

lib/memory-range/temp.ath

```

1  define proof :=
2  method (theorem adapt)
3    let {[get prove chain chain-> chain<-] := (proof-tools adapt Theory);
4        deref := (adapt deref)}
5    match theorem {
6      (val-of correctness) =>
7      by-induction (adapt theorem) {
8        (stop q:(It 'X 'S)) =>
9        pick-any i:(It 'X 'S) j:(It 'X 'S)
10           M:(Memory 'S) k:(It 'Y 'S)
11           M':(Memory 'S) k':(It 'Y 'S)
12        assume A := ((limits i j) = SOME stop q &
13                     ~ k *in stop q &
14                     M' = M \ (copy-memory i j k) &
15                     k' = M \ (copy i j k))
16        conclude goal :=
17          (exists ?r' .
18            (limits k k') = SOME ?r' &
19            (collect M' ?r') = (collect M stop q) &
20            forall ?h . ~ ?h *in ?r' ==>
21              M' at deref ?h = M at deref ?h)
22        let {EL1 := (!prove empty-limits1);
23            _ := conclude (i = j)
24                  (!chain-> [(limits i j)
25                             = (SOME stop q) [A]
26                             ==> (i = j) [EL1]]);
27            _ := conclude (M' = M)
28                  (!chain->
29                    [M' = (M \ (copy-memory i j k)) [A]
30                     = M [copy-memory.empty]]);
31            _ := conclude (k' = k)
32                  (!chain->
33                    [k' = (M \ (copy i j k)) [A]
34                     = k [empty]]);
35            _ := (!chain [(start stop k)
36                           = k [start.of-stop]
37                           = (finish stop k) [finish.of-stop]]);
38            protected := pick-any h
39                          assume (~ h *in stop k)
40                          (!chain
41                            [(M' at deref h)
42                             = (M at deref h) [(M' = M)]]);
43            EL := (!prove empty-limits);
44            B := (!both
45                  (!chain [(limits k k')
46                           = (limits k k) [(k' = k)]
47                           = (SOME stop k) [EL]])
48                  (!both (!chain
49                          [(collect M' stop k)
50                           = nil:(List 'S) [collect.of-stop]
51                           = (collect M stop q) [collect.of-stop]]
52                          protected)))
53            (!chain-> [B ==> goal [existence]]))
54    | (back r:(Range 'X 'S)) =>
55      let {ind-hyp :=
56        (forall ?i:(It 'X 'S) ?j:(It 'X 'S)
57          ?M:(Memory 'S) ?k:(It 'Y 'S)
58          ?M':(Memory 'S) ?k':(It 'Y 'S) .
59          (limits ?i ?j) = SOME r &
60          ~ ?k *in r &
61          ?M' = ?M \ (copy-memory ?i ?j ?k) &
62          ?k' = ?M \ (copy ?i ?j ?k)
63          ==> exists ?r' .
64            (limits ?k ?k') = SOME ?r' &
65            (collect ?M' ?r') = (collect ?M r) &
66            forall ?h . ~ ?h *in ?r' ==>
67              ?M' at deref ?h = ?M at deref ?h)}

```

```

68   pick-any i:(It 'X 'S) j:(It 'X 'S) M:(Memory 'S) k:(It 'Y 'S)
69         M':(Memory 'S) k':(It 'Y 'S)
70   let {M1 := (M \ deref k <- M at deref i);
71       A1 := ((limits i j) = SOME back r);
72       A2 := (~ k *in back r);
73       A3 := (M' = M \ (copy-memory i j k));
74       A4 := (k' = M \\ (copy i j k))}
75   assume (A1 & A2 & A3 & A4)
76   conclude
77     goal := (exists ?r' .
78             (limits k k') = SOME ?r' &
79             (collect M' ?r') = (collect M back r) &
80             forall ?h . ~ ?h *in ?r' ==>
81               M' at deref ?h = M at deref ?h)
82
83   let {(and B1 B2) :=
84         (!chain->
85          [(limits i j)
86           = (SOME back r) [A1]
87           = (limits (start back r) (finish back r))
88             [limits.collapse]
89           ==> (i = (start back r) & j = (finish back r))
90             [limits.injective]]);
91         _ := (!chain->
92              [true
93               ==> ((start back r) /= (finish back r))
94                 [finish.nonempty-back]
95               ==> (i /= j) [B1 B2]]);
96         RR := (!prove *in.range-reduce);
97         CU := (!prove collect.unchanged);
98         B3 := (!chain-> [A2 ==> (~ k *in r) [RR]
99                        ==> ((collect M1 r) =
100                          (collect M r)) [CU]]);
101         B4 := conclude (M' = (M1 \ (copy-memory
102                                (successor i) j
103                                (successor k))))
104           (!chain
105            [M' = (M \ (copy-memory i j k)) [A3]
106             = (M1 \ (copy-memory (successor i) j
107                                (successor k)))
108              [copy-memory.nonempty]]);
109         B5 := conclude (k' = (M1 \\ (copy (successor i) j
110                                           (successor k))))
111           (!chain
112            [k' = (M \\ (copy i j k)) [A4]
113             = (M1 \\ (copy (successor i) j
114                           (successor k)))
115              [nonempty]]);
116         LB := (!prove limits-back);
117         A1' := (!chain->
118                [A1 ==> ((limits (successor i) j) =
119                        SOME r) [LB]]);
120         RS2 := (!prove *in.range-shift2);
121         B6 := (!chain->
122                [A2
123                 ==> (~ (successor k) *in r) [RS2]
124                 ==> (A1' & ~ (successor k) *in r & B4 & B5)
125                     [augment]
126                 ==> (exists ?r' .
127                     (limits (successor k) k') = SOME ?r' &
128                     (collect M' ?r') = (collect M1 r) &
129                     forall ?h . ~ ?h *in ?r' ==>
130                       M' at deref ?h =
131                       M1 at deref ?h) [ind-hyp]]);
132   pick-witness r'' for B6 B6-w
133   (!force goal)
134   } # by-induction
135   } # match theorem
136 (evolve Theory [[correctness] proof])
137 } # copy

```

```
138 } # Forward-Iterator
```