lib/main/nat-gcd.ath

```
# Greatest common divisor of natural numbers, computed with
# Euclid's algorithm.

load "nat-div"

#......................................................................
extend-module N {

declare euclid: [N N] -> N  [[int->nat int->nat]]

module Euclid {


assert axioms :=
  (fun [(euclid x y) = [x                    when (y = zero)
                        (euclid y (x % y))  when (y =/= zero)]])
define [base reduction] := axioms

# Same axioms as:
# assert base := (forall x y . y = zero ==> (euclid x zero) = x)
# assert reduction :=
#   (forall x y . y =/= zero ==> (euclid x y) = (euclid y (x % y)))

define is-common-divisor :=
  lambda (z terms)
    match terms {
      [x y] => (z divides x & z divides y)
    }

#......................................................................
define common-divisor :=
  (forall y x . (euclid x y) is-common-divisor [x y])

define greatest :=
  (forall y x z . z is-common-divisor [x y] ==> z divides (euclid x y))

define correctness :=
  (forall x y . (euclid x y) is-common-divisor [x y] &
              forall z . z is-common-divisor [x y] ==>
                         z divides (euclid x y))

conclude goal := common-divisor
 (!strong-induction.principle goal
   method (y)
    assume IND-HYP := (strong-induction.hypothesis goal y)
      conclude (strong-induction.conclusion goal y)
        pick-any x
          conclude ((euclid x y) is-common-divisor [x y])
            (!two-cases
              assume A1 := (y = zero)
                let {C1 := (!chain [(euclid x y) = x  [base A1]]);
                     C2 :=
                      (!chain->
                       [true ==> (x divides x)  [divides.reflexive]
                             ==> ((euclid x y) divides x)  [C1]])}
                   (!chain->
                    [true ==> ((euclid x y) divides zero)
                                              [divides.right-zero]
                          ==> ((euclid x y) divides y)      [A1]
                          ==> (C2 & (euclid x y) divides y) [augment]])
              assume A2 := (y =/= zero)
                let {C1 :=
                      (!chain
                       [(euclid x y)
                        = (euclid y (x % y)) [reduction A2]]);
                     C2 :=
                      (!chain->
```

```
                       [(y =/= zero) ==> (zero < y)   [Less.zero<]]);
                     C3 :=
                      (!chain->
                       [C2
                    ==> (x % y < y)   [division-algorithm-corollary2]
                    ==> (forall ?x' .
                           (euclid ?x' (x % y))
                             is-common-divisor [?x' (x % y)])   [IND-HYP]
                    ==> ((euclid y (x % y))
                           is-common-divisor [y (x % y)]) [(specify [y])]
                    ==> ((euclid x y) is-common-divisor [y (x % y)])
                                                            [C1]]);
                     C4 := (!left-and C3)}
                  (!chain-> [(C2 & C3)
                         ==> ((euclid x y) divides x) [divides.first-lemma]
                         ==> ((euclid x y) divides x & C4)  [augment]])))

conclude goal := greatest
 (!strong-induction.principle goal
   method (y)
    assume IND-HYP := (strong-induction.hypothesis goal y)
      conclude (strong-induction.conclusion goal y)
        pick-any x z
          assume (z is-common-divisor [x y])
            conclude (z divides (euclid x y))
              (!two-cases
                assume A1 := (y = zero)
                  (!chain->
                   [(z divides x) ==> (z divides (euclid x y)) [A1 base]])
                assume A2 := (y =/= zero)
                  let {C1 :=
                        (!chain
                         [(euclid x y)
                          = (euclid y (x % y))    [A2 reduction]]);
                       C2 :=
                        (!chain->
                         [(y =/= zero) ==> (zero < y)   [Less.zero<]]);
                       C3 :=
                        (!chain->
                         [C2
                          ==> (z divides x & z divides y & C2)   [augment]
                          ==> (z divides x % y)    [divides.Mod-lemma]]);
                       C4 :=
                        (!chain->
                         [C2
                      ==> (x % y < y)
                                    [division-algorithm-corollary2]
                      ==> (forall ?x' ?z .
                            ?z is-common-divisor [?x' (x % y)] ==>
                            ?z divides (euclid ?x' (x % y)))   [IND-HYP]
                      ==> (z is-common-divisor [y (x % y)] ==>
                           z divides (euclid y (x % y)))
                                                      [(specify [y z])]])}
                  (!chain-> [(z divides y & C3)
                             ==> (z divides (euclid y (x % y))) [C4]
                             ==> (z divides (euclid x y))         [C1]])))

conclude correctness
  pick-any x y
    (!both (!chain-> [true ==> ((euclid x y) is-common-divisor [x y])
                                                   [common-divisor]])
           pick-any z
             (!chain [(z is-common-divisor [x y])
                      ==> (z divides (euclid x y)) [greatest]]))
} # Euclid
} # N
```