lib/main/nat-power.ath

```
# Properties of natural number exponentiation operator, Power.

load "nat-times"

#
# Exponentiation operator, **
#

extend-module N {
define [x y m n] := [?x:N ?y:N ?m:N ?n:N]
transform-output eval [nat->int]

open Times
declare **: [N N] -> N [400 [int->nat int->nat]]
module Power {

  assert* axioms := [(x ** zero = one)
                     (x ** S n = x * x ** n)]

  define [if-zero if-nonzero] := axioms

  (print "\n2 raised to the 3rd: " (eval 2 ** 3) "\n")

define Plus-case := (forall m n x . x ** (m + n) = x ** m * x ** n)
define left-one := (forall n . one ** n = one)
define right-one := (forall x . x ** one = x)
define right-two := (forall x . x ** two = x * x)
define left-times := (forall n x y . (x * y) ** n = x ** n * y ** n)
define right-times := (forall m n x . x ** (m * n) = (x ** m) ** n)
define two-case := (forall x . (square x) = x ** two)

by-induction Plus-case {
  zero =>
    conclude (forall n x . x ** (zero + n) = x ** zero * x ** n)
      pick-any n x
        (!chain
         [(x ** (zero + n))
      --> (x ** n)              [Plus.left-zero]
      <-- (one * x ** n)        [Times.left-one]
      <-- (x ** zero * x ** n)  [if-zero]])
| (m as (S m')) =>
  let {ind-hyp := (forall n x . x ** (m' + n) = x ** m' * x ** n)}
    conclude (forall n x . x ** (m + n) = x ** m * x ** n)
      pick-any n x
        (!combine-equations
          (!chain [(x ** ((S m') + n))
               --> (x ** (S (m' + n)))         [Plus.left-nonzero]
               --> (x * x ** (m' + n))         [if-nonzero]
               --> (x * (x ** m' * x ** n))    [ind-hyp]])
       (!chain [(x ** (S m') * x ** n)
            --> ((x * (x ** m')) * x ** n)     [if-nonzero]
            --> (x * (x ** m' * x ** n))       [Times.associative]]))
}


by-induction left-one {
  zero => (!chain [(one ** zero) --> one  [if-zero]])
| (S n) =>
  let {induction-hypothesis := (one ** n = one)}
  (!chain [(one ** (S n))
          --> (one * (one ** n)) [if-nonzero]
          --> (one ** n)         [Times.left-one]
          --> one                [induction-hypothesis]])
}

conclude right-one
  pick-any x:N
```

```
    (!chain [(x ** one)
             --> (x ** (S zero))    [one-definition]
             --> (x * x ** zero)    [if-nonzero]
             --> (x * one)          [if-zero]
             --> x                  [Times.right-one]])

} # close Power
} # close N
```