

## lib/main/nat-power0.ath

```

1  # Properties of natural number exponentiation operator, Power.
2
3  load "nat-times"
4
5  #
6  # Exponentiation operator, **
7  #
8
9  extend-module N {
10 open Times
11
12 declare **: [N N] -> N [400]
13
14 module Power {
15
16 define [x y m n] := [?x:N ?y:N ?m:N ?n:N]
17
18 assert axioms :=
19   (fun [(x ** zero) = one
20         (x ** (S n)) = (x * x ** n)])
21 define [if-zero if-nonzero] := axioms
22
23 define Plus-case := (forall m n x . x ** (m + n) = x ** m * x ** n)
24 define left-one := (forall n . one ** n = one)
25 define right-one := (forall x . x ** one = x)
26 define right-two := (forall x . x ** two = x * x)
27 define left-times :=
28   (forall n x y . (x * y) ** n = x ** n * y ** n)
29 define right-times :=
30   (forall m n x . x ** (m * n) = (x ** m) ** n)
31 define two-case := (forall x . square x = x ** two)
32
33 by-induction Plus-case {
34   zero =>
35     conclude (forall ?n ?x . x ** (zero + ?n) = ?x ** zero * ?x ** ?n)
36     pick-any n x
37       (!chain [(x ** (zero + n))
38                --> (x ** n) [Plus.left-zero]
39                <-- (one * x ** n) [Times.left-one]
40                <-- (x ** zero * x ** n) [if-zero]])
41 | (S m) =>
42   let {induction-hypothesis :=
43       (forall ?n ?x . ?x ** (m + ?n) = ?x ** m * ?x ** ?n)}
44   conclude (forall ?n ?x .
45     ?x ** ((S m) + ?n) = ?x ** (S m) * ?x ** ?n)
46   pick-any n x
47     (!combine-equations
48       (!chain
49         [(x ** ((S m) + n))
50          --> (x ** (S (m + n))) [Plus.left-nonzero]
51          --> (x * x ** (m + n)) [if-nonzero]
52          --> (x * (x ** m * x ** n)) [induction-hypothesis]])
53       (!chain
54         [(x ** (S m) * x ** n)
55          --> ((x * (x ** m)) * x ** n) [if-nonzero]
56          --> (x * (x ** m * x ** n)) [Times.associative]]))
57 }
58
59 by-induction left-times {
60   zero =>
61     conclude (forall ?x ?y . (?x * ?y) ** zero = ?x ** zero * ?y ** zero)
62     pick-any x y
63       (!chain [(x * y) ** zero)
64                --> one [if-zero]
65                <-- (one * one) [Times.right-one]
66                <-- (x ** zero * y ** zero) [if-zero]])
67 | (S n) =>

```

```

68   conclude (forall ?x ?y .
69     (?x * ?y) ** (S n) = ?x ** (S n) * ?y ** (S n))
70   let {induction-hypothesis :=
71     (forall ?x ?y . (?x * ?y) ** n = ?x ** n * ?y ** n)}
72   pick-any x y
73     (!combine-equations
74       (!chain
75         [((x * y) ** (S n))
76           --> ((x * y) * (x * y) ** n)      [if-nonzero]
77           --> ((x * y) * (x ** n * y ** n)) [induction-hypothesis]
78           --> (x * y * x ** n * y ** n)      [Times.associative]])
79       (!chain
80         [(x ** (S n) * y ** (S n))
81           --> ((x * (x ** n)) * y * y ** n) [if-nonzero]
82           --> (x * x ** n * y * y ** n)      [Times.associative]
83           <-- (x * (x ** n * y) * y ** n)      [Times.associative]
84           <-- (x * (y * x ** n) * y ** n)      [Times.commutative]
85           --> (x * y * x ** n * y ** n)      [Times.associative]]))
86 }
87
88 by-induction left-one {
89   zero => (!chain [(one ** zero) --> one [if-zero]])
90 | (S n) =>
91   let {induction-hypothesis := (one ** n = one)}
92     (!chain [(one ** (S n))
93       --> (one * (one ** n)) [if-nonzero]
94       --> (one ** n)         [Times.left-one]
95       --> one                 [induction-hypothesis]])
96 }
97
98 conclude right-one
99   pick-any x
100     (!chain [(x ** one)
101       --> (x ** (S zero))      [one-definition]
102       --> (x * x ** zero)      [if-nonzero]
103       --> (x * one)            [if-zero]
104       --> x                    [Times.right-one]])
105
106 conclude right-two
107   pick-any x
108     (!chain [(x ** two)
109       --> (x ** (S one))        [two-definition]
110       --> (x * x ** one)        [if-nonzero]
111       --> (x * x)                [right-one]])
112
113 by-induction right-times {
114   zero =>
115     conclude (forall ?n ?x . ?x ** (zero * ?n) = (?x ** zero) ** ?n)
116     pick-any n x
117       (!chain [(x ** (zero * n))
118         --> (x ** zero)          [Times.left-zero]
119         --> one                  [if-zero]
120         <-- (one ** n)            [left-one]
121         <-- ((x ** zero) ** n)    [if-zero]])
122 | (S m) =>
123   let {induction-hypothesis :=
124     (forall ?n ?x . ?x ** (m * ?n) = (?x ** m) ** ?n)}
125     conclude (forall ?n ?x . ?x ** ((S m) * ?n) = (?x ** (S m)) ** ?n)
126     pick-any n x
127       (!combine-equations
128         (!chain [(x ** ((S m) * n))
129           --> (x ** (n + m * n))      [Times.left-nonzero]
130           --> (x ** n * x ** (m * n)) [Plus-case]
131           --> (x ** n * ((x ** m) ** n)) [induction-hypothesis]])
132         (!chain [((x ** (S m)) ** n)
133           --> ((x * (x ** m)) ** n)      [if-nonzero]
134           --> ((x ** n) * (x ** m) ** n) [left-times]]))
135 }
136
137 conclude two-case

```

```
138 pick-any x
139   (!combine-equations
140     (!chain
141       [(square x) = (x * x) [square.definition]])
142     (!chain
143       [(x ** two)
144        = (x ** (S one)) [two-definition]
145        = (x ** (S (S zero))) [one-definition]
146        = (x * x ** (S zero)) [if-nonzero]
147        = (x * x * x ** zero) [if-nonzero]
148        = (x * x * one) [if-zero]
149        = (x * x) [Times.right-one]]))
150
151 } # Power
152 } # N
```