

lib/main/nat-gcd.ath

```

1 # Greatest common divisor of natural numbers, computed with
2 # Euclid's algorithm.
3
4 load "nat-div"
5
6 #.....
7 extend-module N {
8
9 declare euclid: [N N] -> N [[int->nat int->nat]]
10
11 module Euclid {
12
13
14 assert axioms :=
15   (fun [(euclid x y) = [x
16     (euclid y (x % y)) when (y /= zero)]]
17   define [base reduction] := axioms
18
19 # Same axioms as:
20 # assert base := (forall x y . y = zero ==> (euclid x zero) = x)
21 # assert reduction :=
22 #   (forall x y . y /= zero ==> (euclid x y) = (euclid y (x % y)))
23
24 define is-common-divisor :=
25   lambda (z terms)
26     match terms {
27       [x y] => (z divides x & z divides y)
28     }
29
30 #.....
31 define common-divisor :=
32   (forall y x . (euclid x y) is-common-divisor [x y])
33
34 define greatest :=
35   (forall y x z . z is-common-divisor [x y] ==> z divides (euclid x y))
36
37 define correctness :=
38   (forall x y . (euclid x y) is-common-divisor [x y] &
39     forall z . z is-common-divisor [x y] ==>
40       z divides (euclid x y))
41
42 conclude goal := common-divisor
43 (!strong-induction.principle goal
44   method (y)
45     assume IND-HYP := (strong-induction.hypothesis goal y)
46     conclude (strong-induction.conclusion goal y)
47     pick-any x
48     conclude ((euclid x y) is-common-divisor [x y])
49     (!two-cases
50       assume A1 := (y = zero)
51       let {C1 := (!chain [(euclid x y) = x [base A1]]);
52         C2 :=
53           (!chain->
54             [true ==> (x divides x) [divides.reflexive]
55               ==> ((euclid x y) divides x) [C1]])}
56       (!chain->
57         [true ==> ((euclid x y) divides zero)
58           [divides.right-zero]
59           ==> ((euclid x y) divides y) [A1]
60           ==> (C2 & (euclid x y) divides y) [augment]])
61       assume A2 := (y /= zero)
62       let {C1 :=
63         (!chain
64           [(euclid x y)
65             = (euclid y (x % y)) [reduction A2]]);
66         C2 :=
67         (!chain->

```

```

68         [(y /= zero) ==> (zero < y)   [Less.zero<]]);
69     C3 :=
70     (!chain->
71     [C2
72     ==> (x % y < y)   [division-algorithm-corollary2]
73     ==> (forall ?x' .
74         (euclid ?x' (x % y))
75         is-common-divisor [?x' (x % y)])   [IND-HYP]
76     ==> ((euclid y (x % y))
77         is-common-divisor [y (x % y)]) [(specify [y])]
78     ==> ((euclid x y) is-common-divisor [y (x % y)])
79         [C1]]);
80     C4 := (!left-and C3)
81     (!chain-> [(C2 & C3)
82         ==> ((euclid x y) divides x) [divides.first-lemma]
83         ==> ((euclid x y) divides x & C4) [augment]]))
84
85 conclude goal := greatest
86 (!strong-induction.principle goal
87 method (y)
88     assume IND-HYP := (strong-induction.hypothesis goal y)
89     conclude (strong-induction.conclusion goal y)
90     pick-any x z
91     assume (z is-common-divisor [x y])
92     conclude (z divides (euclid x y))
93     (!two-cases
94     assume A1 := (y = zero)
95     (!chain->
96     [(z divides x) ==> (z divides (euclid x y)) [A1 base]]
97     assume A2 := (y /= zero)
98     let {C1 :=
99         (!chain
100         [(euclid x y)
101         = (euclid y (x % y))   [A2 reduction]]);
102         C2 :=
103         (!chain->
104         [(y /= zero) ==> (zero < y)   [Less.zero<]]);
105         C3 :=
106         (!chain->
107         [C2
108         ==> (z divides x & z divides y & C2)   [augment]
109         ==> (z divides x % y)   [divides.Mod-lemma]]);
110         C4 :=
111         (!chain->
112         [C2
113         ==> (x % y < y)
114             [division-algorithm-corollary2]
115         ==> (forall ?x' ?z .
116             ?z is-common-divisor [?x' (x % y)] ==>
117             ?z divides (euclid ?x' (x % y)))   [IND-HYP]
118         ==> (z is-common-divisor [y (x % y)] ==>
119             z divides (euclid y (x % y)))
120             [(specify [y z])])])
121         (!chain-> [(z divides y & C3)
122             ==> (z divides (euclid y (x % y))) [C4]
123             ==> (z divides (euclid x y))   [C1]]))
124
125 conclude correctness
126 pick-any x y
127     (!both (!chain-> [true ==> ((euclid x y) is-common-divisor [x y])
128         [common-divisor]])
129     pick-any z
130         (!chain [(z is-common-divisor [x y])
131             ==> (z divides (euclid x y)) [greatest]]))
132 } # Euclid
133 } # N

```