

lib/memory-range/swap-implementation.ath

```

1 load "memory"
2
3 extend-module Memory {
4   define t := ?t:(Loc 'S)
5
6   define swap-open-implementation :=
7     (forall M a b t M1 M2 M3 .
8       a /= t & b /= t &
9       M1 = M \ t <- (M at a) &
10      M2 = M1 \ a <- (M1 at b) &
11      M3 = M2 \ b <- (M2 at t)
12      ==> M3 = (M \ t <- (M at a)) \ (swap a b))
13
14   define swap-implementation :=
15     (forall M a b x M1 M2 .
16       x = (M at a) &
17       M1 = M \ a <- (M at b) &
18       M2 = M1 \ b <- x
19       ==> M2 = M \ (swap a b))
20 #-----
21 define proofs :=
22   method (theorem adapt)
23     let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
24         [at \ swap] := (adapt [at \ swap]);
25         [eq uneq] := [assign.equal assign.unequal]}
26     match theorem {
27       (val-of swap-open-implementation) =>
28       pick-any M:(Memory 'S) a:(Memory.Loc 'S) b:(Memory.Loc 'S)
29         t:(Memory.Loc 'S) M1:(Memory 'S) M2:(Memory 'S)
30         M3:(Memory 'S)
31       let {i := (M1 = M \ t <- (M at a));
32           ii := (M2 = M1 \ a <- (M1 at b));
33           iii := (M3 = M2 \ b <- (M2 at t))}
34       assume (a /= t & b /= t & i & ii & iii)
35       conclude (M3 = (M \ t <- (M at a)) \ (swap a b))
36       let {_ := (!sym (a /= t));
37           _ := (!sym (b /= t));
38           I := (!chain
39             [(M2 at t)
40              = ((M1 \ a <- (M1 at b)) at t)      [ii]
41              = (M1 at t)                          [uneq]
42              = ((M \ t <- (M at a)) at t)         [i]
43              = (M at a)                            [eq]]);
44           II := (!chain
45             [(M3 at a)
46              = ((M2 \ b <- (M2 at t)) at a)      [iii]
47              = ((M2 \ b <- (M at a)) at a)       [I]]);
48           III := conclude (M3 at a = M at b)
49             (!two-cases
50               assume (b = a)
51                 (!chain
52                   [(M3 at a)
53                    = ((M2 \ b <- (M at a)) at a) [II]
54                    = (M at a)                    [eq]
55                    = (M at b)                    [(b = a)]]
56                 assume (b /= a)
57                   (!chain
58                     [(M3 at a)
59                      = ((M2 \ b <- (M at a)) at a) [II]
60                      = (M2 at a)                    [uneq]
61                      = ((M1 \ a <- (M1 at b)) at a) [ii]
62                      = (M1 at b)                    [eq]
63                      = ((M \ t <- (M at a)) at b)   [i]
64                      = (M at b)                    [uneq]]));
65           IV := pick-any u
66                 conclude (M3 at u =
67                   ((M \ t <- (M at a)) \ (swap a b)) at u)

```

```

68         (!three-cases
69           assume (a = u)
70             (!combine-equations
71               (!chain
72                 [(M3 at u)
73                  = (M3 at a) [a = u]
74                  = (M at b) [III]
75                  = ((M \ t <- (M at a)) at b) [uneq]])
76               (!chain
77                 [(((M \ t <- (M at a)) \ (swap a b)) at u)
78                  = (((M \ t <- (M at a)) \ (swap a b)) at a)
79                    [(a = u)]
80                  = ((M \ t <- (M at a)) at b) [swap.equal1]]))
81           assume (b = u)
82             (!combine-equations
83               (!chain
84                 [(M3 at u)
85                  = (M3 at b) [(b = u)]
86                  = ((M2 \ b <- (M2 at t)) at b) [iii]
87                  = (M2 at t) [eq]
88                  = (M at a) [I]
89                  = ((M \ t <- (M at a)) at a) [uneq]])
90               (!chain
91                 [(((M \ t <- (M at a)) \ (swap a b)) at u)
92                  = (((M \ t <- (M at a)) \ (swap a b)) at b)
93                    [(b = u)]
94                  = ((M \ t <- (M at a)) at a) [swap.equal2]]))
95           assume (a /= u & b /= u)
96             (!combine-equations
97               (!chain
98                 [(M3 at u)
99                  = ((M2 \ b <- (M2 at t)) at u) [iii]
100                 = (M2 at u) [uneq]
101                 = ((M1 \ a <- (M1 at b)) at u) [ii]
102                 = (M1 at u) [uneq]
103                 = ((M \ t <- (M at a)) at u) [i]])
104               (!chain
105                 [(((M \ t <- (M at a)) \ (swap a b)) at u)
106                  = ((M \ t <- (M at a)) at u) [swap.unequal]]))
107             (!chain
108               [M3 = ((M \ t <- (M at a)) \ (swap a b)) [equality]])
109 | (val-of swap-implementation) =>
110 pick-any M:(Memory 'S) a:(Memory.Loc 'S) b:(Memory.Loc 'S) x:'S
111         M1:(Memory 'S) M2:(Memory 'S)
112 let {i := (x = (M at a));
113       ii := (M1 = M \ a <- (M at b));
114       iii := (M2 = M1 \ b <- x)}
115 assume (i & ii & iii)
116 conclude (M2 = M \ (swap a b))
117 let {I := (!chain
118           [(M2 at a)
119            = ((M1 \ b <- x) at a) [iii]
120            = ((M1 \ b <- (M at a)) at a) [i]]);
121       II := conclude (M2 at a = M at b)
122           (!two-cases
123             assume (b = a)
124               (!chain
125                 [(M2 at a)
126                  = ((M1 \ b <- (M at a)) at a) [I]
127                  = (M at a) [eq]
128                  = (M at b) [(b = a)]]
129             assume (b /= a)
130               (!chain
131                 [(M2 at a)
132                  = ((M1 \ b <- (M at a)) at a) [I]
133                  = (M1 at a) [uneq]
134                  = ((M \ a <- (M at b)) at a) [ii]
135                  = (M at b) [eq]]));
136       III :=
137 pick-any u

```

```

138     conclude (M2 at u = (M \ (swap a b)) at u)
139     (!three-cases
140     assume (a = u)
141     (!combine-equations
142     (!chain
143     [(M2 at u)
144     = (M2 at a) [ (a = u) ]
145     = (M at b) [II]])
146     (!chain
147     [( (M \ (swap a b)) at u)
148     = ((M \ (swap a b)) at a) [ (a = u) ]
149     = (M at b) [swap.equal1]))
150     assume (b = u)
151     (!combine-equations
152     (!chain
153     [(M2 at u)
154     = (M2 at b) [ (b = u) ]
155     = ((M1 \ b <- x) at b) [iii]
156     = x [eq]
157     = (M at a) [i]])
158     (!chain
159     [( (M \ (swap a b)) at u)
160     = ((M \ (swap a b)) at b) [ (b = u) ]
161     = (M at a) [swap.equal2]))
162     assume (a /= u & b /= u)
163     (!combine-equations
164     (!chain
165     [(M2 at u)
166     = ((M1 \ b <- x) at u) [iii]
167     = (M1 at u) [uneq]
168     = ((M \ a <- (M at b)) at u) [ii]
169     = (M at u) [uneq]))
170     (!chain
171     [( (M \ (swap a b)) at u)
172     = (M at u) [swap.unequal]])
173     (!chain [M2 = (M \ (swap a b)) [equality]])
174     }
175
176 (add-theorems theory
177   |{{swap-open-implementation swap-implementation} := proofs})
178 } # Memory

```