# Properties of natural number Max function.

load "nat-less.ath"

extend-module N {
  declare Max: [N N] -> N
}

module Max {
  assert not-less2 :=
    (forall ?x ?y . ~ (?y < ?x) ==> (Max ?x ?y) = ?y)
  define associative :=

  conclude commutative
  pick-any x:N y
    conclude ((Max x y) = (Max y x))
    {!two-cases
      assume [y < x]
      let {e := (!chain-> ![y < x] ==> ![x < y] [Less.asymmetric]])}
      (!chain [(Max x y) --> x] [less2]
       <=- (Max y x) [not-less2])
      assume ![y =/= x]
      let {!chain-> ![x =/= y] [sym]
        => ![y =/= x] [augment]
        => ![x < y] [Less.trichotomy]])
      (!chain [(Max x y) --> y] [not-less2]
       <=- (Max y x) [less2]))

  define associative :=
    (forall x y z . (Max (Max x y) z) = (Max x (Max y z)))
  conclude associative
  pick-any x:N y:N z:N
  {!two-cases
    assume ![y < x]
    {!two-cases
      assume ![z < x]
      let {e1 := (!chain ![Max (Max x y) z]
        --> (Max x z) [less2]
        --> x [less2]);
        e2 := conclude ((Max x (Max y z)) = x)}
      (!two-cases
        assume ![z < y]
        (!chain ![Max x (Max y z)]
         --> (Max x y) [less2]
         --> x [less2])
        assume ![z < y]
        (!chain ![Max x (Max y z)]
         --> (Max x z) [less2])
        --> x [less2])}
      (!combine-equations e1 e2)
    assume ![z < x]
    let {e3 := (!chain ![Max (Max x y) z]
      --> (Max x z) [less2]}}
---> z [not-less2]);
_ := (!chain->
   ["z < x"
   ===> (y < x & ~ z < x) [augment]
   ===> [y < z] [Less.transitive1]]);
e4 := conclude ((Max x (Max y z)) = z)
   (!chain ((Max x (Max y z))
              --> (Max x (Max z y)) [commutative]
              --> (Max x z) [less2]
              --> z [not-less2]))
   (!combine-equations e3 e4))
assume (~ (y < x))
(!two-cases
  assume (y < z)
  let {e5 := (!chain-
             [(Max (Max x y) z)
              --> (Max y z) [not-less2]
              --> (Max z y) [commutative]
              --> z [less2]]);
    _ := (!chain->
      [(y < z)
      ===> (~ y < x & y < z) [augment]
      ===> (x < z) [Less.transitive3]]);
    e6 := conclude ((Max x (Max y z)) = z)
    (!chain
      [Max x (Max y z)]
      --> (Max x (Max z y)) [commutative]
      --> (Max x z) [less2]
      --> (Max z x) [commutative]
      --> z [less2]]})
  (!combine-equations e5 e6)
assume (~ y < z)
(!two-cases
  assume (z < x)
  (!combine-equations
    (!chain [Max (Max x y) z]
      --> (Max y z) [not-less2]
      --> (Max z y) [commutative]
      --> y [not-less2]])
    (!chain [Max x (Max y z)]
      --> (Max x (Max z y)) [commutative]
      --> (Max x y) [not-less2]
      --> y [not-less2])))
assume (~ z < x)
(!combine-equations
  (!chain [Max (Max x y) z]
    --> (Max y z) [not-less2]
    --> (Max z y) [commutative]
    --> y [not-less2]])
  (!chain [Max x (Max y z)]
    --> (Max x (Max z y)) [commutative]
    --> (Max x y) [not-less2]
    --> y [not-less2])})
assume (~ x < y)
(!combine-equations
  (!chain [Max (Max x y) z]
    --> (Max x (Max y z)) [not-less2]
    --> (Max x y) [commutative]
    --> y [not-less2]))
  (!chain [Max x (Max y z)]
    --> (Max x (Max z y)) [commutative]
    --> (Max x y) [not-less2]
    --> y [not-less2]))})
define idempotent := (forall ?x . (Max ?x ?x) = ?x)
conclude idempotent
pick-any x
  (!chain-> [true ===> (~ {x < x}) [Less.irreflexive]
    ===> ((Max x x) = x) [not-less2])
| # Max
| # N