
Add2Ie: "n ∈ evenempty ⟹ Suc(Suc n) ∈ evenempty"

defines the empty set:

lemma *evenempty_empty: "evenempty = {}"*

The Euclidean algorithm

Define inductively the set *gcd*, which characterizes the greatest common divisor of two natural numbers:

gcd :: "(nat × nat × nat) set"

Here, *(a,b,g) ∈ gcd* means that *g* is the gcd of a und b. The definition should closely follow the Euclidean algorithm.

Reminder: The Euclidean algorithm repeatedly subtracts the smaller from the larger number, until one of the numbers is 0. Then, the other number is the gcd.

Now, compute the gcd of 15 and 10:

lemma *"(15, 10, ?g) ∈ gcd"*

How does your algorithm behave on special cases as the following?

lemma *"(0, 0, ?g) ∈ gcd"*

Show that the gcd is really a divisor (for the proof, you need an appropriate lemma):

lemma *gcd_divides: "(a,b,g) ∈ gcd ⟹ g dvd a ∧ g dvd b"*

Show that the gcd is the greatest common divisor:

lemma *gcd_greatest [rule_format]: "(a,b,g) ∈ gcd ⟹*
0 < a ∨ 0 < b ⟶ (∀ d. d dvd a ⟶ d dvd b ⟶ d ≤ g)"

Here as well, you will have to prove a suitable lemma. What is the precondition `0 < a ∨ 0 < b` good for?

So far, we have only shown that `gcd` is correct, but your algorithm might not compute a result for all values `a,b`. Thus, show completeness of the algorithm:

```
lemma gcd_defined: "∀ a b. ∃ g. (a, b, g) ∈ gcd"
```

The following lemma, proved by course-of-value recursion over `n`, may be useful. Why does standard induction over natural numbers not work here?

```
lemma gcd_defined_aux [rule_format]:
  "∀ a b. (a + b) ≤ n ⟶ (∃ g. (a, b, g) ∈ gcd)"
  apply (induct rule: nat_less_induct)
  apply clarify
```

The idea is to show that `gcd` yields a result for all `a, b` whenever it is known that `gcd` yields a result for all `a', b'` whose sum is smaller than `a + b`.

In order to prove this lemma, make case distinctions corresponding to the different clauses of the algorithm, and show how to reduce computation of `gcd` for `a, b` to computation of `gcd` for suitable smaller `a', b'`.