

lib/main/power.ath

```

1 # Abstract Power function
2
3 -----
4 load "nat-times"
5 load "group"
6 -----
7
8 extend-module Monoid {
9
10 declare +*: (S) [S N] -> S [400]
11
12 define [x y m n] := [?x:'T ?y:'T ?m:N ?n:N]
13
14 module Power {
15
16   define right-zero := (forall x . x +* zero = <0>)
17   define right-nonzero := (forall n x . x +* (S n) = x + x +* n)
18
19   (add-axioms theory [right-zero right-nonzero])
20
21   define [' + ' *'] := [N.+ N.*]
22
23   define right-plus :=
24     (forall m n x . x +* (m + ' n) = x +* m + x +* n)
25   define left-neutral := (forall n . <0> +* n = <0>)
26   define right-one := (forall x . x +* N.one = x)
27   define right-two := (forall x . x +* N.two = x + x)
28   define right-times :=
29     (forall n m x . x +* (m * ' n) = (x +* m) +* n)
30
31   define theorems :=
32     [right-plus left-neutral right-one right-two right-times]
33
34   define proofs :=
35     method (theorem adapt)
36       let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
37           [+ <0> +*] := (adapt [+ <0> +*])}
38       match theorem {
39         (val-of right-plus) =>
40           by-induction (adapt theorem) {
41             zero =>
42               conclude
43                 (forall ?n ?x . ?x +* (zero + ' ?n) = ?x +* zero + ?x +* ?n)
44               pick-any n x
45               (!chain
46                 [(x +* (zero + ' n))
47                  --> (x +* n) [N.Plus.left-zero]
48                  <-- (<0> + (x +* n)) [left-identity]
49                  <-- ((x +* zero) + (x +* n)) [right-zero]])
50               | (S m) =>
51                 let {ind-hyp :=
52                     (forall ?n ?x .
53                       ?x +* (m + ' ?n) = (?x +* m) + (?x +* ?n))}
54                 conclude
55                   (forall ?n ?x . ?x +* ((S m) + ' ?n) =
56                     ?x +* (S m) + ?x +* ?n)
57                 pick-any n x
58                 (!combine-equations
59                   (!chain
60                     [(x +* ((S m) + ' n))
61                      --> (x +* (S (m + ' n))) [N.Plus.left-nonzero]
62                      --> (x + (x +* (m + ' n))) [right-nonzero]
63                      --> (x + ((x +* m) + (x +* n))) [ind-hyp]])
64                   (!chain
65                     [((x +* (S m)) + (x +* n))
66                      --> ((x + (x +* m)) + (x +* n)) [right-nonzero]
67                      --> (x + ((x +* m) + (x +* n))) [associative]]))

```

```

68   }
69   | (val-of left-neutral) =>
70     by-induction (adapt theorem) {
71       zero => (!chain [(<0> +* zero) = <0>] [right-zero]])
72     | (S n) =>
73       let {ind-hyp := (<0> +* n = <0>)}
74         conclude (<0> +* (S n) = <0>)
75           (!chain [(<0> +* (S n))
76                   = (<0> + (<0> +* n)) [right-nonzero]
77                   = (<0> + <0>) [ind-hyp]
78                   = <0> [right-identity]])
79     }
80   | (val-of right-one) =>
81     pick-any x:(sort-of <0>)
82     (!chain [(x +* N.one)
83             --> (x +* (S zero)) [N.one-definition]
84             --> (x + x +* zero) [right-nonzero]
85             --> (x + <0>) [right-zero]
86             --> x [right-identity]])
87   | (val-of right-two) =>
88     let {right-one := (!prove right-one)}
89     pick-any x
90     (!chain [(x +* N.two)
91             = (x +* (S N.one)) [N.two-definition]
92             = (x + x +* N.one) [right-nonzero]
93             = (x + x) [right-one]])
94   | (val-of right-times) =>
95     by-induction (adapt theorem) {
96       zero =>
97         conclude
98           (forall ?m ?x . ?x +* (?m *' zero) = (?x +* ?m) +* zero)
99         pick-any m x
100        (!combine-equations
101         (!chain [(x +* (m *' zero))
102                 = (x +* zero) [N.Times.right-zero]
103                 = <0> [right-zero]])
104         (!chain [(x +* m) +* zero)
105                 = <0> [right-zero]]))
106     | (S n) =>
107       let {ind-hyp := (forall ?m ?x .
108                       ?x +* (?m *' n) = (?x +* ?m) +* n);
109           _ := (!prove right-plus);
110           _ := (!prove right-one)}
111         conclude
112           (forall ?m ?x .
113             ?x +* (?m *' (S n)) = (?x +* ?m) +* (S n))
114         pick-any m x
115         (!combine-equations
116         (!chain [(x +* (m *' (S n)))
117                 = (x +* (m *' n +' m)) [N.Times.right-nonzero]
118                 = (x +* (m *' n) + (x +* m)) [right-plus]
119                 = ((x +* m) +* n + (x +* m)) [ind-hyp]])
120         (!chain
121         [(x +* m) +* (S n)
122          = ((x +* m) + (x +* m) +* n) [right-nonzero]
123          = ((x +* m) +* N.one + (x +* m) +* n) [right-one]
124          = ((x +* m) +* (N.one +' n)) [right-plus]
125          = ((x +* m) +* (n +' N.one)) [N.Plus.commutative]
126          = ((x +* m) +* n + (x +* m) +* N.one) [right-plus]
127          = ((x +* m) +* n + x +* m) [right-one]]))
128     } # by-induction
129   } # match
130
131 (add-theorems theory |{theorems := proofs}|)
132 } # Power
133 } # Monoid
134
135 #-----
136 # The following theorem requires commutativity of +.
137

```

```

138 extend-module Abelian-Monoid {
139
140   define +* := Monoid.+*
141
142   define right-zero := Monoid.Power.right-zero
143
144   define right-nonzero := Monoid.Power.right-nonzero
145
146   define [x y n] := [?x:'T ?y:'T ?n:N]
147
148   define Power-left-times := (forall n x y . (x + y) +* n = x +* n + y +* n)
149
150 define proof :=
151 method (theorem adapt)
152   let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
153        [+ <0> +*] := (adapt [+ <0> +*])}
154   match theorem {
155     (val-of Power-left-times) =>
156     by-induction (adapt theorem) {
157       zero =>
158       pick-any x y
159         (!combine-equations
160           (!chain [((x + y) +* zero)
161                   = <0>                                [right-zero]])
162           (!chain [(x +* zero + y +* zero)
163                   = (<0> + <0>)                        [right-zero]
164                   = <0>                                [right-identity]]))
165       | (S n) =>
166       let {ind-hyp := (forall ?x ?y . (?x + ?y) +* n =
167                                   ?x +* n + ?y +* n)}
168         conclude (forall ?x ?y . (?x + ?y) +* (S n) =
169                   ?x +* (S n) + ?y +* (S n))
170         pick-any x y
171           (!chain
172             [((x + y) +* (S n))
173              = ((x + y) + (x + y) +* n)                [right-nonzero]
174              = ((x + y) + (x +* n) + (y +* n))        [ind-hyp]
175              = ((x + (x +* n)) + (y + (y +* n)))      [associative
176                                                         commutative]
177              = (x +* (S n) + y +* (S n))              [right-nonzero]])
178         }
179     }
180
181 (add-theorems theory |[Power-left-times] := proof|)
182 } # Abelian-Monoid

```