

## lib/main/transitive-closure.ath

```

1 #.....
2 #
3 # Transitive Closure
4
5 load "order"
6 load "nat-plus"
7
8 module Transitive-Closure {
9   open Irreflexive
10  open Strict-Partial-Order
11  overload + N.+
12  declare R+, R*: (S) [S S] -> Boolean
13  declare R**: (S) [N S S] -> Boolean
14  define [x y z y' m n] := [?x:'S ?y:'S ?z:'S ?y':'S ?m:N ?n:N]
15  define R**--zero :=
16    (forall x y . (R** zero x y) <==> x = y)
17  define R**--nonzero :=
18    (forall x n y .
19      (R** (S n) x y) <==> (exists z . (R** n x z) & z R y))
20  define R+-definition :=
21    (forall x y . x R+ y <==> (exists n . (R** (S n) x y)))
22  define R*-definition :=
23    (forall x y . x R* y <==> (exists n . (R** n x y)))
24  define theory :=
25    (make-theory
26      [Irreflexive.theory
27       (adapt-theory Strict-Partial-Order.theory |[R := R+]|)]
28      [R**--zero R**--nonzero R+-definition R*-definition])
29  define R**--sum :=
30    (forall n m x y z .
31      (R** m x y) & (R** n y z) ==> (R** (m + n) x z))
32  define RR+--inclusion := (forall x y . x R y ==> x R+ y)
33  define R+R*--inclusion := (forall x y . x R+ y ==> x R* y)
34  define R+-lemma :=
35    (forall x y .
36      x R+ y <==> x R y |
37      (exists y' . x R+ y' & y' R y))
38  define R*-lemma := (forall x y . x R* y <==> x = y | x R+ y)
39  define R*-Reflexive := (forall x . x R* x)
40  define TC-Transitivity :=
41    (forall x y z . x R+ y & y R+ z ==> x R+ z)
42  define TC-Transitivity1 :=
43    (forall x y z . x R+ y & y R z ==> x R+ z)
44  define TC-Transitivity2 :=
45    (forall x y z . x R y & y R z ==> x R+ z)
46  define TC-Transitivity3 :=
47    (forall x y z . x R* y & y R* z ==> x R* z)
48  define theorems := [R**--sum TC-Transitivity RR+--inclusion R+R*--inclusion
49                    R+-lemma R*-lemma R*-Reflexive TC-Transitivity1
50                    TC-Transitivity2 TC-Transitivity3]
51  define proofs :=
52    method (theorem adapt)
53      let {[get prove chain-> chain<-] := (proof-tools adapt theory);
54          [R R+ R* R**] := (adapt [R R+ R* R**])}
55      match theorem {
56        (val-of R**--sum) =>
57          by-induction (adapt theorem) {
58            zero =>
59              pick-any m x y z
60              let {A1 := (R** m x y);
61                  A2 := (R** zero y z)}
62              assume (A1 & A2)
63              let {B := (!chain-> [A2 ==> (y = z) [R**--zero]])}
64              (!chain->
65                [A1 ==> (R** m x z) [B]
66                 ==> (R** (m + zero) x z) [N.Plus.right-zero]])
65          | (S n) =>
66            let {ind-hyp := (forall ?m ?x ?y ?z .

```

```

69                                     (R** ?m ?x ?y) & (R** n ?y ?z) ==>
70                                     (R** (?m + n) ?x ?z))}
71 pick-any m x y z
72   let {A1 := (R** m x y);
73         A2 := (R** (S n) y z)}
74   assume (A1 & A2)
75   let {B := (!chain->
76             [A2 ==> (exists ?y' . (R** n y ?y') & ?y' R z)
77                   [R**--nonzero]])}
78   pick-witness y' for B
79     let {B-w1 := (R** n y y');
80           B-w2 := (y' R z)}
81     (!chain->
82       [(A1 & B-w1)
83         ==> (R** (m + n) x y') [ind-hyp]
84         ==> ((R** (m + n) x y') & B-w2) [augment]
85         ==> (exists ?y' . (R** (m + n) x ?y') & ?y' R z)
86               [existence]
87         ==> (R** (S (m + n)) x z) [R**--nonzero]
88         ==> (R** (m + (S n)) x z) [N.Plus.right-nonzero]])
89   }
90 | (val-of R*-Reflexive) =>
91 let {sort := (sort-of (first (qvars-of (adapt theorem))))}
92 pick-any x:sort
93   (!chain->
94     [(x = x)
95       ==> (R** zero x x) [R**--zero]
96       ==> (exists n . (R** n x x)) [existence]
97       ==> (x R* x) [R*-definition]])
98 | (val-of TC-Transitivity) =>
99 pick-any x y z
100 let {A1 := (x R+ y);
101       A2 := (y R+ z)}
102 assume (A1 & A2)
103 let {B1 := (!chain->
104             [A1 ==> (exists m . (R** (S m) x y))
105                   [R+-definition]]);
106       B2 := (!chain->
107             [A2 ==> (exists n . (R** (S ?n) y z))
108                   [R+-definition]]);
109       _ := (!prove R**--sum)}
110 pick-witness m for B1 B1-w
111 pick-witness n for B2 B2-w
112   (!chain->
113     [(B1-w & B2-w)
114       ==> (R** ((S m) + (S n)) x z) [R**--sum]
115       ==> (R** (S (m + (S n))) x z) [N.Plus.left-nonzero]
116       ==> (exists ?k . (R** (S ?k) x z)) [existence]
117       ==> (x R+ z) [R+-definition]])
118 | (val-of RR+-inclusion) =>
119 pick-any x y
120   (!chain
121     [(x R y)
122       ==> (x = x & x R y) [augment]
123       ==> ((R** zero x x) & x R y) [R**--zero]
124       ==> (exists ?x' . (R** zero x ?x') & ?x' R y) [existence]
125       ==> (R** (S zero) x y) [R**--nonzero]
126       ==> (exists ?k . (R** (S ?k) x y)) [existence]
127       ==> (x R+ y) [R+-definition]])
128 | (val-of R+-lemma) =>
129 pick-any x y
130   (!equiv
131     assume A := (x R+ y)
132     let {B := (!chain->
133             [A ==> (exists ?k . (R** (S ?k) x y)) [R+-definition]])}
134     pick-witness k for B B-w
135     let {C := (!chain->
136             [B-w ==> (exists ?x' . (R** k x ?x') & ?x' R y)
137                   [R**--nonzero]])}
136     pick-witness x' for C C-w

```

```

139      (!two-cases
140      assume D := (k = zero)
141      let {E :=
142          (!chain->
143              [C-w ==> ((R** zero x x') & x' R y)      [D]
144                  ==> (R** zero x x')                [left-and]
145                  ==> (x = x')                       [R**-zero])]}
146          (!chain->
147              [C-w ==> (x' R y)                        [right-and]
148                  ==> (x R y)                        [(x = x')]
149                  ==> (x R y | (exists ?y' . x R+ ?y' & ?y' R y))
150                  [alternate])])
151      assume D := (k /= zero)
152      let {E :=
153          (!chain->
154              [D ==> (exists ?k' . k = (S ?k'))      [N.nonzero-S])]}
155          pick-witness k' for E E-w
156          let {F := (!chain-> [C-w ==> (x' R y)      [right-and])]}
157          (!chain->
158              [C-w ==> ((R** (S k') x x') & x' R y) [E-w]
159                  ==> (R** (S k') x x')          [left-and]
160                  ==> (exists ?k' . (R** (S ?k') x x'))
161                  [existence]
162                  ==> (x R+ x')                  [R+-definition]
163                  ==> (x R+ x' & F)              [augment]
164                  ==> (exists ?x' . x R+ ?x' & ?x' R y) [existence]
165                  ==> (x R y | (exists ?x' . x R+ ?x' & ?x' R y))
166                  [alternate])])
167      assume A := (x R y | (exists ?y' . x R+ ?y' & ?y' R y))
168      let {RRI := (!prove RR+-inclusion)}
169      (!cases A
170      (!chain [(x R y) ==> (x R+ y)      [RRI]])
171      assume B := (exists ?y' . x R+ ?y' & ?y' R y)
172      pick-witness y' for B B-w
173      let {C :=
174          (!chain->
175              [B-w ==> (x R+ y')          [left-and]
176                  ==> (exists ?k . (R** (S ?k) x y')) [R+-definition])]}
177          pick-witness k for C C-w
178          (!chain->
179              [C-w ==> ((R** (S k) x y') & y' R y) [augment]
180                  ==> (exists ?y' . (R** (S k) x ?y') & ?y' R y)
181                  [existence]
182                  ==> (R** (S (S k)) x y)        [R**-nonzero]
183                  ==> (exists ?k . (R** (S ?k) x y)) [existence]
184                  ==> (x R+ y)                    [R+-definition])]))
185      | (val-of R*-lemma) =>
186      let {sort := (sort-of (first (qvars-of (adapt theorem))))}
187      pick-any x:sort y:sort
188      (!equiv
189      assume A := (x R* y)
190      let {B := (!chain->
191          [A ==> (exists ?n . (R** ?n x y))      [R*-definition])]}
192          pick-witness n for B B-w
193          (!two-cases
194          assume C1 := (n = zero)
195          (!chain->
196              [B-w ==> (R** zero x y)          [C1]
197                  ==> (x = y)                [R**-zero]
198                  ==> (x = y | x R+ y)      [alternate])])
199          assume C2 := (n /= zero)
200          let {D := (!chain-> [C2 ==> (exists ?m . n = S ?m)
201              [N.nonzero-S])]}
202              pick-witness m for D D-w
203              (!chain->
204                  [B-w ==> (R** (S m) x y) [D-w]
205                      ==> (exists ?m . (R** (S ?m) x y)) [existence]
206                      ==> (x R+ y)                    [R+-definition]
207                      ==> (x = y | x R+ y) [alternate])])
208          assume A := (x = y | x R+ y)

```

```

209      (!cases A
210        assume A1 := (x = y)
211        (!chain->
212          [A1 ==> (R** zero x y) [R**-zero]
213            ==> (exists ?n . (R** ?n x y)) [existence]
214            ==> (x R* y) [R*-definition]])
215        assume A2 := (x R+ y)
216        let {B :=
217          (!chain->
218            [A2 ==> (exists ?n . (R** (S ?n) x y))
219              [R+-definition]])}
220        pick-witness n for B B-w
221        (!chain->
222          [B-w ==> (exists ?k . (R** ?k x y)) [existence]
223            ==> (x R* y) [R*-definition]]))
224  | (val-of R+R+-inclusion) =>
225  let {R*L := (!prove R*-lemma)}
226  pick-any x y
227  (!chain
228    [(x R+ y)
229     ==> (x = y | x R+ y) [alternate]
230     ==> (x R* y) [R*L]])
231  | (val-of TC-Transitivity1) =>
232  pick-any x y z
233  let {A1 := (x R+ y);
234        A2 := (y R z);
235        R+L := (!prove R+-lemma)}
236  assume (A1 & A2)
237  (!chain->
238    [(A1 & A2)
239     ==> (exists ?y . x R+ ?y & ?y R z) [existence]
240     ==> (x R z | (exists ?y . x R+ ?y & ?y R z))
241           [alternate]
242     ==> (x R+ z) [R+L]])
243  | (val-of TC-Transitivity2) =>
244  pick-any x y z
245  let {A1 := (x R y);
246        A2 := (y R z);
247        R+-transitive := ((renaming {|R := R+|}) transitive);
248        RR+I := (!prove RR+-inclusion)}
249  assume (A1 & A2)
250  (!chain->
251    [A1 ==> (x R+ y) [RR+I]
252     ==> (x R+ y & A2) [augment]
253     ==> (x R+ y & y R+ z) [RR+I]
254     ==> (x R+ z) [R+-transitive]])
255  | (val-of TC-Transitivity3) =>
256  let {sort := (sort-of (first (qvars-of (adapt theorem))))}
257  pick-any x:sort y:sort z:sort
258  let {A1 := (x R* y);
259        A2 := (y R* z);
260        RRI := (!prove R+R+-inclusion);
261        R*L := (!prove R*-lemma)}
262  assume (A1 & A2)
263  let {B1 := (!chain->
264            [A1 ==> (x = y | x R+ y) [R*L]]);
265        B2 := (!chain->
266            [A2 ==> (y = z | y R+ z) [R*L]])}
267  (!cases B1
268    assume C1 := (x = y)
269    (!cases B2
270      assume D1 := (y = z)
271      (!chain->
272        [x = y [C1] = z [D1]
273         ==> (x = z | x R+ z) [alternate]
274         ==> (x R* z) [R*L]])
275      assume D2 := (y R+ z)
276      (!chain->
277        [D2 ==> (x R+ z) [C1]
278         ==> (x R* z) [RRI]]))

```

```
279     assume C2 := (x R+ y)
280     (!cases B2
281       assume D1 := (y = z)
282       (!chain->
283         [C2 ==> (x R+ z)      [D1]
284          ==> (x R* z)         [RRI]])
285       assume D2 := (y R+ z)
286       (!chain->
287         [D2 ==> (C2 & D2)      [augment]
288          ==> (x R+ z)         [TC-Transitivity]
289          ==> (x = z | x R+ z) [alternate]
290          ==> (x R* z)         [R*L]))))
291   }
292 }
293 (add-theorems theory |{theorems := proofs}|)
294 } # close module Transitive-Closure
```