lib/memory-range/range-length.ath

```
load "random-access-iterator"
#.....................................................................

extend-module Random-Access-Iterator {
  define join := List.join
  overload <= N.<=

  define length1 := (forall r . start r = (finish r) - (length r))

  define length2 := (forall r . length r = (finish r) - (start r))

  define length3 :=
    (forall i j r . (range i j) = SOME r ==> length r = j - i)

  define r'' := ?r'':(Range 'X 'S)

  define length4 :=
    (forall i j n r r' r'' .
      (range i j) = SOME r &
      (range i i + n) = SOME r' &
      (range i + n j) = SOME r''
      ==> length r = (length r') + (length r''))

  define (contained-range-prop n) :=
    (forall r i j k .
      (range i j) = SOME r &
      k = i + n &
      n <= length r
      ==> exists r' . (range i k) = SOME r')

  define contained-range := (forall n . contained-range-prop n)

  define (collect-split-range-prop n) :=
    (forall i j r .
      (range i j) = SOME r & n <= length r
      ==> 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''))

  define collect-split-range := (forall n . collect-split-range-prop n)

define [n0 r0 r0'] := [?n0 ?r0 ?r0']

define proofs :=
  method (theorem adapt)
    let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
        [successor predecessor I+N I-N I-I] :=
          (adapt [successor predecessor I+N I-N I-I])}
    match theorem {
      (val-of length1) =>
      by-induction (adapt theorem) {
        (stop j) =>
        conclude (start stop j = (finish stop j) - (length stop j))
          (!combine-equations
           (!chain [(start stop j) = j                      [start.of-stop]])
           (!chain [((finish stop j) - (length stop j))
                    = (j - zero)                            [finish.of-stop
                                                             length.of-stop]
                    = j                                     [I-0]]))
      | (r as (back r')) =>
        conclude (start r = (finish r) - (length r))
          let {ind-hyp := (start r' = (finish r') - length r')}
          (!combine-equations
           (!chain [(start r)
                    = (predecessor start r')          [predecessor.of-start]
```

```
68                    = (predecessor
69                       ((finish r') - length r'))      [ind-hyp]])
70            (!chain [((finish r) - length r)
71                    = ((finish r') - S length r')      [finish.of-back
72                                                        length.of-back]
73                    = (predecessor
74                       ((finish r') - length r'))      [I-pos]]))
75        }
76      | (val-of length2) =>
77        pick-any r:(Range 'X 'S)
78         let {RL1 := (!prove length1)}
79          (!chain->
80           [true
81            ==> (start r = ((finish r) - length r))      [RL1]
82            ==> ((finish r) - (start r) = length r)      [I-I]
83            ==> (length r = (finish r) - start r)        [sym]])
84      | (val-of length3) =>
85        pick-any i:(It 'X 'S) j:(It 'X 'S) r:(Range 'X 'S)
86          assume A := ((range i j) = SOME r)
87            let {B := (!chain->
88                       [(range (start r) (finish r))
89                      = (SOME r)                          [range.collapse]
90                      = (range i j)                       [A]
91                      ==> (start r = i & finish r = j)    [range.injective]]);
92                 RL2 := (!prove length2)}
93             (!chain->
94              [(length r) = ((finish r) - start r)       [RL2]
95                          = (j - i)                      [B]])
96      | (val-of length4) =>
97        conclude
98          (forall i j n r r' r'' .
99            (range i j) = SOME r &
100           (range i i + n) = SOME r' &
101           (range i + n j) = SOME r''
102           ==> length r = (length r') + length r'')
103       pick-any i:(It 'X 'S) j:(It 'X 'S) n r:(Range 'X 'S)
104                r':(Range 'X 'S) r'':(Range 'X 'S)
105        let {A1 := ((range i j) = SOME r);
106             A2 := ((range i i + n) = SOME r');
107             A3 := ((range i + n j) = SOME r'');
108             RL3 := (!prove length3);
109             IIMN := (!prove I-M-N);
110             IIC := (!prove I-I-cancellation)}
111         assume (A1 & A2 & A3)
112           let {k := (i + n);
113                B0 := (!chain-> [A1 ==> (length r = j - i) [RL3]]);
114                B1 := (!chain->
115                       [A3 ==> (length r'' = j - k)        [RL3]
116                           ==> (j - k = length r'')        [sym]
117                           ==> (k = j - length r'')        [I-I]]);
118                B2 := (!chain->
119                       [A2 ==> (length r' = k - i)         [RL3]
120                           ==> (k - i = length r')         [sym]
121                           ==> (i = k - length r')         [I-I]])}
122              (!chain->
123               [i = (k - length r')                       [B2]
124                  = ((j - (length r'')) - length r')      [B1]
125                  = (j - ((length r'') + length r'))      [I-M-N]
126                  ==> (j - i = (length r'') + length r')  [I-I]
127                  ==> (length r = (length r'') + length r') [B0]
128                  ==> (length r = (length r') + length r'')
129                                          [N.Plus.commutative]])
130     | (val-of contained-range) =>
131       by-induction (adapt theorem) {
132         zero =>
133         pick-any r:(Range 'X 'S) i:(It 'X 'S) j:(It 'X 'S) k:(It 'X 'S)
134           let {A1 := ((range i j) = SOME r);
135                A2 := (k = i + zero);
136                A3 := (zero <= length r);
137                EL := (!prove empty-range)}
```

```
138           assume (A1 & A2 & A3)
139             let {C1 := (!chain [k = (i + zero) [A2]
140                                = i          [I+0]])}
141              (!chain->
142               [(range i k)
143                = (range i i)      [C1]
144                = (SOME stop i)    [EL]
145                ==> (exists r' . (range i k) = SOME r') [existence]])
146       | (n as (S n')) =>
147         let {ind-hyp := (contained-range-prop n')}
148         pick-any r:(Range 'X 'S) i:(It 'X 'S) j:(It 'X 'S) k:(It 'X 'S)
149           let {A1 := ((range i j) = SOME r);
150                A2 := (k = i + n);
151                A3 := (n <= length r);
152                goal := (exists r' . (range i k) = SOME r');
153                NL := (!prove nonzero-length)}
154           assume (A1 & A2 & A3)
155             let {B0 := (!chain->
156                         [A3
157                          ==> (exists n0 . length r = S n0)  [N.Less=.S4]])}
158               pick-witness n0 for B0 B0-w
159                 let {B := (!chain->
160                            [true
161                             ==> (S n0 =/= zero)               [N.S-not-zero]
162                             ==> (length r =/= zero)           [B0-w]
163                             ==> (exists r0 . r = (back r0))   [NL]]);
164                      LB := (!prove range-back)}
165                 pick-witness r0 for B B-w
166                  let {C0 := (!chain->
167                              [(range i j)
168                               = (SOME r)                      [A1]
169                               = (SOME back r0)                [B-w]
170                               ==> ((range (successor i) j) =
171                                    SOME r0)                   [LB]]);
172                       C1 := (!chain [k = (i + n)             [A2]
173                                      = ((successor i) + n') [I+pos]]);
174                       C2 := (!chain->
175                              [A3
176                               ==> (n <= length back r0) [B-w]
177                               ==> (n <= S length r0)    [length.of-back]
178                               ==> (n' <= length r0)     [N.Less=.injective]]);
179                       C3 := (!chain->
180                              [(C0 & C1 & C2)
181                               ==> (exists r' .
182                                      (range (successor i) k) = SOME r')
183                                                         [ind-hyp]])}
184                  pick-witness r' for C3 C3-w
185                    (!chain->
186                     [C3-w
187                      ==> ((range i k) = SOME (back r')) [LB]
188                      ==> goal                           [existence]])
189         }
190       | (val-of collect-split-range) =>
191         by-induction (adapt theorem) {
192           zero =>
193            pick-any i:(It 'X 'S) j:(It 'X 'S) r:(Range 'X 'S)
194             let {A1 := ((range i j) = SOME r);
195                  A2 := (zero <= length r)}
196              assume (A1 & A2)
197                let {goal := (exists r' r'' .
198                              (range i i + zero) = SOME r' &
199                              (range i + zero j) = SOME r'' &
200                              (forall M .
201                               (collect M r) =
202                               (collect M r') join (collect M r'')));
203                     EL := (!prove empty-range);
204                     B1 := (!chain
205                            [(range i i + zero)
206                             = (range i i)          [I+0]
207                             = (SOME stop i)        [empty-range]]);
```

```
208                 B2 := (!chain
209                        [(range i + zero j)
210                         = (range i j)          [I+0]
211                         = (SOME r)             [A1]]);
212                 B3 := pick-any M
213                       (!sym (!chain
214                              [((collect M stop i) join (collect M r))
215                               = (nil join (collect M r))
216                               [collect.of-stop]
217                               = (collect M r)  [List.join.left-empty]]))}
218           (!chain-> [(B1 & B2 & B3) ==> goal  [existence]])
219       | (n as (S n')) =>
220         pick-any i:(It 'X 'S) j:(It 'X 'S) r:(Range 'X 'S)
221          let {A1 := ((range i j) = SOME r);
222               A2 := (S n' <= length r)}
223           assume (A1 & A2)
224            let {goal := (exists r' r'' .
225                           (range i i + n) = SOME r' &
226                           (range (i + n) j) = SOME r'' &
227                           (forall M .
228                             (collect M r) =
229                             (collect M r') join (collect M r'')));
230                 ind-hyp := (collect-split-range-prop n');
231                 B1 := (!chain->
232                        [A2
233                         ==> (exists n0 . length r = S n0)
234                         [N.Less=.S4]])}
235              pick-witness n0 for B1 B1-w
236                let {NL := (!prove nonzero-length);
237                     C1 := (!chain->
238                            [true
239                             ==> (S n0 =/= zero)              [N.S-not-zero]
240                             ==> (length r =/= zero)          [B1-w]
241                             ==> (exists r0 . r = (back r0))  [NL]])}
242                pick-witness r0 for C1 C1-w
243                  let {LB := (!prove range-back);
244                       D1 := (!chain->
245                              [(range i j)
246                               = (SOME r)          [A1]
247                               = (SOME back r0) [C1-w]
248                               ==> ((range (successor i) j) =
249                                    SOME r0)                  [LB]]);
250                       D2 := (!chain->
251                              [A2 ==> (n <= length back r0)   [C1-w]
252                                  ==> (n <= S length r0)      [length.of-back]
253                                  ==> (n' <= length r0) [N.Less=.injective]]);
254                       D3 := (!chain->
255                              [(D1 & D2)
256                               ==>
257                               (exists r0' r'' .
258                                (range (successor i) (successor i) + n') =
259                                       SOME r0' &
260                                (range (successor i) + n' j) = SOME r'' &
261                                (forall M .
262                                 (collect M r0) =
263                                 (collect M r0') join (collect M r'')))
264                                                              [ind-hyp]])}
265                  pick-witnesses r0' r'' for D3 D3-w
266                    let {D3-w1 := ((range (successor i) (successor i) + n')
267                                   = SOME r0');
268                         D3-w2 := ((range (successor i) + n' j) = SOME r'');
269                         D3-w3 := (forall M .
270                                    (collect M r0) =
271                                    (collect M r0') join (collect M r''));
272                         E1 := (!chain->
273                                [D3-w1
274                                 ==> ((range (successor i) i + n) =
275                                      SOME r0')                    [I+pos]
276                                 ==>((range i i + n) =
277                                      SOME back r0')               [LB]]);
```

```
278                         E2 := (!chain->
279                                [D3-w2
280                                 ==> ((range i + n j) = SOME r'')   [I+pos]]);
281                         E3 := pick-any M
282                                let {SB := (!prove range.start-back);
283                                     F1 := (!chain->
284                                            [E1 ==> (i = start back r0')
285                                                    [range.start-back]]);
286                                     F2 := (!chain->
287                                            [(range i j)
288                                             = (SOME r)             [A1]
289                                             = (SOME (back r0))     [C1-w]
290                                             ==> (i = start back r0)
291                                                    [range.start-back]]);
292                                     F3 := (!chain
293                                            [(start back r0)
294                                             = i                    [F2]
295                                             = (start back r0')     [F1]])}
296                                  (!chain
297                                   [(collect M r)
298                                    = (collect M (back r0))         [C1-w]
299                                    = ((M at deref start back r0) ::
300                                       (collect M r0))    [collect.of-back]
301                                    = ((M at deref start back r0) ::
302                                       ((collect M r0') join
303                                        (collect M r'')))           [D3-w3]
304                                    = (((M at deref start back r0) ::
305                                        (collect M r0'))
306                                       join (collect M r''))
307                                                  [List.join.left-nonempty]
308                                    = (((M at deref start back r0') ::
309                                        (collect M r0'))
310                                       join (collect M r''))        [F3]
311                                    = ((collect M back r0') join
312                                       (collect M r''))   [collect.of-back]])}
313                       (!chain-> [(E1 & E2 & E3) ==> goal  [existence]])
314     }
315   }
316
317   (add-theorems theory |{[length1 length2 length3 length4 contained-range
318                           collect-split-range] := proofs}|)
319 } # Random-Access-Iterator
```