lib/memory-range/range.ath

load "nat-plus"
domain (It X S)

datatype (Range X S) :=
  (stop (It X S))  # An empty range beginning and ending at the
                 # given iterator
  | (back (Range X S))  # A range that begins one step back from where
                 # the argument range begins

assert Range-axioms := (datatype-axioms "Range")

#..........................................................................

module Range {
  define theory := (make-theory [] [])

  define [h i i' j j' r r'] :=
    [h:(It X S) ?i:(It X S) ?i':(It X S) ?j:(It X S) ?j':(It X S) ?r:(Range X S) ?r':(Range X S)]

  # (start r) returns the beginning of range r
  declare start: (X, S) [(Range X S)] -> (It X S)

  module start {
    define of-stop := (forall i . start stop i = i)
    define injective := (forall r r' . start r = start r' ==> r = r')
    (add-axioms theory [of-stop injective])
  }

  # (finish r) returns the end of range r
  declare finish: (X, S) [(Range X S)] -> (It X S)

  module finish {
    define of-stop := (forall i . finish stop i = i)
    define of-back := (forall r . finish back r = finish r)
    (add-axioms theory [of-stop of-back])
  }

  declare range: (X, S) [(It X S) (It X S)] -> (Option (Range X S))

  module range {
    define collapse := (forall r . (range (start r) (finish r)) = SOME r)
    define injective :=
      (forall i j i' j'. (range i j) = (range i' j') ==> i = i' & j = j')
    define start-back :=
      (forall i j r . (range i j) = SOME back r ==> i = start back r)
    (add-axioms theory [collapse injective start-back])
  }

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

  module empty {
    define of-stop := (forall i . empty stop i)
    define of-back := (forall r . ~ empty back r)
    (add-axioms theory [of-stop of-back])
  }
}
declare length : (X, S) [(Range X S)] -> N

module length {

  define of-stop := (forall j . length stop j = zero)
  define of-back := (forall r . length back r = S length r)

  (add-axioms theory [of-stop of-back])
}

# Range theorems:

define nonempty-back := (forall r . start back r /= finish back r)
define nonempty-back1 :=
  (forall i j r . (range i j) = SOME back r ==> i /= j)
define back-not-same := (forall r . back r /= r)
define empty-range := (forall i . (range i i) = SOME stop i)
define empty-range1 :=
  (forall h i j . (range i j) = SOME stop h ==> i = j)
define zero-length :=
  (forall r . length r = zero ==> exists i . r = stop i)
define nonzero-length :=
  (forall r . length r /= zero ==> exists r' . r = back r')
define theorems := [nonempty-back nonempty-back1 back-not-same empty-range empty-range1 zero-length nonzero-length]
define proofs :=
method (theorem adapt)
let {[get prove chain chain-> chain<-] := (proof-tools adapt theory)}
match theorem {
  (val-of nonempty-back) =>
    pick-any r
    (by-contradiction (start back r /= finish back r))
assume A := (start back r = finish back r)
(!absurd
  (!chain-> [(start back r) = (finish back r) [A]
  = (start stop finish r) [finish.of-back]
  = (start of-stop) [start.of-stop]
  ==> (back r = stop finish r) [start.injective]])
  (!chain-> [true ==> (stop finish r /= back r) [Range-axioms]
  ==> (back r /= stop finish r) [sym]]))
  | (val-of nonempty-back1) =>
    pick-any r
    assume A := ((range i j) = SOME back r)
    conclude (i /= j)
let {NB := (!prove nonempty-back)};
B := (!chain->
  [(range i j) = (SOME back r) [A]
  = (range (start back r)
  (finish back r)) [range.collapse]
  ==> (i = start back r &
  j = finish back r) [range.injective]])
  (!chain->
  [true ==> (start back r /=
  finish back r) [NB]
  ==> (i /= j) [B]])
  | (val-of back-not-same) =>
    by-induction (adapt theorem) {
    (stop i) =>
      (!chain->
      [true ==> (stop i /= back stop i) [Range-axioms]
      ==> (back stop i /= stop i) [sym]])
    | (back r) =>
      let {ind-hyp := (back r /= r)}
      (!chain->
[(ind-hyp ==> (back back r =/= back r) [Range-axioms]))
]

| (val-of empty-range) =>
  pick-any i.
  \{chain
   \{range i i\} = (range (start stop i) (finish stop i)) [start.of-stop
   finish.of-stop]
   \{SOME stop i\} [range.collapse]\}
  |
  | (val-of empty-range) =>
  pick-any h:(It X S) i:(It X S) j:(It X S)
  assume A := ((range i j) = SOME stop h)
  conclude (i = j)
  let EL := (!prove empty-range);
     (and B1 B2) :=
       \{chain->
       \{some stop h\} [A]
       \{range h\} [EL]
       \{i = h & j = h\} [range.injective]\})
  |
  | (val-of zero-length) =>
  datatype-cases (adapt theorem) {
    (stop i) =>
      assume A := (length stop i = zero)
      (!chain->
       \{(stop i = stop i) => (exists ?i . stop i = stop ?i) \[existence]\})
      (!chain ->
       \{true ==> (S length r =/= zero) \[N.S-not-zero]\}
       \{length back r =/= zero \[length.of-back]\}))
    | (back r) =>
      assume A := (length back r = zero)
      (!from-complements (exists ?i . back r = stop ?i)
       A)
      (!chain->
       \{true ==> (S length r =/= zero) \[N.S-not-zero]\}
       \{length back r =/= zero \[length.of-back]\}))
    |
    | (val-of nonzero-length) =>
    datatype-cases (adapt theorem) {
      (stop i) =>
        assume A := (length stop i =/= zero)
        (!chain->
         \{(length stop i) = zero \[length.of-stop]\})
        A)
      | (back r) =>
        assume A := (length back r =/= zero)
        (!chain->
         \{true ==> (S length r =/= zero) \[N.S-not-zero]\}
         \{length of-back\})
      |
      }
define proofs :=

method (theorem adapt)
let {
\{get prove chain chain-> chain<-\} := (proof-tools adapt theory)}

match theorem {
(val-of range-expand) =>
pick-any i:(It 'X 'S) r:(Range 'X 'S)
{!chain
[(i in r)
===> (i = start back r | i in r) [alternate]
===> (i in (back r)) [of-back]]}

| (val-of range-reduce) =>
pick-any i r

let (RE := {!prove range-expand});
P := {!chain [(i in r) ==> (i in back r) [RE]]}
(!contra-pos P)
}

(add-theorems theory |{theorems := proofs}|)

} # close module in

} # close module Range