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))]
[1 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+I-N I-I])

match theorem {
(val-of position-found) =>
(position-found-prop)

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)} =>}
conclude (goal (stop i0))
let
(EL := (!prove empty-rangel));
  _ := (!chain->
    [(range i j)
     = (SOME r) [A1]
     = (SOME stop i0) [(r = stop i0)]
     ==> (i = j) [EL]);
C0 := (!chain
   [k = (M \ \ (lower-bound i j x)) [A3]
   = i [empty
   (i = j)]]);
C1 := (!chain [k = i [C0]
   = j [([i = j])]);
C2 := (!chain->
   [C1 ==> (k *in (stop i0) | k = j)
   [alternate]]);
C3 := assume (k /= i)
  (!from-complements
   (M at deref predecessor k < x)
   (k = i)
   (k /= i));
C4 := assume (k /= j)
  (!from-complements
   (x <E M at deref k)
   (k = j)
   (k /= j));
  (!chain-> [(C2 & C3 & C4) ==> (goal (stop i0)]
  [prop-taut]))
| (back r0:(Range 'X 'S)) =>
conclude (goal (back r0))
let {NB := (!prove nonempty-back1);
E1 := (!chain->
  [(range i j)
   = (SOME r) [A1]
   = (SOME back r0) [(r = back r0)]
   ==> (i /= j) [NB]);
(and E2 E3) :=
  (!chain->
   [A1 ==> ((range i j) =
   (range start r finish r))
   ==> (i = start r & j = finish r)]
   [range.collapse]
   [range.injective]);
RL2 := (!prove length2);
n := (length r);
E4 := (!chain
  [n = ((finish r) - (start r)) [RL2]
  = (j - i) [E2 E3]));
E4' := (!by-contradiction (n /= zero)
  assume (n = zero)
  (!absurd (!chain
    [S (length r0) = (length (back r0)) [length.of-back]
    = n [(r = back r0)]
    = zero [(n = zero)])]
    (!chain->
     [true ==> (S (length r0) /= zero)
     [N.S-not-zero]]));
E5 := (!chain->
  [(n /= zero)
  ==> (half length r '< n) [N.half.less]
  ==> (half (j - i) '< n) [E4]]);
E6 := (!chain->
  [E5 ==> (half (j - i) <= n)
  [N.Less=.Implied-by-<]]);
mid := (i + half (j - i));
OS := (!prove ordered-subranges);
E7 := (!chain->
  [(A1 & A2 & E6)
138 => (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);
IC := (!prove I-I-cancellation);
RL3 := (!prove length3);
X1 := (!prove -> (length r1 = mid - i) [RL3]])
X2 := (!prove -> (length r2 = j - mid) [RL3]])

Q1 := (!chain [(length r1)
  = (mid - i) [X1]
  = (half (j - i)) [IIIC]
  = (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.paruty .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)
   => (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]])
H3 := (!chain->
   [Q2 ==> (n = (length r2) + (length r1))
    [N.Plus.commutative]
    ==> (length r2 <= n)
    [N.Less=.k-Less=]]));

_ := (!chain->
   [true ==> (length r3 <= length r3)
    [N.Less=.reflexive]
    ==> (length r3 <= S length r3)
    [N.Less=.S1]
    ==> (length r3 <= length r2)
    [H2]
    ==> (length r3 <= length r2 & H3)
    [augment]
    ==> (length r3 <= n)
    [N.Less=.transitive]]);

ORR := (!prove ordered-rest-range);
E7-w4' := (!chain->
   [E7-w4 ==> (ordered M back r3)
    [F2-w]
    ==> (ordered M r3)
    [ORR]]);

(and H5 (and H6 H7)) :=
(!chain->
   [(E7-w2' & E7-w4' & H1)
    ==> (k *in r3 | k = j) &
    (k /= successor mid ==> M at deref predecessor k < x) &
    (k /= j ==> x < E M at deref k))
    [IND-HYP]]);

IWR2 := (!prove Random-Access-Iterator.collect-locs.*in-whole-range-2);
H8 := (!chain->
   [(n /= zero)
    ==> (S half n <= n)
    [N.half.less-equal-1]]);

SI := (!prove successor-in);
H9 := (!sym
   ![chain
      [(SOME r3)
       = (range (successor mid) j) [E7-w2']
       = (range (successor i) + half
         (j - i) j) [SI]
       = (range i + S half (j - i) j) [I+pos]
       = (range i + (S half n) j) [E4]])];

H10 := (!chain->
   [(A1 & H8 & H9 & H5)
    ==> (k *in r | k = j) [IWR2]
    ==> (k *in (back r0) | k = j)
    [(r = back r0)]]);

subgoal := (M at deref predecessor k < x);
H11 := assume (k /= i)
   (!two-cases
    assume J1 := (k = successor mid)
    let {K1 := (!chain
       [(predecessor k)
        = (predecessor successor mid) [J1]
        = mid
        [predecessor.of-successor]]))
    ![chain->
       [G1 ==> subgoal [K1]]]
    assume J2 := (k /= successor mid)
    ![chain->
       [J2 ==> subgoal [H6]]]])
   ![chain->
      [(H10 & H11 & H7) ==> (goal (back r0))
      [prop-taut]]]

assume G2 := (~ M at deref mid < x)
let {
   J1 := (!chain
      [k = (M \ {lower-bound i j x}) [A3]}}
278  = (M \ (lower-bound i mid x))
279  [([i /= j] G2 go-left)])
280  
281  (and J2 (and J3 J4)) :=
282  (!chain->
283   (E7-w1 & E7-w3 & J1)
284   ==> ((k *in r1 | k = mid) &
285     (k /=/= i ==> M at deref predecessor k < x) &
286     (k /=/= mid ==> x <E M at deref k))
287   [IND-HYP]]);
288
289  IWR := (!prove
290   Random-Access-Iterator.collect-locs.*in-whole-range);
291  J5 := (!chain->
292   (A1 & E5 & E7-w1 & J2)
293   ==> (k *in r) [IWR]
294   ==> (k *in r | k = j) [alternate]
295   ==> (k *in (back r0) | k = j)
296   [(r = back r0)]);
297  FNI := (!prove finish-not-*in);
298  J6 := assume (k /=/= j)
299  (!cases J2
300     assume (k *in r1)
301     (!chain->
302       (E7-w1 & k *in r1)
303       ==> (k /=/= mid) [FNI]
304       ==> (x <E M at deref k) [J4])]
305     assume (k = mid)
306     (!chain->
307       (G2
308       ==> (~ M at deref k < x) [(k = mid)]
309       ==> (x <E M at deref k)
310       [<E-definition]])
311   )
312   [prop-taut]]))
313
314   (!chain-> [J5 & J3 & J6] ==> (goal (back r0))
315   [prop-taut]]))
316
317 )  
318 )
319 )
320  (add-theorems theory |{[position-found] := proof}|
321 )
322 )