

lib/memory-range/memory.ath

```

1  ### Memory theory
2
3  #-----
4
5  domain (Memory S)
6
7  module Memory {
8    domain (Change S)
9    domain (Loc S)
10
11   declare \: (S) [(Memory S) (Change S)] -> (Memory S)
12   declare \\<: (S, T) [(Memory S) T] -> T
13   declare at: (S) [(Memory S) (Loc S)] -> S
14
15   define [M M1 M2 M3 a b c x y] :=
16     [?M:(Memory 'S) ?M1:(Memory 'S) ?M2:(Memory 'S)
17      ?M3:(Memory 'S) ?a:(Loc 'S) ?b:(Loc 'S) ?c:(Loc 'S)
18      ?x:'S ?y:'S]
19
20   define equality :=
21     (forall M1 M2 . (forall a . M1 at a = M2 at a) <==> M1 = M2)
22
23   declare <-: (S) [(Loc S) S] -> (Change S)
24
25   module assign {
26     define axioms :=
27       (fun [(M \ a <- x) at b] =
28         [x           when (a = b)
29          (M at b)    when (a /= b)])
30     define [equal unequal] := axioms
31   }
32
33   define theory := (make-theory [] [equality assign.equal assign.unequal])
34
35  #-----
36  declare swap: (S) [(Loc S) (Loc S)] -> (Change S)
37
38  module swap {
39
40    define axioms :=
41      (fun [(M \ (swap a b)) at c] =
42        [(M at b)           when (a = c)
43         (M at a)           when (b = c)
44         (M at c)           when (a /= c & b /= c)])
45    define [equal1 equal2 unequal] := axioms
46
47    (add-axioms theory axioms)
48  }
49
50  #-----
51  # Theorems:
52
53  define Double-assign :=
54    (forall b M a x y . ((M \ a <- x) \ a <- y) at b = (M \ a <- y) at b)
55  define Direct-double-assign :=
56    (forall M a x y . (M \ a <- x) \ a <- y = M \ a <- y)
57  define Self-assign :=
58    (forall M a b . (M \ a <- M at a) at b = M at b)
59  define Direct-self-assign := (forall M a . M \ a <- M at a = M)
60  define Double-swap :=
61    (forall c M a b .
62      ((M \ (swap a b)) \ (swap b a)) at c = M at c)
63  define Direct-double-swap :=
64    (forall M a b . (M \ (swap a b)) \ (swap b a) = M)
65
66  define theorems := [Double-assign Direct-double-assign Self-assign
67                     Direct-self-assign Double-swap Direct-double-swap]

```

```

68 define proofs :=
69   method (theorem adapt)
70   let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
71       [at \ <- swap] := (adapt [at \ <- swap])}]
72   match theorem {
73     (val-of Double-assign) =>
74     pick-any b:(Loc 'S) M:(Memory 'S) a:(Loc 'S) x:'S y:'S
75     (!two-cases
76       assume (a = b)
77       (!chain [(((M \ a <- x) \ a <- y) at b)
78         <-- ((M \ a <- x) \ a <- y) at a] [(a = b)]
79         --> y [assign.equal]
80         <-- ((M \ a <- y) at a) [assign.equal]
81         --> ((M \ a <- y) at b) [(a = b)]])
82       assume (a /= b)
83       (!chain [(((M \ a <- x) \ a <- y) at b)
84         --> ((M \ a <- x) at b) [assign.unequal]
85         --> (M at b) [assign.unequal]
86         <-- ((M \ a <- y) at b) [assign.unequal]])
87 | (val-of Direct-double-assign) =>
88   pick-any M:(Memory 'S) i:(Loc 'S) x:'S y:'S
89   let {DA := (!prove Double-assign);
90       A := pick-any a:(Loc 'S)
91         (!chain [(((M \ i <- x) \ i <- y) at a)
92           --> ((M \ i <- y) at a) [DA]])}
93   (!chain->
94     [A <==> ((M \ i <- x) \ i <- y = M \ i <- y)
95       [equality]])
96 | (val-of Self-assign) =>
97   pick-any M:(Memory 'S) a:(Loc 'S) b:(Loc 'S)
98   let {goal := ((M \ a <- (M at a)) at b = M at b)}
99   (!two-cases
100     assume (a = b)
101     (!chain [((M \ a <- (M at a)) at b)
102       <-- ((M \ a <- (M at a)) at a) [(a = b)]
103       --> (M at a) [assign.equal]
104       --> (M at b) [(a = b)]])
105     (!chain [a =/= b ==> goal [assign.unequal]])
106 | (val-of Direct-self-assign) =>
107   pick-any M:(Memory 'S) i:(Loc 'S)
108   let {SA := (!prove Self-assign);
109       A := pick-any a:(Loc 'S)
110         (!chain [((M \ i <- (M at i)) at a)
111           --> (M at a) [SA]])}
112   (!chain->
113     [A ==> ((M \ i <- (M at i)) = M) [equality]])
114 | (val-of Double-swap) =>
115   pick-any c:(Loc 'S) M:(Memory 'S) a:(Loc 'S) b:(Loc 'S)
116   (!three-cases
117     assume (a = c)
118     (!chain [(((M \ (swap a b)) \ (swap b a)) at c)
119       <-- ((M \ (swap a b)) \ (swap b a)) at a [(a = c)]
120       --> ((M \ (swap a b)) at b) [swap.equal2]
121       --> (M at a) [swap.equal2]
122       --> (M at c) [(a = c)]])
123     assume (b = c)
124     (!chain [(((M \ (swap a b)) \ (swap b a)) at c)
125       <-- ((M \ (swap a b)) \ (swap b a)) at b [(b = c)]
126       --> ((M \ (swap a b)) at a) [swap.equal1]
127       --> (M at b) [swap.equal1]
128       --> (M at c) [(b = c)]])
129     assume (a /= c & b /= c)
130     (!chain [(((M \ (swap a b)) \ (swap b a)) at c)
131       --> ((M \ (swap a b)) at c) [swap.unequal]
132       --> (M at c) [swap.unequal]])
133 | (val-of Direct-double-swap) =>
134   pick-any M:(Memory 'S) a:(Loc 'S) b:(Loc 'S)
135   let {DS := (!prove Double-swap);
136       A := pick-any c:(Loc 'S)
137         (!chain [(((M \ (swap a b)) \ (swap b a)) at c)

```

```
138         --> (M at c) [DS]])}
139     (!chain-> [A ==> ((M \ (swap a b)) \ (swap b a)) = M)
140               [equality])
141   }
142
143 (add-theorems theory |{theorems := proofs}|)
144
145 } # Memory
```