

lib/search/binary-search-tree.ath

```

1 # Binary search trees, a subset of binary trees defined by a
2 # predicate, BST
3
4 load "ordered-list"
5 load "binary-tree"
6
7 #-----
8
9 extend-module SWO {
10 open BinTree
11
12 declare BST: (S) [(BinTree S)] -> Boolean
13
14 module BST {
15
16 declare in: (S) [S (BinTree S)] -> Boolean
17
18 module in {
19
20 define empty := (forall x . ~ x in null)
21 define nonempty :=
22   (forall x L y R . x in (node L y R) <==> x E y | x in L | x in R)
23
24   (add-axioms theory [empty nonempty]) # SWO.Theory
25
26 define root := (forall x L y R . x E y ==> x in (node L y R))
27 define left := (forall x L y R . x in L ==> x in (node L y R))
28 define right := (forall x L y R . x in R ==> x in (node L y R))
29
30 define proofs :=
31   method (theorem adapt)
32     let [[get prove chain chain-> chain<-] := (proof-tools adapt theory);
33         [E in] := (adapt [E in])]
34     match theorem {
35       (val-of root) =>
36         pick-any x L y R
37         (!chain
38           [(x E y) ==> (x E y | x in L | x in R) [alternate]
39             ==> (x in (node L y R)) [nonempty]])
40       | (val-of left) =>
41         pick-any x L y R
42         (!chain
43           [(x in L) ==> (x in L | x in R) [alternate]
44             ==> (x E y | x in L | x in R) [alternate]
45             ==> (x in (node L y R)) [nonempty]])
46       | (val-of right) =>
47         pick-any x L y R
48         assume (x in R)
49         (!chain->
50           [(x in R) ==> (x in L | x in R) [alternate]
51             ==> (x E y | x in L | x in R) [alternate]
52             ==> (x in (node L y R)) [nonempty]])
53     }
54
55   (add-theorems theory |[root left right] := proofs)|)
56 } # in
57
58 define empty := (BST null)
59 define nonempty :=
60   (forall L y R .
61     BST (node L y R) <==>
62     BST L & (forall x . x in L ==> x <E y) &
63     BST R & (forall z . z in R ==> y <E z))
64
65   (add-axioms theory [empty nonempty])
66
67 } # BST

```

68 } # SWO