

lib/search/binary-search.ath

```

1 # Binary search function for searching in a binary search tree, and
2 # correctness theorems. Generalized from natural number version in
3 # binary-search1-nat.ath.
4
5 load "binary-search-tree"
6
7 #-----
8
9 extend-module SWO {
10
11 declare binary-search: (S) [S (BinTree S)] -> (BinTree S)
12
13 module binary-search {
14 define [x L y R L1 y1 R1 T] := [?x:'S ?L:(BinTree 'S) ?y:'S ?R:(BinTree 'S)
15                               ?L1:(BinTree 'S) ?y1:'S ?R1:(BinTree 'S)
16                               ?T:(BinTree 'S)]
17
18 define (axioms as [go-left go-right at-root empty]) :=
19   (fun
20     [(binary-search x (node L y R)) =
21      [(binary-search x L)   when (x < y)
22       (binary-search x R)   when (y < x)
23       (node L y R)          when (~ x < y & ~ y < x)]
24      (binary-search x null) = null])
25
26 (add-axioms theory axioms)
27
28 # Theorems:
29
30 define in := BST.in
31
32 define found :=
33   (forall T . BST T ==>
34     forall x L y R .
35       (binary-search x T) = (node L y R) ==> x E y & x in T)
36
37 define not-found :=
38   (forall T . BST T ==>
39     forall x . (binary-search x T) = null ==> ~ x in T)
40
41 define in-iff-result-not-null :=
42   (forall T .
43     BST T ==>
44     forall x . x in T <==> (binary-search x T) != null)
45
46 define theorems := [found not-found in-iff-result-not-null]
47
48 define tree-axioms := (datatype-axioms "BinTree")
49
50 define (found-property T) :=
51   (forall x L1 y1 R1 .
52     (binary-search x T) = (node L1 y1 R1) ==> x E y1 & x in T)
53
54 define (not-found-prop T) :=
55   (forall x . (binary-search x T) = null ==> ~ x in T)
56
57 define proofs :=
58   method (theorem adapt)
59     let [[get prove chain chain-> chain<-] := (proof-tools adapt theory);
60         [< <E E BST binary-search] :=
61         (adapt [< <E E BST binary-search])]
62     match theorem {
63       (val-of found) =>
64       by-induction (adapt theorem) {
65         null =>
66         conclude (BST null ==> found-property null)
67         assume (BST null)

```

```

68     pick-any x L y R
69     assume A := ((binary-search x null) = (node L y R))
70     let {is-null :=
71         (!chain
72         [null
73         = (binary-search x null)      [empty]
74         = (node L y R)                [A]]);
75     is-not := (!chain->
76         [true ==> (null != (node L y R))
77         [tree-axioms]])}
78     (!from-complements (x E y & x in null) is-null is-not)
79 | (T as (node L:(BinTree 'S) y:'S R:(BinTree 'S))) =>
80 let {[ind-hyp1 ind-hyp2] := [(BST L ==> found-property L)
81     (BST R ==> found-property R)]}
82 assume hyp := (BST T)
83 conclude (found-property T)
84 let {p0 := (BST L & (forall x . x in L ==> x <E y) &
85     BST R & (forall z . z in R ==> y <E z));
86     _ := (!chain-> [hyp ==> p0      [BST.nonempty]]);
87     fpl := (!chain->
88         [p0 ==> (BST L)            [left-and]
89         ==> (found-property L) [ind-hyp1]]);
90     fpr := (!chain->
91         [p0 ==> (BST R)            [prop-taut]
92         ==> (found-property R) [ind-hyp2]]}
93 pick-any x:'S L1:(BinTree 'S) y1:'S R1:(BinTree 'S)
94 let {subtree := (node L1 y1 R1)}
95 assume hyp' := ((binary-search x T) = subtree)
96 conclude (x E y1 & x in T)
97 (!two-cases
98     assume (x < y)
99     let {in-left := (!prove BST.in.left)}
100    (!chain->
101        [(binary-search x L)
102         = (binary-search x T)      [go-left]
103         = subtree                  [hyp']
104         ==> (x E y1 & x in L)      [fpl]
105         ==> (x E y1 & x in T)      [in-left]])
106    assume (~ x < y)
107    (!two-cases
108        assume (y < x)
109        let {in-right := (!prove BST.in.right)}
110        (!chain->
111            [(binary-search x R)
112             = (binary-search x T)  [go-right]
113             = subtree              [hyp']
114             ==> (x E y1 & x in R)  [fpr]
115             ==> (x E y1 & x in T)  [in-right]])
116        assume (~ y < x)
117        let {_ := (!chain->
118            [ (~ x < y & ~ y < x)
119             ==> (x E y)            [E-definition]])};
120        i := conclude (y = y1)
121            (!chain->
122                [T = (binary-search x T)
123                 [at-root]
124                 = subtree [hyp']
125                 ==> (y = y1) [tree-axioms]]);
126        ii := conclude (x E y1)
127            (!chain->
128                [(x E y)
129                 ==> (x E y1)      [i]]);
130        in-root := (!prove BST.in.root)}
131    (!chain-> [(x E y)
132             ==> (x in T)          [in-root]
133             ==> (ii & x in T)    [augment]]))
134 }
135 | (val-of not-found) =>
136 by-induction (adapt theorem) {
137     null =>

```

```

138     assume (BST null)
139     conclude (not-found-prop null)
140     pick-any x
141     assume ((binary-search x null) = null)
142     (!chain-> [true ==> (~ x in null) [BST.in.empty]])
143 | (T as (node L y R)) =>
144     let {p1 := (not-found-prop L);
145         p2 := (not-found-prop R);
146         [ind-hyp1 ind-hyp2] := [(BST L ==> p1) (BST R ==> p2)]}
147     assume hyp := (BST T)
148     conclude (not-found-prop T)
149     let {smaller-in-left := (forall x . x in L ==> x <E y);
150         larger-in-right := (forall z . z in R ==> y <E z);
151         p0 := (BST L & smaller-in-left &
152             BST R & larger-in-right);
153         _ := (!chain-> [hyp ==> p0 [BST.nonempty]]);
154         _ := (!chain-> [p0 ==> smaller-in-left [prop-taut]]);
155         _ := (!chain-> [p0 ==> larger-in-right [prop-taut]]);
156         _ := (!chain-> [p0
157             ==> (BST L) [prop-taut]
158             ==> (not-found-prop L) [ind-hyp1]]);
159         _ := (!chain-> [p0
160             ==> (BST R) [prop-taut]
161             ==> (not-found-prop R) [ind-hyp2]]]}
162     pick-any x
163     assume hyp' := ((binary-search x T) = null)
164     (!by-contradiction (~ x in (node L y R))
165     assume (x in T)
166     let {C := (!chain->
167         [(x in T)
168         ==> (x E y | x in L | x in R)
169         [BST.in.nonempty]])}
170     (!two-cases
171     assume (x < y)
172     let {_ := (!chain->
173         [(binary-search x L)
174         = (binary-search x T) [go-left]
175         = null [hyp']
176         ==> (~ x in L) [p1]])}
177     (!cases C
178     assume (x E y)
179     (!absurd
180     (x < y)
181     (!chain->
182     [(x E y)
183     ==> (~ x < y & ~ y < x) [E-definition]
184     ==> (~ x < y) [left-and]]))
185     assume (x in L)
186     (!absurd (x in L) (~ x in L))
187     assume (x in R)
188     (!absurd (x < y)
189     (!chain->
190     [(x in R)
191     ==> (y <E x) [larger-in-right]
192     ==> (~ x < y) [<E-definition]]))
193     assume (~ x < y)
194     (!two-cases
195     assume (y < x)
196     let {_ := (!chain->
197         [(binary-search x R)
198         = (binary-search x T) [go-right]
199         = null [hyp']
200         ==> (~ x in R) [p2]])}
201     (!cases C
202     assume (x E y)
203     (!absurd
204     (y < x)
205     (!chain->
206     [(x E y)
207     ==> (~ x < y & ~ y < x) [E-definition]

```

```

208         ==> (~ y < x)                [right-and]])
209     assume (x in L)
210         (!absurd
211         (y < x)
212         (!chain->
213         [(x in L)
214         ==> (x <E y)                [smaller-in-left]
215         ==> (~ y < x)                [<E-definition]]))
216     assume (x in R)
217         (!absurd (x in R) (~ x in R))
218     assume (~ y < x)
219         (!absurd
220         (!chain->
221         [null = (binary-search x T) [hyp']
222         = T                [at-root]])
223         (!chain->
224         [true
225         ==> (null =/= T)                [tree-axioms]]))))))
226 }
227 | (val-of in-iff-result-not-null) =>
228 pick-any T
229 assume (BST T)
230 let {NF := (!prove not-found);
231      F := (!prove found)}
232 pick-any x
233 let {right :=
234      assume (x in T)
235          (!by-contradiction ((binary-search x T) /= null)
236          assume A1 := ((binary-search x T) = null)
237          (!absurd (x in T)
238          (!chain-> [A1 ==> (~ x in T) [NF]]))};
239 left :=
240 assume A2 := ((binary-search x T) =/= null)
241 let {p := (exists ?L ?y ?R .
242          (binary-search x T) = (node ?L ?y ?R));
243      _ := (!chain->
244          [true
245          ==> ((binary-search x T) = null | p)
246          [tree-axioms]
247          ==> p                [(dsyl with A2)]])}
248 pick-witnesses L y R for p p'
249 (!chain->
250 [p' ==> (x E y & x in T) [F]
251 ==> (x in T)                [right-and]])}
252 (!equiv right left)
253 } # match theorem
254
255 (add-theorems theory |{theorems := proofs}|)
256 } # binary-search
257 } # SWO

```