

lib/memory-range/range.ath

```

1 load "nat-plus"
2
3 domain (It X S)
4
5 datatype (Range X S) :=
6   (stop (It X S))      # An empty range beginning and ending at the
7                       # given iterator
8 | (back (Range X S))  # A range that begins one step back from where
9                       # the argument range begins
10
11 assert Range-axioms := (datatype-axioms "Range")
12
13 #.....
14
15 module Range {
16
17   define theory := (make-theory [] [])
18
19   define [h i i' j j' r r'] := [?h:(It 'X 'S) ?i:(It 'X 'S) ?i':(It 'X 'S)
20                               ?j:(It 'X 'S) ?j':(It 'X 'S)
21                               ?r:(Range 'X 'S) ?r':(Range 'X 'S)]
22
23 # (start r) returns the beginning of range r
24 declare start: (X, S) [(Range X S)] -> (It X S)
25
26 module start {
27
28   define of-stop := (forall i . start stop i = i)
29   define injective := (forall r r' . start r = start r' ==> r = r')
30
31   (add-axioms theory [of-stop injective])
32 }
33
34 # (finish r) returns the end of range r
35 declare finish: (X, S) [(Range X S)] -> (It X S)
36
37 module finish {
38
39   define of-stop := (forall i . finish stop i = i)
40   define of-back := (forall r . finish back r = finish r)
41
42   (add-axioms theory [of-stop of-back])
43 }
44
45 declare range: (X, S) [(It X S) (It X S)] -> (Option (Range X S))
46
47 module range {
48
49   define collapse := (forall r . (range (start r) (finish r)) = SOME r)
50   define injective :=
51     (forall i j i' j' . (range i j) = (range i' j') ==> i = i' & j = j')
52   define start-back :=
53     (forall i j r . (range i j) = SOME back r ==> i = start back r)
54
55   (add-axioms theory [collapse injective start-back])
56 }
57
58 declare empty: (X, S) [(Range X S)] -> Boolean
59
60 module empty {
61
62   define of-stop := (forall i . empty stop i)
63   define of-back := (forall r . ~ empty back r)
64
65   (add-axioms theory [of-stop of-back])
66 }
67

```

```

68 declare length: (X, S) [(Range X S)] -> N
69
70 module length {
71
72   define of-stop := (forall j . length stop j = zero)
73   define of-back := (forall r . length back r = S length r)
74
75   (add-axioms theory [of-stop of-back])
76 }
77
78 # Range theorems:
79
80 define nonempty-back := (forall r . start back r /= finish back r)
81 define nonempty-back1 :=
82   (forall i j r . (range i j) = SOME back r ==> i /= j)
83 define back-not-same := (forall r . back r /= r)
84 define empty-range := (forall i . (range i i) = SOME stop i)
85 define empty-rangel :=
86   (forall h i j . (range i j) = SOME stop h ==> i = j)
87 define zero-length :=
88   (forall r . length r = zero ==> exists i . r = stop i)
89 define nonzero-length :=
90   (forall r . length r /= zero ==> exists r' . r = back r')
91
92 define theorems := [nonempty-back nonempty-back1
93                   back-not-same empty-range empty-rangel
94                   zero-length nonzero-length]
95
96 define proofs :=
97   method (theorem adapt)
98   let {[get prove chain chain-> chain<-] := (proof-tools adapt theory)}
99   match theorem {
100     (val-of nonempty-back) =>
101     pick-any r
102     (!by-contradiction (start back r /= finish back r)
103     assume A := (start back r = finish back r)
104     (!absurd
105     (!chain-> [(start back r)
106              = (finish back r) [A]
107              = (finish r) [finish.of-back]
108              = (start stop finish r) [start.of-stop]
109              ==> (back r = stop finish r) [start.injective]]))
110     (!chain->
111     [true ==> (stop finish r /= back r) [Range-axioms]
112     ==> (back r /= stop finish r) [sym]]))
113 | (val-of nonempty-back1) =>
114   pick-any i j r
115   assume A := ((range i j) = SOME back r)
116   conclude (i /= j)
117   let {NB := (!prove nonempty-back);
118       B := (!chain->
119           [(range i j)
120            = (SOME back r) [A]
121            = (range (start back r)
122                    (finish back r)) [range.collapse]
123            ==> (i = start back r &
124                j = finish back r) [range.injective]]))
125   (!chain->
126   [true ==> (start back r /=
127            finish back r) [NB]
128   ==> (i /= j) [B]])
129 | (val-of back-not-same) =>
130   by-induction (adapt theorem) {
131     (stop i) =>
132     (!chain->
133     [true ==> (stop i /= back stop i) [Range-axioms]
134     ==> (back stop i /= stop i) [sym]])
135 | (back r) =>
136   let {ind-hyp := (back r /= r)}
137   (!chain->

```

```

138     [ind-hyp ==> (back back r /= back r)   [Range-axioms]])
139   }
140 | (val-of empty-range) =>
141   pick-any i
142   (!chain
143     [(range i i)
144      = (range (start stop i) (finish stop i)) [start.of-stop
145                                                finish.of-stop]
146      = (SOME stop i)                          [range.collapse]])
147 | (val-of empty-range1) =>
148   pick-any h:(It 'X 'S) i:(It 'X 'S) j:(It 'X 'S)
149   assume A := ((range i j) = SOME stop h)
150   conclude (i = j)
151   let {EL := (!prove empty-range);
152       (and B1 B2) :=
153         (!chain->
154           [(range i j)
155            = (SOME stop h)           [A]
156            = (range h h)             [EL]
157            ==> (i = h & j = h)      [range.injective]]})
158     (!chain [i = h [B1] = j [B2]])
159 | (val-of zero-length) =>
160   datatype-cases (adapt theorem) {
161     (stop i) =>
162     assume A := (length stop i = zero)
163     (!chain->
164       [(stop i = stop i)
165        ==> (exists ?i . stop i = stop ?i)   [existence]])
166 | (back r) =>
167   assume A := (length back r = zero)
168   (!from-complements (exists ?i . back r = stop ?i)
169     A
170     (!chain->
171       [true ==> (S length r /= zero)   [N.S-not-zero]
172        ==> (length back r /= zero) [length.of-back]]))
173   }
174 | (val-of nonzero-length) =>
175   datatype-cases (adapt theorem) {
176     (stop i) =>
177     assume A := (length stop i /= zero)
178     (!from-complements (exists ?r0 . stop i = back ?r0)
179       (!chain-> [(length stop i) = zero [length.of-stop]])
180       A)
181 | (back r) =>
182   assume (length back r /= zero)
183   (!chain->
184     [(back r = back r)
185      ==> (exists ?r0 . back r = back ?r0)   [existence]])
186   }
187 }
188
189 (add-theorems theory |{theorems := proofs|})
190
191 #.....
192
193 declare in: (X, S) [(It X S) (Range X S)] -> Boolean
194
195 module in {
196
197   define of-stop := (forall i j . ~ i in stop j)
198   define of-back :=
199     (forall i r . i in back r <==> i = start back r | i in r)
200
201   (add-axioms theory [of-stop of-back])
202
203   define range-expand := (forall i r . i in r ==> i in (back r))
204
205   define range-reduce := (forall i r . ~ i in back r ==> ~ i in r)
206
207   define theorems := [range-expand range-reduce]

```

```
208
209 define proofs :=
210   method (theorem adapt)
211     let {[get prove chain chain-> chain<-] := (proof-tools adapt theory)}
212     match theorem {
213       (val-of range-expand) =>
214         pick-any i:(It 'X 'S) r:(Range 'X 'S)
215         (!chain
216           [(i in r)
217             ==> (i = start back r | i in r)           [alternate]
218             ==> (i in (back r))                       [of-back]])
219       | (val-of range-reduce) =>
220         pick-any i r
221         let {RE := (!prove range-expand);
222             p := (!chain [(i in r) ==> (i in back r) [RE]])}
223             (!contra-pos p)
224     }
225
226   (add-theorems theory |{theorems := proofs}|)
227 } # close module in
228 } # close module Range
```