

lib/main/order.ath

```

1 # Abstract-level order concepts and theorems
2
3 #.....
4 # Strict Partial Order
5
6 module Binary-Relation {
7   declare R, R': (T) [T T] -> Boolean
8   define [x y z] := [?x ?y ?z]
9   define inverse-def := (forall x y . x R' y <==> y R x)
10  define theory := (make-theory [] [inverse-def])
11 }
12
13 module Irreflexive {
14   open Binary-Relation
15   define irreflexive := (forall x . ~ x R x)
16   define theory := (make-theory ['Binary-Relation] [irreflexive])
17   define inverse := (forall x . ~ x R' x)
18   define proof :=
19     method (theorem adapt)
20       let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
21           [R R'] := (adapt [R R'])}
22       match theorem {
23         (val-of inverse) =>
24           pick-any x
25             (!chain-> [true ==> (~ x R x) [irreflexive]
26                    ==> (~ x R' x) [inverse-def]])
27       }
28       (add-theorems theory |{inverse := proof}|)
29   }
30
31   (test-all-proofs 'Irreflexive)
32
33 module Transitive {
34   open Binary-Relation
35   define transitive := (forall x y z . x R y & y R z ==> x R z)
36   define theory := (make-theory ['Binary-Relation] [transitive])
37   define inverse := (forall x y z . x R' y & y R' z ==> x R' z)
38   define proof :=
39     method (theorem adapt)
40       let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
41           [R R'] := (adapt [R R'])}
42       match theorem {
43         (val-of inverse) =>
44           pick-any x y z
45             (!chain [(x R' y & y R' z)
46                   ==> (y R x & z R y) [inverse-def]
47                   ==> (z R y & y R x) [and-comm]
48                   ==> (z R x) [transitive]
49                   ==> (x R' z) [inverse-def]])
50       }
51
52       (add-theorems theory |{inverse := proof}|)
53   }
54
55 module Strict-Partial-Order {
56   open Irreflexive, Transitive
57   define theory := (make-theory ['Irreflexive 'Transitive] [])
58   define asymmetric := (forall x y . x R y ==> ~ y R x)
59   define implies-not-equal := (forall x y . x R y ==> x /= y)
60   define proofs :=
61     method (theorem adapt)
62       let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
63           [R R'] := (adapt [R R'])}
64       match theorem {
65         (val-of asymmetric) =>
66           pick-any x y
67             assume (x R y)
68             (!by-contradiction (~ y R x)

```

```

69         assume (y R x)
70         (!absurd
71           (!chain-> [(y R x)
72                     ==> (x R y & y R x)      [augment]
73                     ==> (x R x)              [transitive]])
74           (!chain-> [true ==> (~ x R x)        [irreflexive]]))
75 | (val-of implies-not-equal) =>
76   pick-any x y
77   assume (x R y)
78   (!by-contradiction (x /= y)
79     assume (x = y)
80     let {xRx := (!chain-> [(x R y)
81                           ==> (x R x)      [(x = y)]]);
82         -xRx := (!chain-> [true
83                           ==> (~ x R x)    [irreflexive]])}
84         (!absurd xRx -xRx))
85   }
86 (add-theorems theory |{[asymmetric implies-not-equal] := proofs}|)
87 }
88
89 #.....
90 # Preorder
91
92 module Reflexive {
93   open Binary-Relation
94   define reflexive := (forall x . x R x)
95   define theory := (make-theory ['Binary-Relation] [reflexive])
96
97   define inverse := (forall x . x R' x)
98   define proof :=
99     method (theorem adapt)
100     let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
101          [R R'] := (adapt [R R'])}
102     match theorem {
103       (val-of inverse) =>
104         pick-any x
105         (!chain-> [true ==> (x R x)  [reflexive]
106                  ==> (x R' x) [inverse-def]])
107     }
108 (add-theorems theory |{inverse := proof}|)
109 }
110
111 module Preorder {
112   open Transitive, Reflexive
113   define theory := (make-theory ['Transitive 'Reflexive] [])
114 }
115
116 #.....
117 # (Nonstrict) Partial Order
118
119 module Antisymmetric {
120   open Binary-Relation
121   define antisymmetric := (forall x y . x R y & y R x ==> x = y)
122   define theory := (make-theory ['Binary-Relation] [antisymmetric])
123
124   define inverse := (forall x y . x R' y & y R' x ==> x = y)
125   define proof :=
126     method (theorem adapt)
127     let {[get prove chain chain-> chain<-] := (proof-tools adapt theory);
128          [R R'] := (adapt [R R'])}
129     match theorem {
130       (val-of inverse) =>
131         pick-any x y
132         (!chain [(x R' y & y R' x)
133                 ==> (y R x & x R y)  [inverse-def]
134                 ==> (x R y & y R x)  [and-comm]
135                 ==> (x = y)         [antisymmetric]])
136     }
137 (add-theorems theory |{inverse := proof}|)
138 }

```

```

139
140 module Partial-Order {
141   open Preorder, Antisymmetric
142   define theory := (make-theory ['Preorder 'Antisymmetric] [])
143 }
144
145 #.....
146 # SPO: Strict Partial Order with < instead of R, > instead of R'
147
148 module SPO {
149   declare <, >: (T) [T T] -> Boolean
150   define sm := |{Binary-Relation.R := <, Binary-Relation.R' := >}|
151   define renaming := (renaming sm)
152   define theory := (adapt-theory 'Strict-Partial-Order sm)
153 }
154
155 #.....
156 # PO: Partial Order with <= instead of R, >= instead of R'
157
158 module PO {
159
160   declare <=, >=: (T) [T T] -> Boolean
161   define sm := |{Binary-Relation.R := <=, Binary-Relation.R' := >=}|
162   define renaming := (renaming sm)
163   define theory := (adapt-theory 'Partial-Order sm)
164 }
165
166 #.....
167 # Show that if we start with SPO.theory and add a definition of <=, we
168 # can derive the axioms of PO.theory as theorems of SPO.theory.
169
170 module PO-from-SPO {
171
172   define [x y z] := [?x ?y ?z]
173
174   define [< > <= >=] := [SPO.< SPO.> PO.<= PO.>=]
175
176   define <=-definition := (forall x y . x <= y <=> x < y | x = y)
177   define >=-definition := (forall x y . x >= y <=> x > y | x = y)
178
179   (add-axioms 'SPO [<=-definition >=-definition])
180
181   define implied-by-less := (forall x y . x < y ==> x <= y)
182   define implied-by-equal := (forall x y . x = y ==> x <= y)
183   define implies-not-reverse := (forall x y . x <= y ==> ~ y < x)
184   define PO-inverse := (forall x y . x >= y <=> y <= x)
185   define PO-reflexive := (forall x . x <= x)
186   define PO-transitive := (forall x y z . x <= y & y <= z ==> x <= z)
187   define PO-antisymmetric := (forall x y . x <= y & y <= x ==> x = y)
188
189   define theorems := [<=-definition implied-by-less implied-by-equal
190                     implies-not-reverse PO-inverse PO-reflexive
191                     PO-antisymmetric PO-transitive]
192
193   define proofs :=
194     method (theorem adapt)
195       let {adapt := (o adapt SPO.renaming);
196           [get prove chain chain-> chain<-] := (proof-tools adapt SPO.theory);
197           [< > <= >=] := (adapt [< > <= >=]);
198           inverse-def := Strict-Partial-Order.inverse-def;
199           irreflexive := Strict-Partial-Order.irreflexive;
200           transitive := Strict-Partial-Order.transitive;
201           asymmetric := (!prove Strict-Partial-Order.asymmetric)}
202       match theorem {
203         (val-of implied-by-less) =>
204           pick-any x y
205             (!chain [(x < y) ==> (x < y | x = y) [alternate]
206                   ==> (x <= y) [=<=-definition]])
207       | (val-of implied-by-equal) =>
208         pick-any x y

```

```

209     (!chain [(x = y) ==> (x < y | x = y) [alternate]
210             ==> (x <= y) [<=definition]])
211 | (val-of implies-not-reverse) =>
212   pick-any x y
213   assume A := (x <= y)
214   let {B := (!chain-> [A ==> (x < y | x = y) [<=definition]])}
215     (!cases B
216       (!chain [(x < y) ==> (~ y < x) [asymmetric]])
217       assume (x = y)
218       (!by-contradiction (~ y < x)
219         assume (y < x)
220         let {is := (!chain-> [(y < x) ==> (y < y) [(x = y)]];
221           is-not := (!chain-> [true ==> (~ y < y)
222                               [irreflexive]])}
223         (!absurd is is-not)))
224 | (val-of PO-inverse) =>
225   pick-any x y
226   (!chain [(x >= y) <==> (x > y | x = y) [>=definition]
227           <==> (y < x | y = x) [inverse-def sym]
228           <==> (y <= x) [<=definition]])
229 | (val-of PO-reflexive) =>
230   pick-any x
231   let {IBE := (!prove implied-by-equal)}
232     (!chain-> [(x = x) ==> (x <= x) [IBE]])
233 | (val-of PO-antisymmetric) => (!force (adapt theorem))
234 | (val-of PO-transitive) => (!force (adapt theorem))
235 }
236
237 (add-theorems SPO.theory |{theorems := proofs}|)
238 }
239
240 extend-module PO-from-SPO {
241
242   define proofs :=
243     method (theorem adapt)
244       let {adapt := (o adapt SPO.renaming);
245         [get prove chain chain-> chain<-] := (proof-tools adapt SPO.theory);
246         [< <=] := (adapt [< <=]);
247         irreflexive := Strict-Partial-Order.irreflexive;
248         transitive := Strict-Partial-Order.transitive;
249         asymmetric := (!prove Strict-Partial-Order.asymmetric)}
250     match theorem {
251       (val-of PO-antisymmetric) =>
252         pick-any x y
253         assume (x <= y & y <= x)
254         let {disj1 := (!chain->
255           [(x <= y) ==> (x < y | x = y) [<=definition]];
256           disj2 := (!chain->
257             [(y <= x) ==> (y < x | y = x) [<=definition]])}
258         (!cases disj1
259           assume (x < y)
260           (!cases disj2
261             assume (y < x)
262             (!from-complements (x = y)
263               (y < x)
264               (!chain-> [(x < y) ==> (~ y < x) [asymmetric]]))
265             assume (y = x)
266             (!sym (y = x)))
267             assume (x = y)
268             (!claim (x = y)))
269         | (val-of PO-transitive) =>
270         pick-any x y z
271         assume (x <= y & y <= z)
272         let {disj1 := (!chain->
273           [(x <= y) ==> (x < y | x = y) [<=definition]];
274           disj2 := (!chain->
275             [(y <= z) ==> (y < z | y = z) [<=definition]])}
276         by-less := (!prove implied-by-less);
277         by-equal := (!prove implied-by-equal)}
278         (!cases disj1

```

```

279     assume (x < y)
280     (!cases disj2
281       assume i := (y < z)
282       (!chain->
283         [i ==> (x < y & y < z)           [augment]
284         ==> (x < z)                       [transitive]
285         ==> (x <= z)                      [by-less]])
286       assume ii := (y = z)
287       (!chain-> [(x < y) ==> (x < z)      [ii]
288                 ==> (x <= z)           [by-less]]))
289     assume (x = y)
290     (!cases disj2
291       assume i := (y < z)
292       (!chain-> [i ==> (x < z)           [(x = y)]
293                 ==> (x <= z)           [by-less]])
294       assume ii := (y = z)
295       (!chain-> [x --> y                 [(x = y)]
296                 --> z                   [ii]
297                 ==> (x <= z)           [by-equal]]))
298   }
299
300   (add-theorems SPO.theory |{[PO-antisymmetric PO-transitive] := proofs}|)
301 }
302
303 #.....
304 # SWO: Strict Weak Order, a refinement of SPO
305
306 extend-module SPO {
307   declare E: (T) [T T] -> Boolean [100]
308   define E-definition := (forall x y . x E y <==> ~ x < y & ~ y < x)
309   (add-axioms theory [E-definition])
310 }
311
312 module SWO {
313   open SPO
314
315   define E-transitive := (forall x y z . x E y & y E z ==> x E z)
316
317   define theory := (make-theory ['SPO] [E-transitive])
318
319   define E-reflexive := (forall x . x E x)
320   define E-symmetric := (forall x y . x E y ==> y E x)
321   define <-E-transitive-1 := (forall x y z . x < y & y E z ==> x < z)
322   define <-E-transitive-2 := (forall x y z . x < y & x E z ==> z < y)
323   define not-<-property := (forall x y . ~ x < y ==> y < x | y E x)
324   define <-transitive-not-1 := (forall x y z . x < y & ~ z < y ==> x < z)
325   define <-transitive-not-2 := (forall x y z . x < y & ~ x < z ==> z < y)
326   define <-transitive-not-3 := (forall x y z . ~ y < x & y < z ==> x < z)
327   define not-<-is-transitive :=
328     (forall x y z . ~ x < y & ~ y < z ==> ~ x < z)
329
330   define <-E-theorems :=
331     [E-reflexive E-symmetric <-E-transitive-1 <-E-transitive-2
332     not-<-property <-transitive-not-1 <-transitive-not-2
333     <-transitive-not-3 not-<-is-transitive]
334
335   define ren := (get-renaming 'SPO)
336
337   define <-E-proofs :=
338     method (theorem adapt)
339       let {adapt := (o adapt SPO.renaming);
340           [get prove chain chain-> chain<-] := (proof-tools adapt theory);
341           E := lambda (x y) (adapt (x E y));
342           < := lambda (x y) (adapt (x < y));
343           irreflexive := Strict-Partial-Order.irreflexive;
344           transitive := Strict-Partial-Order.transitive;
345           asymmetric := Strict-Partial-Order.asymmetric}
346     match theorem {
347       (val-of E-reflexive) =>
348       pick-any x

```

```

349         (!chain-> [true
350                   ==> (~ x < x)                [irreflexive]
351                   ==> (~ x < x & ~ x < x)      [augment]
352                   ==> (x E x)                  [E-definition]])
353     | (val-of E-symmetric) =>
354       pick-any x y
355         assume (x E y)
356           (!chain-> [(x E y)
357                   ==> (~ x < y & ~ y < x)      [E-definition]
358                   ==> (~ y < x & ~ x < y)      [and-comm]
359                   ==> (y E x)                  [E-definition]])
360     | _ => (!force (adapt theorem))
361   }
362
363   (add-theorems theory |{<-E-theorems := <-E-proofs}|)
364 } # close module SWO
365
366 extend-module SWO {
367   declare <E: (T) [T T] -> Boolean
368   define <E-definition := (forall x y . x <E y <==> ~ y < x)
369   (add-axioms theory [<E-definition])
370 }
371
372 # Show that <E is a preorder:
373
374 extend-module SWO {
375
376   define <E-reflexive := (forall x . x <E x)
377   define <E-transitive := (forall x y z . x <E y & y <E z ==> x <E z)
378   define theorems := [<E-reflexive <E-transitive]
379
380   define proofs :=
381     method (theorem adapt)
382       let {adapt := (o adapt SPO.renaming);
383           [get prove chain chain-> chain<-] := (proof-tools adapt theory);
384           < := lambda (x y) (adapt (x < y));
385           <E := lambda (x y) (adapt (x <E y));
386           irreflexive := Strict-Partial-Order.irreflexive;
387           transitive := Strict-Partial-Order.transitive}
388       match theorem {
389         (val-of <E-reflexive) =>
390           pick-any x
391             (!chain-> [true ==> (~ x < x)          [irreflexive]
392                     ==> (x <E x)                [<E-definition]])
393         | (val-of <E-transitive) =>
394           let {transitive := (!prove not-<-is-transitive)}
395             pick-any x y z
396               (!chain [(x <E y & y <E z)
397                       ==> (~ y < x & ~ z < y)    [<E-definition]
398                       ==> (~ z < x)              [transitive]
399                       ==> (x <E z)                [<E-definition]])
400             }
401
402     (add-theorems theory |{theorems := proofs}|)
403 } # close module SWO
404
405 #.....
406 # STO: Strict Total Order theory
407
408 module STO {
409   open SWO
410
411   define strict-trichotomy := (forall x y . ~ x < y & ~ y < x ==> x = y)
412
413   define theory := (make-theory ['SWO] [strict-trichotomy])
414
415   define E-iff-equal := (forall x y . x E y <==> x = y)
416 } # close module STO
417
418 extend-module STO {

```

```

419 define proof :=
420 method (theorem adapt)
421   let {adapt := (o adapt SPO.renaming);
422       [get prove chain chain-> chain<-] := (proof-tools adapt theory);
423       E := lambda (x y) (adapt (x E y));
424       < := lambda (x y) (adapt (x < y))}
425   match theorem {
426     (val-of E-iff-equal) =>
427     pick-any x y
428     (!equiv
429       (!chain [(x E y)
430               ==> (~ x < y & ~ y < x)      [E-definition]
431               ==> (x = y)                  [strict-trichotomy]])
432       assume (x = y)
433       (!chain-> [true
434                 ==> (~ x < x)              [Strict-Partial-Order.irreflexive]
435                 ==> (~ x < x & ~ x < x)    [augment]
436                 ==> (x E x)               [E-definition]
437                 ==> (x E y)               [(x = y)]])])
438   }
439
440 (add-theorems theory |{E-iff-equal := proof}|)
441 } # close module STO

```