

lib/memory-range/copy-range-backward.ath

```

1 load "copy-range"
2 load "reverse-range"
3 #.....
4
5 extend-module Bidirectional-Iterator {
6   open Reversing
7
8   declare copy-memory-backward: (S, X, Y) [(It X S) (It X S) (It Y S)] ->
9     (Change S)
10  declare copy-backward: (S, X, Y) [(It X S) (It X S) (It Y S)] -> (It Y S)
11
12  module copy-memory-backward {
13    define [def] :=
14      (fun
15        [(M \ (copy-memory-backward i j k)) =
16          (M \ (copy-memory (reverse-iterator j)
17            (reverse-iterator i)
18              (reverse-iterator k)))]])
19  }
20
21  module copy-backward {
22    define [def] :=
23      (fun
24        [(M \ \ (copy-backward i j k)) =
25          (base-iterator (M \ \ (copy (reverse-iterator j)
26            (reverse-iterator i)
27              (reverse-iterator k)))]])
28  }
29
30  (add-axioms theory [copy-memory-backward.def
31    copy-backward.def])
32
33 #.....
34
35 extend-module copy-backward {
36  define [r r'] := [?r:(Range 'X 'S) ?r':(Range 'Y 'S)]
37
38  define correctness :=
39    (forall r i j M k M' k' .
40      (range i j) = (SOME r) &
41      ~ (predecessor k) *in r &
42      M' = (M \ (copy-memory-backward i j k)) &
43      k' = (M \ \ (copy-backward i j k))
44      ==> exists r' .
45        (range k' k) = SOME r' &
46        (collect M' r') = (collect M r) &
47        forall h . ~ h *in r' ==> M' at deref h = M at deref h)
48
49  define proof :=
50    method (theorem adapt)
51      let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
52          [deref *in successor predecessor] :=
53            (adapt [deref *in successor predecessor])}]
54      match theorem {
55        (val-of correctness) =>
56          pick-any r:(Range 'X 'S) 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          let {A1 := ((range i j) = SOME r);
60              A2 := (~ (predecessor k) *in r);
61              A3 := (M' = (M \ (copy-memory-backward i j k)));
62              A4 := (k' = (M \ \ (copy-backward i j k)));
63              goal := (exists r' .
64                (range 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   assume (A1 & A2 & A3 & A4)
69   let {ri := reverse-iterator;
70       RLR := (!prove reverse-range-reverse);
71       B1 := (!chain->
72             [A1 ==> ((range (ri j) (ri i)) =
73                     SOME reverse-range r)      [RLR]])];
74       B2 := (!chain->
75             [A2
76             ==> (~ (ri k) *in reverse-range r)   [*reverse-in]]);
77       B3 := (!chain
78             [M'
79             = (M \ (copy-memory-backward i j k)) [A3]
80             = (M \ (copy-memory (ri j) (ri i) (ri k)))
81                   [copy-memory-backward.def]]);
82       B4 := (!chain
83             [(ri k')
84             = (ri (M \ (copy-backward i j k))) [A4]
85             = (ri (base-iterator
86                   (M \ (copy (ri j) (ri i) (ri k))))
87                   [copy-backward.def]
88             = (M \ (copy (ri j) (ri i) (ri k))) [reverse-base]]);
89       CC := (!prove copy.correctness);
90       B5 := (!chain->
91             [(B1 & B2 & B3 & B4)
92             ==> (exists r' .
93                 (range (ri k) (ri k')) = SOME r' &
94                 (collect M' r') = (collect M reverse-range r) &
95                 forall h .
96                   ~ h *in r' ==>
97                     M' at deref h = M at deref h) [CC]])
98   pick-witness r' for B5
99   let {B5-w1 := ((range (ri k) (ri k')) = SOME r');
100       B5-w2 := ((collect M' r') =
101                (collect M (reverse-range r)));
102       B5-w3 := (forall h .
103                ~ h *in r' ==>
104                M' at deref h = M at deref h);
105       C1 := (!chain->
106             [B5-w1
107             ==> ((range k' k) = SOME base-range r')
108                   [reverse-of-range]]);
109       CRC := (!prove collect-reverse-corollary);
110       C2 := (!chain->
111             [B5-w2
112             ==> ((collect M' base-range r') = (collect M r))
113                   [CRC]]);
114       C3 :=
115       conclude (forall h . ~ h *in base-range r' ==>
116                M' at deref h = M at deref h)
117   pick-any h
118   (!chain
119     [(~ h *in base-range r')
120     ==> (~ predecessor successor h *in
121         base-range r') [predecessor.of-successor]
122     ==> (~ (reverse-iterator successor h) *in
123         reverse-range base-range r')
124         [*reverse-in]
125     ==> (~ (reverse-iterator successor h) *in r')
126         [reverse-base-range]
127     ==> (M' at deref reverse-iterator successor h =
128         M at deref reverse-iterator successor h)
129         [B5-w3]
130     ==> (M' at deref predecessor successor h =
131         M at deref predecessor successor h)
132         [deref-reverse]
133     ==> (M' at deref h = M at deref h)
134         [predecessor.of-successor])])
135   (!chain->
136     [(C1 & C2 & C3) ==> goal      [existence]])
137 }

```

```
138  
139   (add-theorems theory |[correctness] := proof)|)  
140 } # close module copy-backward  
141 } # close module Bidirectional-Iterator
```