```
1 # Properties of ordered lists
3 load "list-of"
4 load "order"
6 extend-module SWO {
  open List
9 \# <EL: is a T value < or E the first element of a list of T
10 # values (true if the list is empty)
12 declare <EL: (T) [T (List T)] -> Boolean
13
14 module <EL {
15 define empty := (forall x . x <EL nil)</pre>
  define nonempty :=
    (forall x y L . x <EL (y :: L) <==> x <E y)
17
19 (add-axioms theory [empty nonempty])
20
21 define left-transitive := (forall L x y . x <E y & y <EL L ==> x <EL L)
22 define before-all-implies-before-first :=
     (forall L x . (forall y . y in L ==> x <E y) ==> x <EL L)
24 define append := (forall L M x . x <EL L & x <EL M ==> x <EL (L join M))
25 define append-2 := (forall L M x . x <EL (L join M) ==> x <EL L)
27 define theorems := [left-transitive before-all-implies-before-first
                       append append-2]
29 define proofs :=
    method (theorem adapt)
30
      let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);</pre>
31
           [< <E <EL] := (adapt [< <E <EL])}
32
       match theorem {
        (val-of left-transitive) =>
34
35
         datatype-cases theorem {
36
          nil =>
             pick-any x y
37
               assume (x <E y & y <EL nil)</pre>
                  (!chain-> [true ==> (x < EL nil) [empty]])
39
        | (z :: M) =>
            let {ET := (!prove <E-transitive) }</pre>
41
             pick-any x y
42
43
               assume (x <E y & y <EL (z :: M))</pre>
                 conclude (x <EL (z :: M))</pre>
44
                    (!chain-> [(x < E y & y < EL (z :: M))]
45
                               ==> (x < E y \& y < E z)
                                                             [nonempty]
46
                               ==> (x < E z)
                                                             [ET]
                               ==> (x <EL (z :: M))
48
                                                             [nonempty]])
49
       | (val-of before-all-implies-before-first) =>
         datatype-cases theorem {
51
          nil =>
52
53
           pick-any x
             assume (forall ?y . ?y in nil ==> x <E ?y)
54
                conclude (x <EL nil)</pre>
                 (!chain-> [true ==> (x <EL nil) [empty]])
         | (z :: L) =>
           pick-any x
58
             assume i := (forall ?y . ?y in (z :: L) ==> x < E ?y)
59
               conclude (x <EL (z :: L))</pre>
60
                  (!chain-> [(z = z) ==> (z = z | z in L) [alternate]
61
                                      ==> (z in (z :: L))
                                                           [in.nonempty]
                                     ==> (x < E z)
                                                             [i]
63
                                      ==> (x <EL (z :: L)) [nonempty]])
64
65
       | (val-of append) =>
          datatype-cases theorem {
68
            nil =>
```

```
pick-any M x
                 (!chain
70
                 [(x < EL nil & x < EL M)
                  ==> (x < EL M)
                                          [right-and]
72
                  ==> (x <EL (nil join M)) [join.left-empty]])
73
74
          | (u :: N) =>
              pick-any M x
75
                assume (x \le EL (u :: N) \& (x \le EL M))
                  (!chain-> [(x <EL (u :: N))
77
                              ==> (x <E u)
78
                                                            [nonempty]
                             ==> (x <EL (u :: (N join M))) [nonempty]
79
                              ==> (x <EL ((u :: N) join M))
80
                                                [join.left-nonempty]])
82
       | (val-of append-2) =>
83
84
         datatype-cases theorem {
          nil =>
85
          pick-any M x
             assume (x <EL (nil join M))
87
               (!chain-> [true ==> (x <EL nil)
88
                                                 [empty]])
        | (y :: L) =>
89
90
           pick-any M x
91
             (!chain [(x <EL ((y :: L) join M))
                       ==> (x < EL (y :: (L join M))) [join.left-nonempty]
92
93
                      ==> (x <E y)
                                                     [nonempty]
                      ==> (x < EL (y :: L))
94
                                                     [nonempty]])
95
97
   (add-theorems theory | {theorems := proofs} |)
99 } # <F.T.
100 #....
101 # ordered: are the elements of a list in (nondecending) order?
102
   declare ordered: (T) [(List T)] -> Boolean
103
104
105 module ordered {
106 open <EL
107
   define empty := (ordered nil)
108
  define nonempty :=
109
     (forall L x . ordered (x :: L) \langle == \rangle x \langle EL L & ordered L)
111
112
   (add-axioms theory [empty nonempty])
113
114 define head := (forall L x . ordered (x :: L) ==> x <EL L)</pre>
iii define tail := (forall L x . ordered (x :: L) ==> ordered L)
116
117 define proofs :=
    method (theorem adapt)
118
      let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);</pre>
119
            [< ordered <EL] := (adapt [< ordered <EL]) }</pre>
120
     match theorem {
121
122
        (val-of head) =>
        pick-any L x
123
          (!chain [(ordered (x :: L))
124
                   ==> (x <EL L & ordered L) [nonempty]
125
                   ==> (x <EL L)
                                             [left-and]])
126
127
     | (val-of tail) =>
128
        pick-any L x
          (!chain [(ordered (x :: L))
130
                   ==> (x <EL L & ordered L) [nonempty]
131
132
                   ==> (ordered L)
                                              [right-and]])
      }
133
135 (add-theorems theory |{[head tail] := proofs}|)
136
137
138
```

```
define first-to-rest-relation :=
     (forall L x y . ordered (x :: L) & y in L ==> x <E y)
   define cons := (forall L \times ... ordered L & (forall y ... y in <math>L ==> x < E y)
                                   ==> (ordered (x :: L)))
142
   define append :=
143
144
      (forall L M . ordered L & ordered M &
                     (forall x y \cdot x in L \& y in M ==> x <E y)
145
                     ==> ordered (L join M))
   define append-2 :=
147
      (forall L M . ordered (L join M) ==> ordered L & ordered M)
148
149
150 define theorems := [first-to-rest-relation cons append append-2]
151
   define proofs :=
     method (theorem adapt)
152
       let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);</pre>
153
             [ordered <EL] := (adapt [ordered <EL]) }</pre>
154
       match theorem {
155
          (val-of first-to-rest-relation) =>
         by-induction (adapt theorem) {
157
            nil =>
158
            pick-any x y
159
              assume i := (ordered (x :: nil) & y in nil)
160
                let {not-in := (!chain-> [true ==> (~ y in nil) [in.empty]])}
                (!from-complements (x \langle E y \rangle (y in nil) not-in)
162
163
          | (z :: M) =>
            let {ind-hyp := (forall ?x ?y .
164
                                ordered (?x :: M) & ?y in M
165
                                ==> ?x <E ?y);
166
                  goal := (forall ?x ?y .
167
                            ordered (?x :: (z :: M)) &
168
                            ?y in (z :: M)
169
170
                             ==> x <E ?y);
                  transitive := (!prove <EL.left-transitive)}</pre>
171
            conclude goal
172
173
              pick-any x y
                assume (ordered (x :: (z :: M)) & y in (z :: M))
174
                   let {B1 := (x <E z & z <EL M & (ordered M));</pre>
                        B2 := (!chain->
176
                               [(ordered (x :: (z :: M)))
177
178
                             ==> (x <EL (z :: M) & ordered (z :: M))
                                                    [nonempty]
179
                            ==> (x < EL (z :: M) \& z < EL M \& ordered M)
180
181
                                                    [nonemptv]
                            ==> B1
182
                                                    [<EL.nonempty]]);
                        B3 := (!chain->
183
                               [B1 ==> (ordered M) [prop-taut]]);
184
                        B4 := (!chain->
                               [B1
186
187
                             ==> (x < E z \& z < EL M)
                                                        [prop-taut]
                            ==> (x <EL M)
188
                                                         [transitive]
                            ==> (x <EL M & ordered M) [augment]
189
                            ==> (ordered (x :: M)) [nonempty]]);
190
                        B4 := (!chain->
191
192
                                [(y in (z :: M))]
                                ==> (y = z | y in M) [in.nonempty]])}
193
                   (!cases (y = z | y in M))
194
195
                     assume (y = z)
                       (!chain-> [B1 ==> (x < E z) [left-and]
196
                                         ==> (x < E y) [(y = z)])
197
                     (!chain
198
                      [(y in M)
199
200
                       ==> (ordered (x :: M) & y in M) [augment]
                       ==> (x < E y)
                                                            [ind-hyp]]))
201
202
       | (val-of cons) =>
203
          pick-any L x
            let {A1 := (ordered L);
205
                  A2 := (forall ?y . ?y in L ==> x < E ?y);
206
                 BAIBF := (!prove <EL.before-all-implies-before-first) }</pre>
207
            assume (A1 & A2)
208
```

```
(!chain-> [A2 ==> (x < EL L)]
                                                              [BAIBF]
                             ==> ((x <EL L) & A1)
                                                              [augment]
210
                             ==> (ordered (x :: L))
                                                             [nonempty]])
        | (val-of append) =>
212
         by-induction (adapt theorem) {
213
214
            nil =>
             conclude (forall ?M .
215
                         ordered nil & ordered ?M &
                         (forall ?x ?y . ?x in nil & ?y in ?M ==> ?x <E ?y)
217
                         ==> ordered (nil join ?M))
218
219
               pick-anv M
                 assume (ordered nil & ordered M &
220
                          (forall ?x ?y . ?x in nil & ?y in M ==> ?x <E ?y))
                    (!chain->
222
223
                     [(ordered M)
                      ==> (ordered (nil join M)) [join.left-empty]])
224
          | (z :: L:(List 'S)) =>
225
            let {ind-hyp :=
                    (forall ?M .
227
                       (ordered L) & (ordered ?M) &
228
                       (forall ?x ?y . ?x in L & ?y in ?M ==> ?x <E ?y)
229
                       ==> (ordered (L join ?M)));
230
                 goal :=
231
                     (forall ?M .
232
233
                       ordered (z :: L) & ordered ?M &
                       (forall ?x ?y . ?x in (z :: L) & ?y in ?M ==> ?x <E ?y)
234
                       ==> (ordered ((z :: L) join ?M)));
235
236
                 OLT := (!prove tail);
                 ELA := (!prove <EL.append) }</pre>
237
            pick-any M:(List 'S)
238
              let {A1 := (ordered (z :: L));
239
                   A2 := (ordered M);
                   A3 := (forall ?x ?y.
241
                            ?x in (z :: L) & ?y in M ==> ?x <E ?y)}
242
243
              assume (A1 & A2 & A3)
                let {C1 := (!chain-> [A1 ==> (ordered L)
                                                             [OLT11):
244
                      C2 := conclude (forall ?x ?y .
                                        ?x in L & ?y in M ==> ?x <E ?y)
246
                              pick-any x y
247
                                 assume A4 := (x in L \& y in M)
248
                                   (!chain->
249
250
                                    [A4 ==> (x in (z :: L) & y in M)
                                                          [in.tail]
251
                                        ==> (x <E y)
252
                                                           [A3]]);
                     C3 := (!chain->
253
                             [(ordered L)
254
255
                              ==> (ordered L &
                                   ordered M & C2)
                                                         [augment]
256
257
                              ==> (ordered (L join M)) [ind-hyp]]);
                     C4 := conclude (z <EL M)
258
                               (!two-cases
259
                               assume (M = nil)
260
                                  (!chain->
261
                                   [true ==> (z <EL nil) [<EL.empty]</pre>
262
                                        ==> (z <EL M) [(M = nil)]])
263
                                assume (M =/= nil)
264
                                  let {D1 := conclude (z in (z :: L))
265
                                               (!chain->
266
267
                                                [(z = z)
                                                 ==> (z = z | z in L) [alternate]
268
                                                 ==> (z in (z :: L))
270
                                                                  [in.nonempty]]);
                                       D2 := (exists ?u ?P . M = (?u :: ?P));
271
                                       D3 := conclude D2
272
                                                (!chain->
273
                                                 [true
                                                  ==> (M = nil \mid D2)
275
276
                                                  [(datatype-axioms "List")]
                                                  ==> (M =/= nil &
277
                                                        (M = nil \mid D2)) [augment]
278
```

```
==> D2
                                                                 [prop-taut]])}
                                pick-witnesses u P for D2
280
                                   (!chain->
282
                                    [true
                                     ==> (u in (u :: P))
                                                             [in.head]
283
                                     ==> (u in M)
284
                                                             [(M = u :: P)]
                                     ==> (z in (z :: L) & u in M) [augment]
285
                                     ==> (z < E u)
                                                             [A3]
                                     ==> (z <EL (u :: P)) [<EL.nonempty]
287
                                     ==> (z <EL M) [(M = u :: P)]]));
288
                     OLH := (!prove head)}
289
                conclude (ordered ((z :: L) join M))
290
                  (!chain->
                   ſA1
292
                    ==> (z <EL L)
293
                                                        [OLH]
                    ==> ((z <EL L) & C4)
294
                                                        [augment]
                    ==> (z <EL (L join M))
                                                       [ELA]
295
                    ==> ((z <EL (L join M)) & C3)
                                                       [augment]
                    ==> (ordered (z :: (L join M))) [nonempty]
297
                    ==> (ordered ((z :: L) join M))
                                                      [join.left-nonempty]])
298
299
       | (val-of append-2) =>
300
301
         by-induction (adapt theorem) {
           nil => pick-any M
302
303
                    assume A := (ordered (nil join M))
                     let {goal := (ordered nil & ordered M);
304
                          B := (!chain->
305
306
                                [true ==> (ordered nil)
                                             [empty]])}
307
                     (!chain-> [A ==> (ordered M) [join.left-empty]
308
                                  ==> goal
                                                       [augment]])
309
         | (x :: L) =>
311
           pick-any M
             assume A := (ordered (x :: L) join M)
312
313
             let {goal := (ordered (x :: L) & ordered M);
                   ind-hyp := (forall ?M .
314
                                (ordered (L join ?M)) ==>
                                (ordered L & ordered ?M));
316
                   ELA := (!prove <EL.append-2) }</pre>
317
318
              (!chain->
               [A ==> (ordered (x :: (L join M)))     [join.left-nonempty]
319
                  ==> (x < EL (L join M) & ordered (L join M)) [nonempty]
                  ==> (x <EL L & ordered (L join M))
                                                                [ELA]
321
                  ==> (x <EL L & ordered L & ordered M)
                                                                 [ind-hyp]
322
                  ==> ((x <EL L & ordered L) & ordered M)
323
                                                                 [prop-taut]
                  ==> goal
                                                                 [nonempty]])
324
326
328 (add-theorems theory | {theorems := proofs}|)
329 } # ordered
330 } # SWO
```