lib/memory-range/ordered-range.ath

```
load "ordered-list"
load "range-length"

#------------------------------------------------------------------------
# <ER: is an S value < or E the first element of a range of S
# values (true if the range is empty)

extend-module SWO {
  define deref := Trivial-Iterator.deref
  define start := Range.start
  define at := Memory.at

  declare <ER: (S, X) [S (Range X S)] -> Boolean

  define [M r i x] :=
         [?M:(Memory 'S) ?r:(Range 'X 'S) ?i:(It 'X 'S) ?x:'S]

  module <ER {

    define empty := (forall x i . x <ER stop i)

    define nonempty :=
      (forall x M r . x <ER back r <==> x <E M at deref start back r)

    (add-axioms theory [empty nonempty])
  }
}

#------------------------------------------------------------------------
module Ordered-Range {
  open SWO, Random-Access-Iterator

  declare ordered: (S, X) [(Memory S) (Range X S)] -> Boolean

  define ordered' := SWO.ordered

  define def := (forall r M . (ordered M r) <==> ordered' (collect M r))

  define theory := (make-theory ['SWO 'Random-Access-Iterator] [def])

  define ordered-rest-range :=
    (forall M r . (ordered M back r) ==> (ordered M r))

  define ordered-empty-range := (forall M i . (ordered M stop i))

  define ordered-subranges :=
    (forall M r i j n . (range i j) = SOME r &
                        (ordered M r) &
                        n <= length r
                        ==> exists r' r'' .
                              (range i i + n) = SOME r' &
                              (range i + n j) = SOME r'' &
                              (ordered M r') &
                              (ordered M r''))

define proofs :=
  method (theorem adapt)
   let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
        [deref <EL I+N I-N I-I ordered ordered'] :=
           (adapt [deref <EL I+N I-N I-I ordered ordered'])}
   match theorem {
     (val-of ordered-rest-range) =>
     pick-any M r
      (!chain [(ordered M (back r))
           ==> (ordered' (collect M (back r)))      [def]
           ==> (ordered' ((M at deref start back r)
                          :: (collect M r)))       [collect.of-back]
```

```
          ==> ((M at deref start back r) <EL (collect M r) &
                (ordered' (collect M r)))          [SWO.ordered.nonempty]
          ==> (ordered' (collect M r))             [right-and]
          ==> (ordered M r)                        [def]])
  | (val-of ordered-empty-range) =>
    pick-any M i
      (!chain->
       [true ==> (ordered' nil)                    [SWO.ordered.empty]
             ==> (ordered' (collect M stop i))     [collect.of-stop]
             ==> (ordered M stop i)                [def]])
  | (val-of ordered-subranges) =>
    pick-any M r:(Range 'X 'S) i:(It 'X 'S)  j:(It 'X 'S) n
      let {A1 := ((range i j) = SOME r);
           A2 := (ordered M r);
           A3 := (n <= length r)}
       assume (A1 & A2 & A3)
        let {goal := (exists r' r'' .
                       (range i i + n) = SOME r' &
                       (range i + n j) = SOME r'' &
                       (ordered M r') &
                       (ordered M r''));
             CSR := (!prove collect-split-range);
             B1 := (!chain->
                    [(A1 & A3)
                     ==> (exists r' r'' .
                           (range i i + n) = SOME r' &
                           (range i + n j) = SOME r'' &
                           (forall M .
                             (collect M r) =
                             (collect M r') join (collect M r'')))
                                                        [CSR]])}
         pick-witnesses r' r'' for B1 B1-w
          let {B1-w1 := ((range i i + n) = SOME r');
               B1-w2 := ((range i + n j) = SOME r'');
               B1-w3 := (forall M .
                           (collect M r) =
                           (collect M r') join (collect M r''));
               OA2 := (!prove SWO.ordered.append-2);
               C1 := (!chain->
                      [(ordered M r)
                       ==> (ordered' (collect M r))      [def]
                       ==> (ordered'
                             (collect M r') join
                             (collect M r''))            [B1-w3]
                       ==> (ordered' (collect M r') &
                            ordered' (collect M r''))    [OA2]
                       ==> ((ordered M r') &
                             (ordered M r''))            [def]])}
           (!chain-> [(B1-w1 & B1-w2 & C1) ==> goal     [existence]])
  }

 (add-theorems theory |{[ordered-rest-range ordered-empty-range
                         ordered-subranges] := proofs}|)
} # close module Ordered-Range
```