lib/memory-range/binary-search-range.ath

```
load "ordered-range"
load "strong-induction"
load "nat-half"
load "collect-locs"

extend-module Ordered-Range {

  declare lower-bound: (S, X) [(It X S) (It X S) S] -> (It X S)

  module lower-bound {

    define half := N.half

    define axioms :=
      (fun
       [(M \\ (lower-bound i j x)) =
          let {mid := (i + half (j - i))}
            [i              when (i = j)
             (M \\ (lower-bound (successor mid) j x))
                            when (i =/= j & M at deref mid < x)
             (M \\ (lower-bound i mid x))
                            when (i =/= j & ~ M at deref mid < x)]])
    define [empty go-right go-left] := axioms

    (add-axioms theory axioms)

    define (position-found-prop r) :=
      (forall M i j x k .
        (range i j) = SOME r &
        (ordered M r) &
        k = M \\ (lower-bound i j x)
        ==> (k *in r | k = j) &
            (k =/= i ==> M at deref predecessor k < x) &
            (k =/= j ==> x <E M at deref k))

    define position-found := (forall r . position-found-prop r)

define <' := N.<
overload * N.*
define [r1 r2 r3] := [?r1 ?r2 ?r3]

define proof :=
  method (theorem adapt)
    let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
         parity := N.parity;
         [< <E ordered deref *in successor predecessor I+N I-N I-I] :=
           (adapt [< <E ordered deref *in successor predecessor I+N I-N I-I])}
      match theorem {
       (val-of position-found) =>
       (!strong-induction.measure-induction (adapt theorem) length
        pick-any r:(Range 'X 'S)
          assume IND-HYP :=
                  (forall r' .
                    length r' <' length r ==> position-found-prop r')
           pick-any M:(Memory 'S) i:(It 'X 'S) j:(It 'X 'S)
                    x:'S k:(It 'X 'S)
            let {A1 := ((range i j) = SOME r);
                 A2 := (ordered M r);
                 A3 := (k = M \\ (lower-bound i j x))}
             assume (A1 & A2 & A3)
              let {goal :=
                     lambda (r)
                       ((k *in r | k = j) &
                        (k =/= i ==> M at deref predecessor k < x) &
                        (k =/= j ==> x <E M at deref k))}
                datatype-cases (goal r) on r {
                 (stop i0:(It 'X 'S)) =>
```

```
68                  conclude (goal (stop i0))
69                   let
70                     {EL := (!prove empty-range1);
71                      _ := (!chain->
72                            [(range i j)
73                             = (SOME r)                        [A1]
74                             = (SOME stop i0)                  [(r = stop i0)]
75                             ==> (i = j)                       [EL]]);
76                      C0 := (!chain
77                             [k = (M \\ (lower-bound i j x))   [A3]
78                                = i                            [empty
79                                                               (i = j)]]);
80                      C1 := (!chain [k = i                     [C0]
81                                     = j                       [(i = j)]]);
82                      C2 := (!chain->
83                             [C1 ==> (k *in (stop i0) | k = j)
84                                                               [alternate]]);
85                      C3 := assume (k =/= i)
86                              (!from-complements
87                               (M at deref predecessor k < x)
88                               (k = i)
89                               (k =/= i));
90                      C4 := assume (k =/= j)
91                              (!from-complements
92                               (x <E M at deref k)
93                               (k = j)
94                               (k =/= j))}
95                    (!chain-> [(C2 & C3 & C4) ==> (goal (stop i0))
96                                                               [prop-taut]])
97                | (back r0:(Range 'X 'S)) =>
98                  conclude (goal (back r0))
99                   let {NB := (!prove nonempty-back1);
100                        E1 := (!chain->
101                               [(range i j)
102                                = (SOME r)                     [A1]
103                                = (SOME back r0)               [(r = back r0)]
104                                ==> (i =/= j)                  [NB]]);
105                       (and E2 E3) :=
106                        (!chain->
107                         [A1 ==> ((range i j) =
108                                  (range start r finish r))
109                                                  [range.collapse]
110                              ==> (i = start r & j = finish r)
111                                                  [range.injective]]);
112                       RL2 := (!prove length2);
113                       n := (length r);
114                       E4 := (!chain
115                              [n = ((finish r) - (start r))  [RL2]
116                                 = (j - i)                   [E2 E3]]);
117                       E4' := (!by-contradiction (n =/= zero)
118                               assume (n = zero)
119                                 (!absurd
120                                  (!chain
121                                   [(S (length r0))
122                                   = (length (back r0))  [length.of-back]
123                                   = n                   [(r = back r0)]
124                                   = zero                [(n = zero)]])
125                                  (!chain->
126                                   [true ==> (S (length r0) =/= zero)
127                                                         [N.S-not-zero]])));
128                       E5 := (!chain->
129                              [(n =/= zero)
130                               ==> (half length r <' n)  [N.half.less]
131                               ==> (half (j - i) <' n)   [E4]]);
132                       E6 := (!chain-> [E5 ==> (half (j - i) <= n)
133                                               [N.Less=.Implied-by-<]]);
134                       mid := (i + half (j - i));
135                       OS := (!prove ordered-subranges);
136                       E7 := (!chain->
137                              [(A1 & A2 & E6)
```

```
                ==> (exists r1 r2 .
                      (range i mid) = SOME r1 &
                      (range mid j) = SOME r2 &
                      (ordered M r1) &
                      (ordered M r2))    [OS]])}
          pick-witnesses r1 r2 for E7 E7-w
           let {E7-w1 := ((range i mid) = SOME r1);
                E7-w2 := ((range mid j) = SOME r2);
                E7-w3 := (ordered M r1);
                E7-w4 := (ordered M r2);
                IIC := (!prove I-I-cancellation);
                RL3 := (!prove length3);
                X1 := (!chain
                       [E7-w1 ==> (length r1 = mid - i) [RL3]]);
                X2 := (!chain
                       [E7-w2 ==> (length r2 = j - mid) [RL3]]);
                Q1 := (!chain [(length r1)
                               = (mid - i)              [X1]
                               = (half (j - i))         [IIC]
                               = (half n)               [E4]]);
                RL4 := (!prove length4);
                Q2 := (!chain->
                       [(A1 & E7-w1 & E7-w2)
                        ==> (n = (length r1) + (length r2))
                                                        [RL4]]);
                Q3 := (!chain
                       [n = (N.two * (half n) + (parity n))
                                          [N.parity.half-case]
                          = (((half n) + (half n)) + (parity n))
                                          [N.Times.two-times]
                          = (((half n) + (parity n)) + (half n))
                                          [N.Plus.associative
                                           N.Plus.commutative]]);
                Q4 := (!chain->
                       [((length r2) + (half n))
                        = ((half n) + (length r2))
                                          [N.Plus.commutative]
                        = ((length r1) + (length r2))    [Q1]
                        = n                              [Q2]
                        = (((half n) + (parity n)) + (half n))
                                                         [Q3]
                        ==> (length r2 = (half n) + (parity n))
                                          [N.Plus.=-cancellation]]);
                NZL := (!prove nonzero-length);
                F2 := (!chain->
                       [(n =/= zero)
                        ==> ((half n) + (parity n) =/= zero)
                                          [N.parity.plus-half]
                        ==> (length r2 =/= zero)         [Q4]
                        ==> (exists r3 . r2 = back r3)   [NZL]])}
          pick-witness r3 for F2 F2-w
           (!two-cases
            assume G1 := (M at deref mid < x)
             let
               {H1 := (!chain
                       [k = (M \\ (lower-bound i j x))   [A3]
                          = (M \\ (lower-bound (successor mid)
                                               j x))
                                    [(i =/= j) G1 go-right]]);
                LB := (!prove range-back);
                E7-w2' := (!chain->
                           [E7-w2
                            ==> ((range mid j) = SOME back r3)
                                                        [F2-w]
                            ==> ((range (successor mid) j) =
                                 SOME r3)               [LB]]);
                H2 := (!chain
                       [(length r2)
                        = (length back r3)              [F2-w]
                        = (S length r3)       [length.of-back]]);
```

```
208                     H3 := (!chain->
209                            [Q2 ==> (n = (length r2) + (length r1))
210                                                    [N.Plus.commutative]
211                                ==> (length r2 <= n)
212                                                    [N.Less=.k-Less=]]);
213                     _ := (!chain->
214                            [true ==> (length r3 <= length r3)
215                                                    [N.Less=.reflexive]
216                                  ==> (length r3 <' S length r3)
217                                                    [N.Less=.S1]
218                                  ==> (length r3 <' length r2)
219                                                    [H2]
220                                  ==> (length r3 <' length r2 & H3)
221                                                    [augment]
222                                  ==> (length r3 <' n)
223                                                 [N.Less=.transitive1]]);
224                     ORR := (!prove ordered-rest-range);
225                     E7-w4' := (!chain->
226                                 [E7-w4 ==> (ordered M back r3)
227                                                    [F2-w]
228                                        ==> (ordered M r3)
229                                                    [ORR]]);
230                     (and H5 (and H6 H7)) :=
231                      (!chain->
232                       [(E7-w2' & E7-w4' & H1)
233                    ==> ((k *in r3 | k = j) &
234                         (k =/= successor mid ==>
235                          M at deref predecessor k < x) &
236                         (k =/= j ==> x <E M at deref k))
237                                                    [IND-HYP]]);
238                     IWR2 := (!prove
239            Random-Access-Iterator.collect-locs.*in-whole-range-2);
240                     H8 := (!chain->
241                            [(n =/= zero)
242                             ==> (S half n <= n)
243                                                 [N.half.less-equal-1]]);
244                     SI := (!prove successor-in);
245                     H9 := (!sym
246                             (!chain
247                              [(SOME r3)
248                             = (range (successor mid) j)    [E7-w2']
249                             = (range (successor i) + half
250                                      (j - i) j)            [SI]
251                             = (range i + S half (j - i) j) [I+pos]
252                             = (range i + (S half n) j)     [E4]]));
253                     H10 := (!chain->
254                             [(A1 & H8 & H9 & H5)
255                              ==> (k *in r | k = j)         [IWR2]
256                              ==> (k *in (back r0) | k = j)
257                                                    [(r = back r0)]]);
258                     subgoal := (M at deref predecessor k < x);
259                     H11 := assume (k =/= i)
260                              (!two-cases
261                               assume J1 := (k = successor mid)
262                                 let {K1 := (!chain
263                                             [(predecessor k)
264                                            = (predecessor
265                                               successor mid) [J1]
266                                             = mid
267                                         [predecessor.of-successor]])}
268                                  (!chain->
269                                   [G1 ==> subgoal            [K1]])
270                               assume J2 := (k =/= successor mid)
271                                 (!chain-> [J2 ==> subgoal    [H6]]))}
272                (!chain-> [(H10 & H11 & H7) ==> (goal (back r0))
273                                                    [prop-taut]])
274               assume G2 := (~ M at deref mid < x)
275                let {
276                   J1 := (!chain
277                          [k = (M \\ (lower-bound i j x))   [A3]
```

```
                                    = (M \\ (lower-bound i mid x))
                                              [(i =/= j) G2 go-left]]);
                          _ := (!chain->
                                [(n =/= zero)
                                 ==> ((half n) <' n)          [N.half.less]
                                 ==> (length r1 <' n)         [Q1]]);
                          (and J2 (and J3 J4)) :=
                           (!chain->
                            [(E7-w1 & E7-w3 & J1)
                          ==> ((k *in r1 | k = mid) &
                               (k =/= i ==>
                                M at deref predecessor k < x) &
                               (k =/= mid ==> x <E M at deref k))
                                                          [IND-HYP]]);
                          IWR := (!prove
                       Random-Access-Iterator.collect-locs.*in-whole-range);
                          J5 := (!chain->
                                  [(A1 & E5 & E7-w1 & J2)
                                       ==> (k *in r)          [IWR]
                                       ==> (k *in r | k = j) [alternate]
                                       ==> (k *in (back r0) | k = j)
                                                     [(r = back r0)]]);
                          FNI := (!prove finish-not-*in);
                          J6 := assume (k =/= j)
                                  (!cases J2
                                   assume (k *in r1)
                                     (!chain->
                                      [(E7-w1 & k *in r1)
                                     ==> (k =/= mid)          [FNI]
                                     ==> (x <E M at deref k) [J4]])
                                   assume (k = mid)
                                     (!chain->
                                      [G2
                                     ==> (~ M at deref k < x)  [(k = mid)]
                                     ==> (x <E M at deref k)
                                                     [<E-definition]]))}
                     (!chain-> [(J5 & J3 & J6) ==> (goal (back r0))
                                                     [prop-taut]]))
                    } # datatype-cases
                ) # strong-induction.measure-induction
      } # match theorem

  (add-theorems theory |{[position-found] := proof}|)
 } # lower-bound
} # Ordered-Range
```