

## lib/memory-range/ordered-range.ath

```

1 load "ordered-list"
2 load "range-length"
3
4 #-----
5 # <ER: is an S value < or E the first element of a range of S
6 # values (true if the range is empty)
7
8 extend-module SWO {
9   define deref := Trivial-Iterator.deref
10  define start := Range.start
11  define at := Memory.at
12
13  declare <ER: (S, X) [S (Range X S)] -> Boolean
14
15  define [M r i x] :=
16    [?M:(Memory 'S) ?r:(Range 'X 'S) ?i:(It 'X 'S) ?x:'S]
17
18  module <ER {
19
20    define empty := (forall x i . x <ER stop i)
21
22    define nonempty :=
23      (forall x M r . x <ER back r <==> x <E M at deref start back r)
24
25    (add-axioms theory [empty nonempty])
26  }
27 }
28
29 #-----
30 module Ordered-Range {
31   open SWO, Random-Access-Iterator
32
33   declare ordered: (S, X) [(Memory S) (Range X S)] -> Boolean
34
35   define ordered' := SWO.ordered
36
37   define def := (forall r M . (ordered M r) <==> ordered' (collect M r))
38
39   define theory := (make-theory ['SWO 'Random-Access-Iterator] [def])
40
41   define ordered-rest-range :=
42     (forall M r . (ordered M back r) ==> (ordered M r))
43
44   define ordered-empty-range := (forall M i . (ordered M stop i))
45
46   define ordered-subranges :=
47     (forall M r i j n . (range i j) = SOME r &
48       (ordered M r) &
49       n <= length r
50       ==> exists r' r'' .
51         (range i i + n) = SOME r' &
52         (range i + n j) = SOME r'' &
53         (ordered M r') &
54         (ordered M r''))
55
56   define proofs :=
57     method (theorem adapt)
58       let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
59         [deref <EL I+N I-N I-I ordered ordered'] :=
60           (adapt [deref <EL I+N I-N I-I ordered ordered'])}
61     match theorem {
62       (val-of ordered-rest-range) =>
63       pick-any M r
64         (!chain [(ordered M (back r))
65           ==> (ordered' (collect M (back r))) [def]
66           ==> (ordered' ((M at deref start back r)
67             :: (collect M r))) [collect.of-back]

```

```

68     ==> ((M at deref start back r) <EL (collect M r) &
69         (ordered' (collect M r))) [SWO.ordered.nonempty]
70     ==> (ordered' (collect M r)) [right-and]
71     ==> (ordered M r) [def]])
72 | (val-of ordered-empty-range) =>
73   pick-any M i
74   (!chain->
75     [true ==> (ordered' nil) [SWO.ordered.empty]
76       ==> (ordered' (collect M stop i)) [collect.of-stop]
77       ==> (ordered M stop i) [def]])
78 | (val-of ordered-subranges) =>
79   pick-any M r:(Range 'X 'S) i:(It 'X 'S) j:(It 'X 'S) n
80   let {A1 := ((range i j) = SOME r);
81       A2 := (ordered M r);
82       A3 := (n <= length r)}
83   assume (A1 & A2 & A3)
84   let {goal := (exists r' r'' .
85               (range i i + n) = SOME r' &
86               (range i + n j) = SOME r'' &
87               (ordered M r') &
88               (ordered M r''))};
89   CSR := (!prove collect-split-range);
90   B1 := (!chain->
91     [(A1 & A3)
92      ==> (exists r' r'' .
93          (range i i + n) = SOME r' &
94          (range i + n j) = SOME r'' &
95          (forall M .
96            (collect M r) =
97            (collect M r') join (collect M r'')))]
98     [CSR]))
99   pick-witnesses r' r'' for B1 B1-w
100   let {B1-w1 := ((range i i + n) = SOME r');
101       B1-w2 := ((range i + n j) = SOME r'');
102       B1-w3 := (forall M .
103                 (collect M r) =
104                 (collect M r') join (collect M r''))};
105   OA2 := (!prove SWO.ordered.append-2);
106   C1 := (!chain->
107     [(ordered M r)
108      ==> (ordered' (collect M r)) [def]
109      ==> (ordered'
110          (collect M r') join
111          (collect M r'')) [B1-w3]
112      ==> (ordered' (collect M r') &
113          ordered' (collect M r'')) [OA2]
114      ==> ((ordered M r') &
115          (ordered M r'')) [def]])}
116   (!chain-> [(B1-w1 & B1-w2 & C1) ==> goal [existence]])
117 }
118
119 (add-theorems theory |{[ordered-rest-range ordered-empty-range
120   ordered-subranges] := proofs}|)
121 } # close module Ordered-Range

```