

lib/main/nat-max.ath

```

1 # Properties of natural number Max function.
2
3 load "nat-less.ath"
4
5 extend-module N {
6 declare Max: [N N] -> N
7
8 module Max {
9 assert less2 := (forall ?x ?y . ?y < ?x ==> (Max ?x ?y) = ?x)
10 assert not-less2 :=
11   (forall ?x ?y . ~ (?y < ?x) ==> (Max ?x ?y) = ?y)
12 define commutative := (forall ?x ?y . (Max ?x ?y) = (Max ?y ?x))
13 define associative :=
14   (forall ?x ?y ?z . (Max (Max ?x ?y) ?z) = (Max ?x (Max ?y ?z)))
15
16 conclude commutative
17 pick-any x:N y
18   conclude ((Max x y) = (Max y x))
19     (!two-cases
20       assume (y < x)
21         let {_ :=
22           (!chain-> [(y < x) ==> (~ (x < y)) [Less.asymmetric]])}
23           (!chain [(Max x y)
24                 --> x [less2]
25                 <-- (Max y x) [not-less2]])
26         assume (~ y < x)
27           (!two-cases
28             assume (x = y)
29               (!chain [(Max x y)
30                     --> (Max y y) [(x = y)]
31                     <-- (Max y x) [(x = y)]])
32             assume (x /= y)
33               let {_ :=
34                 (!chain->
35                   [(x /= y)
36                   ==> (y /= x) [sym]
37                   ==> (~ y < x & y /= x) [augment]
38                   ==> (x < y) [Less.trichotomy]])}
39                 (!chain [(Max x y)
40                       --> y [not-less2]
41                       <-- (Max y x) [less2]])
42
43             )
44         )
45       )
46   )
47   pick-any x:N y:N z:N
48     (!two-cases
49       assume (y < x)
50         (!two-cases
51           assume (z < x)
52             let {e1 := (!chain [(Max (Max x y) z)
53                               --> (Max x z) [less2]
54                               --> x [less2]])};
55             e2 := conclude ((Max x (Max y z)) = x)
56               (!two-cases
57                 assume (z < y)
58                   (!chain [(Max x (Max y z))
59                           --> (Max x y) [less2]
60                           --> x [less2]])
61                 assume (~ z < y)
62                   (!chain [(Max x (Max y z))
63                           --> (Max x z) [not-less2]
64                           --> x [less2]])
65                 )
66             )
67           (!combine-equations e1 e2)
68         )
69       assume (~ z < x)
70         let {e3 := (!chain [(Max (Max x y) z)
71                           --> (Max x z) [less2]

```

```

69         --> z                               [not-less2]]);
70     _ := (!chain->
71         [ (~ z < x)
72         ==> (y < x & ~ z < x)                [augment]
73         ==> (y < z)                            [Less.transitive1]]);
74     e4 := conclude ((Max x (Max y z)) = z)
75         (!chain [(Max x (Max y z))
76                 --> (Max x (Max z y)) [commutative]
77                 --> (Max x z)         [less2]
78                 --> z                 [not-less2]]})
79     (!combine-equations e3 e4)
80 assume (~ (y < x))
81     (!two-cases
82     assume (y < z)
83     let {e5 := (!chain [(Max (Max x y) z)
84                       --> (Max y z)         [not-less2]
85                       --> (Max z y)         [commutative]
86                       --> z                 [less2]]);
87     _ := (!chain->
88         [(y < z)
89         ==> (~ y < x & y < z)                [augment]
90         ==> (x < z)                            [Less.transitive3]]);
91     e6 := conclude ((Max x (Max y z)) = z)
92         (!chain
93         [(Max x (Max y z))
94         --> (Max x (Max z y)) [commutative]
95         --> (Max x z)         [less2]
96         --> (Max z x)         [commutative]
97         --> z                 [less2]]})
98     (!combine-equations e5 e6)
99 assume (~ y < z)
100     (!two-cases
101     assume (z < x)
102     (!combine-equations
103     (!chain [(Max (Max x y) z)
104             --> (Max y z)         [not-less2]
105             --> (Max z y)         [commutative]
106             --> y                 [not-less2]]))
107     (!chain [(Max x (Max y z))
108             --> (Max x (Max z y)) [commutative]
109             --> (Max x y)         [not-less2]
110             --> y                 [not-less2]])))
111     assume (~ z < x)
112     (!combine-equations
113     (!chain [(Max (Max x y) z)
114             --> (Max y z)         [not-less2]
115             --> (Max z y)         [commutative]
116             --> y                 [not-less2]]))
117     (!chain [(Max x (Max y z))
118             --> (Max x (Max z y)) [commutative]
119             --> (Max x y)         [not-less2]
120             --> y                 [not-less2]]))))))
121
122 define idempotent := (forall ?x . (Max ?x ?x) = ?x)
123
124 conclude idempotent
125 pick-any x
126     (!chain-> [true ==> (~ (x < x))          [Less.irreflexive]
127              ==> ((Max x x) = x) [not-less2]])
128 } # Max
129 } # N

```