lib/main/nat-power.ath

# Properties of natural number exponentiation operator, Power.

load "nat-times"

# Exponentiation operator, **

extend-module N {


transform-output eval [nat->int]

open Times

module Power {

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

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

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 n x . x ** (two * n) = (x ** two) ** n)
define two-case := (forall x . (square x) = x ** two)

by-induction Plus-case {

zero =>

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]

| (S n) =>

let {induction-hypothesis := (one ** n = one)}

(!chain [(one ** (S n))
   -->> (one * (one ** n)) [if-nonzero]
   -->> one [Times.left-one]

| (S n) =>

let {induction-hypothesis := (one ** n = one)}

(!chain [(one ** (S n))
   -->> (one * (one ** n)) [if-nonzero]
   -->> one [Times.left-one]

conclude right-one

pick-any x:N

}
68 (\!chain \ [(x ** one)
69     \rightarrow (x ** (S \zerono))) \ [one-definition]
70     \rightarrow (x * x ** \zerono) \ [if-nonzero]
71     \rightarrow (x * one) \ [if-zero]
72     \rightarrow x \ [Times.right-one]))
73
74 | \# close Power
75 | \# close N