

lib/main/group.ath

```

1  ## Abstract algebraic theories: Semigroup, Identity, Monoid, Group
2
3  module Semigroup {
4    declare +: (S) [S S] -> S [200]
5    define associative := (forall x y z . (x + y) + z = x + (y + z))
6    define theory := (make-theory [] [associative])
7  }
8
9  module Identity {
10   open Semigroup
11   declare <0>: (S) [] -> S
12
13   define left-identity := (forall x . <0> + x = x)
14   define right-identity := (forall x . x + <0> = x)
15
16   define theory := (make-theory [] [left-identity right-identity])
17 }
18
19 module Monoid {
20   open Identity
21   define theory := (make-theory ['Semigroup 'Identity] [])
22 }
23
24 module Group {
25   open Monoid
26
27   declare U-: (S) [S] -> S # Unary minus
28   declare -: (S) [S S] -> S # Binary minus
29
30   define right-inverse := (forall x . x + U- x = <0>)
31   define minus-definition := (forall x y . x - y = x + U- y)
32
33   define theory :=
34     (make-theory ['Monoid] [right-inverse minus-definition])
35 }
36
37 extend-module Group {
38
39   define left-inverse := (forall x . (U- x) + x = <0>)
40   define double-negation := (forall x . U- U- x = x)
41   define unique-negation := (forall x y . x + y = <0> ==> U- x = y)
42   define neg-plus := (forall x y . U- (x + y) = (U- y) + (U- x))
43
44   define left-inverse-proof :=
45     method (theorem adapt)
46       let {[_ _ chain _ _] := (proof-tools adapt theory);
47           [+ U- <0>] := (adapt [+ U- <0>])}
48       conclude (adapt theorem)
49         pick-any x
50           (!chain
51             [((U- x) + x)
52              <-- ((U- x) + x) + <0>] [right-identity]
53              --> ((U- x) + (x + <0>)) [associative]
54              <-- ((U- x) + (x + ((U- x) + U- U- x))) [right-inverse]
55              <-- ((U- x) + ((x + U- x) + U- U- x)) [associative]
56              --> ((U- x) + <0> + U- U- x) [right-inverse]
57              <-- ((U- x) + <0>) + U- U- x [associative]
58              --> ((U- x) + U- U- x) [right-identity]
59              --> <0> [right-inverse]])
60
61   (add-theorems 'Group {| left-inverse := left-inverse-proof |})
62 }
63
64 extend-module Group {
65
66   define proofs :=
67     method (theorem adapt)

```

```

68 let [[get prove chain chain-> chain<-] := (proof-tools adapt theory);
69 [+ U- <0>] := (adapt [+ U- <0>])
70 match theorem {
71   (val-of double-negation) =>
72     conclude (adapt theorem)
73     pick-any x:(sort-of <0>)
74     (!chain [(U- (U- x))
75              <-- (<0> + (U- (U- x)))           [left-identity]
76              <-- ((x + (U- x)) + (U- (U- x))) [right-inverse]
77              --> (x + ((U- x) + (U- (U- x)))) [associative]
78              --> (x + <0>)                     [right-inverse]
79              --> x                               [right-identity]])
80   | (val-of unique-negation) =>
81     conclude (adapt theorem)
82     pick-any x:(sort-of <0>) y:(sort-of <0>)
83     let {LI := (!prove left-inverse)}
84     assume A := (x + y = <0>)
85     (!chain [(U- x)
86              <-- ((U- x) + <0>)                 [right-identity]
87              <-- ((U- x) + (x + y))             [A]
88              <-- (((U- x) + x) + y)            [associative]
89              --> (<0> + y)                     [LI]
90              --> y                             [left-identity]])
91   | (val-of neg-plus) =>
92     conclude (adapt theorem)
93     pick-any x y
94     let {UN := (!prove unique-negation);
95         A := (!chain
96              [((x + y) + ((U- y) + (U- x)))
97               <-- (x + ((y + (U- y)) + (U- x))) [associative]
98               --> (x + (<0> + (U- x)))         [right-inverse]
99               --> (x + (U- x))                 [left-identity]
100              --> <0>                           [right-inverse]])}
101     (!chain->
102      [A ==> (U- (x + y) = (U- y) + (U- x)) [UN]])
103   }
104
105 (add-theorems 'Group |{{double-negation unique-negation neg-plus} := proofs|})
106 }
107
108 module Abelian-Monoid {
109   open Monoid
110   define commutative := (forall x y . x + y = y + x)
111   define theory := (make-theory ['Monoid] [commutative])
112 }
113
114 module Abelian-Group {
115   open Group
116   define commutative := (forall x y . x + y = y + x)
117   define theory := (make-theory ['Group] [commutative])
118 }
119
120 # Commutativity allows a shorter proof for Left-Inverse and
121 # a more natural statement of Neg-Plus:
122
123 extend-module Abelian-Group {
124
125   define left-inverse-proof :=
126     method (theorem adapt)
127     let [[get prove chain chain-> chain<-] := (proof-tools adapt theory);
128         [+ U- <0>] := (adapt [+ U- <0>])]
129     conclude (adapt theorem)
130     pick-any x
131     (!chain [((U- x) + x)
132             --> (x + (U- x))           [commutative]
133             --> <0>                   [right-inverse]])
134
135 (add-theorems theory |{left-inverse := left-inverse-proof|})
136
137 define neg-plus := (forall x y . U- (x + y) = (U- x) + (U- y))

```

```

138
139 define neg-plus-proof :=
140   method (theorem adapt)
141     let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
142           [+ U- <0>] := (adapt [+ U- <0>])}]
143     conclude (adapt theorem)
144     pick-any x y
145       let {Group-version := (!prove-property Group.neg-plus
146                               adapt
147                               Group.theory)}
148           (!chain [(U- (x + y))
149                     --> ((U- y) + (U- x)) [Group-version]
150                     --> ((U- x) + (U- y)) [commutative]])]
151
152   (add-theorems theory |{neg-plus := neg-plus-proof}|)
153 } # close module Abelian-Group
154
155 #.....
156 # Multiplicative Semigroup, Monoid, and Group theories
157
158 module MSG {      # Multiplicative-Semigroup
159   declare *: (S) [S S] -> S [300]
160   define theory := (adapt-theory 'Semigroup |{Semigroup.+ := *}|)
161 }
162
163 module MM {      # Multiplicative-Monoid
164   declare <1>: (S) [] -> S
165   define theory :=
166     (adapt-theory 'Monoid |{Semigroup.+ := MSG.*, Monoid.<0> := <1>}|)
167 }
168
169 module MAM {    # Multiplicative-Abelian-Monoid
170   open MM
171
172   define theory :=
173     (adapt-theory 'Abelian-Monoid |{Semigroup.+ := MSG.*, Monoid.<0> := <1>}|)
174 }
175
176 module MG {    # Multiplicative-Group
177   declare inv: (T) [T] -> T
178   declare /: (T) [T T] -> T
179   define theory :=
180     (adapt-theory 'Group |{Semigroup.+ := MSG.*, Monoid.<0> := MM.<1>,
181                               Group.U- := inv, Group.- := /}|)
182 }

```