

lib/basic/sets.ath

```

1 load "nat-minus"
2
3 module Set {
4
5 structure (Set S) := null | (insert S (Set S))
6
7 define (lst->set L) :=
8   (let ((f ((from-list "(Set.Set 'S)" id) lst->set)))
9     (f L))
10
11 define (lst->set L) :=
12   match L {
13     [] => null
14     | (list-of x rest) => (insert (lst->set x) (lst->set rest))
15     | _ => L
16   }
17
18 (lst->set [1 2 3])
19
20 define (set->lst S) :=
21   (let ((f ((to-list "(Set.Set 'T)" dedup) set->lst)))
22     (f S))
23
24 define (set->lst-aux s) :=
25   match s {
26     null => []
27     | (insert x rest) => (add (set->lst-aux x) (set->lst-aux rest))
28     | _ => s
29   }
30
31 define (set->lst s) :=
32   match (set->lst-aux s) {
33     (some-list L) => (dedup L)
34     | _ => s
35   }
36
37 #define (set->lst S) :=
38 # (let ((f ((to-list "(Set 'T)" dedup) set->lst)))
39 #   (f S))
40
41 #define set->lst := ((to-list "(Set.Set 'T)" dedup) id)
42
43 (set->lst (1 insert 2 insert 1 insert 3 insert null))
44
45 expand-input insert [id lst->set]
46
47 define ++ := insert
48
49 (1 ++ [2 3])
50
51 set-precedence ++ 210
52
53 define [x y z h h' a b s s' t t' s1 s2 s3 A B C D E U] :=
54   [?x ?y ?z ?h ?h' ?a ?b ?s:(Set 'T1) ?s':(Set 'T2)
55     ?t:(Set 'T3) ?t':(Set 'T4) ?s1:(Set 'T5)
56     ?s2:(Set 'T6) ?s3:(Set 'T7) ?A:(Set 'T8)
57     ?B:(Set 'T9) ?C:(Set 'T10)
58     ?D:(Set 'T10) ?E:(Set 'T11) ?U]
59
60 declare in: (T) [T (Set T)] -> Boolean [[id lst->set]]
61
62 assert* in-def :=
63   [(~ _ in null)
64     (x in h ++ t <==> x = h | x in t)]
65
66 (eval 23 in [1 5 23 98])
67
68 (eval 23 in [1 5 98])

```

```

69
70 (eval 5 in [])
71
72 (eval 5 in [5])
73
74 conclude null-characterization := (forall x . x in [] <==> false)
75   pick-any x
76     (!equiv
77       assume hyp := (x in [])
78       (!absurd hyp
79         (!chain-> [true ==> (~ x in []) [in-def]]))
80       assume false
81       (!from-false (x in [])))
82
83 conclude in-lemma-1 := (forall x A . x in x ++ A)
84   pick-any x A
85     (!chain-> [(x = x) ==> (x in x ++ A) [in-def]])
86
87
88 define NC := null-characterization
89
90 declare singleton: (T) [T] -> (Set T)
91
92 assert* singleton-axiom := (singleton x = x ++ null)
93
94 conclude singleton-characterization :=
95   (forall x y . x in singleton y <==> x = y)
96   pick-any x y
97     (!chain [(x in singleton y)
98       <==> (x in y ++ null) [singleton-axiom]
99       <==> (x = y | x in null) [in-def]
100       <==> (x = y | false) [null-characterization]
101       <==> (x = y) [prop-taut]])
102
103 define singleton-lemma := (forall x . x in singleton x)
104   pick-any x
105     (!chain-> [(x = x)
106       ==> (x in singleton x) [singleton-characterization]])
107
108 declare subset, proper-subset: (S) [(Set S) (Set S)] -> Boolean [[lst->set lst->set]]
109
110 assert* subset-def :=
111   [([] subset _)
112     (h ++ t subset A <==> h in A & t subset A)]
113
114 (eval [1 2] subset [3 2 4 1 5])
115
116 (eval [1 2] subset [3 2])
117
118 (eval [] subset [])
119
120 define subset-characterization-1 :=
121   by-induction (forall A B . A subset B ==> forall x . x in A ==> x in B) {
122     null => pick-any B
123       assume (null subset B)
124         pick-any x
125           (!chain [(x in null) ==> false [NC]
126             ==> (x in B) [prop-taut]])
127   | (A as (insert h t)) =>
128     pick-any B
129       assume hyp := (A subset B)
130       pick-any x
131         let {ih := (forall B . t subset B ==>
132           forall x . x in t ==> x in B);
133           _ := (!chain-> [hyp ==> (t subset B) [subset-def]])}
134         assume hyp' := (x in A)
135         (!cases (!chain<- [(x = h | x in t) <== hyp' [in-def]])
136           assume (x = h)
137             (!chain-> [hyp ==> (h in B) [subset-def]
138               ==> (x in B) [(x = h)]])

```

```

139         (!chain [(x in t) ==> (x in B) [ih]]))
140     }
141
142
143 define subset-characterization-2 :=
144   by-induction (forall A B . (forall x . x in A ==> x in B) ==> A subset B) {
145     null => pick-any B
146       assume (forall x . x in null ==> x in B)
147         (!chain-> [true ==> (null subset B) [subset-def]])
148   | (A as (insert h t)) =>
149     pick-any B
150     assume hyp := (forall x . x in A ==> x in B)
151     let {ih := (forall B . (forall x . x in t ==> x in B)
152               ==> t subset B);
153         goal := (A subset B);
154         ih-cond := pick-any x
155           (!chain [(x in t) ==> (x in A) [in-def]
156                  ==> (x in B) [hyp]]);
157         _ := (!chain-> [ih-cond ==> (t subset B) [ih]])}
158     (!chain-> [(h = h)
159              ==> (h in A) [in-def]
160              ==> (h in B) [hyp]
161              ==> (h in B & t subset B) [augment]
162              ==> goal [subset-def]])
163   }
164
165 conclude subset-characterization :=
166   (forall s1 s2 . s1 subset s2 <==> forall x . x in s1 ==> x in s2)
167   pick-any s1 s2
168     (!equiv (!chain [(s1 subset s2)
169                    ==> (forall x . x in s1 ==> x in s2) [subset-characterization-1]]
170              (!chain [(forall x . x in s1 ==> x in s2)
171                    ==> (s1 subset s2) [subset-characterization-2]])))
172
173 define SC := subset-characterization
174
175 define subset-intro :=
176   method (p)
177     match p {
178       (forall (some-var x) ((x in (some-term A)) ==> (x in (some-term B)))) =>
179         (!chain-> [p ==> (A subset B) [subset-characterization]])
180     }
181
182 assert* set-identity :=
183   (A = B <==> A subset B & B subset A)
184
185 (eval 1 ++ 2 ++ [] = 2 ++ 1 ++ [])
186
187 (eval 1 ++ 2 ++ 3 ++ 4 ++ [] = 3 ++ 2 ++ 1 ++ [])
188
189
190 conclude set-identity-characterization :=
191   (forall A B . A = B <==> forall x . x in A <==> x in B)
192   pick-any A:(Set 'S) B
193     (!equiv
194       assume hyp := (A = B)
195       pick-any x
196         let {_ := (!chain-> [hyp ==> (A subset B) [set-identity]]);
197             _ := (!chain-> [hyp ==> (B subset A) [set-identity]])}
198         (!chain [(x in A) <==> (x in B) [subset-characterization]])
199       assume hyp := (forall x . x in A <==> x in B)
200       let {A-subset-B := (!subset-intro
201                          pick-any x
202                            (!chain [(x in A) ==> (x in B) [hyp]]));
203           B-subset-A := (!subset-intro
204                          pick-any x
205                            (!chain [(x in B) ==> (x in A) [hyp]]));
206           p := (!both A-subset-B B-subset-A)
207         (!chain-> [p ==> (A = B) [set-identity]])}
208

```

```

209 define SIC := set-identity-characterization
210
211 define set-identity-intro :=
212   method (p1 p2)
213     match [p1 p2] {
214       [(A subset B) (B subset A)] =>
215         (!chain-> [p1 ==> (p1 & p2) [augment]
216                   ==> (A = B) [set-identity]])
217     }
218
219 define set-identity-intro-direct :=
220   method (premise)
221     match premise {
222       (forall x ((x in A) <==> (x in B))) =>
223         (!chain-> [premise ==> (A = B) [set-identity-characterization]])
224     }
225
226
227 assert* proper-subset-def :=
228   [(s1 proper-subset s2 <==> s1 subset s2 & s1 != s2)]
229
230 (eval [1 2] proper-subset [2 3 1])
231
232 (eval [1 2] proper-subset [2 1])
233
234 conclude neg-set-identity-characterization-1 :=
235   (forall s1 s2 . s1 != s2 <==> ~ s1 subset s2 | ~ s2 subset s1)
236 pick-any s1 s2
237   (!chain [(s1 != s2)
238            <==> (~ (s1 subset s2 & s2 subset s1)) [set-identity]
239            <==> (~ s1 subset s2 | ~ s2 subset s1) [prop-taut]])
240
241 conclude neg-set-identity-characterization-2 :=
242   (forall s1 s2 . s1 != s2 <==>
243     (exists x . x in s1 & ~ x in s2) |
244     (exists x . x in s2 & ~ x in s1))
245 pick-any s1 s2
246   (!chain [(s1 != s2)
247            <==> (~ s1 subset s2 | ~ s2 subset s1) [neg-set-identity-characterization-1]
248            <==> (~ (forall x . x in s1 ==> x in s2) | ~ (forall x . x in s2 ==> x in s1)) [SC]
249            <==> ((exists x . ~ (x in s1 ==> x in s2)) | (exists x . ~ (x in s2 ==> x in s1))) [qn]
250            <==> ((exists x . x in s1 & ~ x in s2) | (exists x . x in s2 & ~ x in s1)) [prop-taut]])
251
252
253 define proper-subset-characterization :=
254   (forall s1 s2 . s1 proper-subset s2 <==> s1 subset s2 & exists x . x in s2 & ~ x in s1)
255
256 conclude PSC := proper-subset-characterization
257 pick-any s1 s2
258   (!chain [(s1 proper-subset s2)
259            <==> (s1 subset s2 & s1 != s2) [proper-subset-def]
260            <==> (s1 subset s2 & ((exists x . x in s1 & ~ x in s2) |
261                               (exists x . x in s2 & ~ x in s1))) [neg-set-identity-characterization-2]
262            <==> (s1 subset s2 & (((s1 subset s2) & (exists x . x in s1 & ~ x in s2)) |
263                               (exists x . x in s2 & ~ x in s1))) [prop-taut]
264            <==> (s1 subset s2 & (((forall x . x in s1 ==> x in s2) & (exists x . x in s1 & ~ x in s2)) |
265                               (exists x . x in s2 & ~ x in s1))) [SC]
266            <==> (s1 subset s2 & ((~ (forall x . x in s1 ==> x in s2) & (exists x . x in s1 & ~ x in s2)) |
267                               (exists x . x in s2 & ~ x in s1))) [bdn]
268            <==> (s1 subset s2 & ((~ (exists x . ~ (x in s1 ==> x in s2)) & (exists x . x in s1 & ~ x in s2)) |
269                               (exists x . x in s2 & ~ x in s1))) [qn]
270            <==> (s1 subset s2 & ((~ (exists x . x in s1 & ~ x in s2) & (exists x . x in s1 & ~ x in s2)) |
271                               (exists x . x in s2 & ~ x in s1))) [prop-taut]
272            <==> (s1 subset s2 & (false | (exists x . x in s2 & ~ x in s1))) [prop-taut]
273            <==> (s1 subset s2 & (exists x . x in s2 & ~ x in s1)) [prop-taut]])
274
275
276 conclude proper-subset-lemma :=
277   (forall A B x . A subset B & x in B & ~ x in A ==> A proper-subset B)
278 pick-any A B x

```

```

279     assume h := (A subset B & x in B & ~ x in A)
280     (!chain-> [(x in B)
281               ==> (x in B & ~ x in A) [augment]
282               ==> (exists x . x in B & ~ x in A) [existence]
283               ==> (A subset B & exists x . x in B & ~ x in A) [augment]
284               ==> (A proper-subset B) [PSC]])
285
286 conclude in-lemma-2 := (forall h t . h in t ==> h ++ t = t)
287 pick-any h t
288   assume hyp := (h in t)
289   (!set-identity-intro-direct
290     pick-any x
291     (!chain [(x in h ++ t)
292              <==> (x = h | x in t) [in-def]
293              <==> (x in t | x in t) [hyp prop-taut]
294              <==> (x in t) [prop-taut]]))
295
296 conclude in-lemma-3 := (forall x h t . x in t ==> x in h ++ t)
297 pick-any x h t
298   (!chain [(x in t)
299            ==> (x = h | x in t) [alternate]
300            ==> (x in h ++ t) [in-def]])
301
302
303 conclude in-lemma-4 :=
304   (forall A x y . x in A ==> y in A <==> y = x | y in A)
305 pick-any A x y
306   assume (x in A)
307   (!equiv assume h := (y in A)
308     (!chain-> [h ==> (y = x | y in A) [alternate]])
309     assume h := (y = x | y in A)
310     (!cases h
311       (!chain [(y = x) ==> (y in A) [(x in A)]]
312       (!chain [(y in A) ==> (y in A) [claim]])))
313
314 conclude null-characterization-2 :=
315   (forall A . A = null <==> forall x . ~ x in A)
316 pick-any A
317   (!chain [(A = null)
318            <==> (forall x . x in A <==> x in null) [SIC]
319            <==> (forall x . x in A <==> false) [NC]
320            <==> (forall x . ~ x in A) [prop-taut]])
321
322 define NC-2 := null-characterization-2
323
324 conclude NC-3 :=
325   (forall A . A /= null <==> exists x . x in A)
326 pick-any A
327   (!chain [(A /= null)
328            <==> (~ forall x . ~ x in A) [NC-2]
329            <==> (exists x . ~ ~ x in A) [qn-strict]
330            <==> (exists x . x in A) [bdn]])
331
332 define (non-empty S) := (S /= null)
333
334 conclude subset-reflexivity := (forall A . A subset A)
335 pick-any A
336   (!subset-intro
337     pick-any x
338     (!chain [(x in A) ==> (x in A) [claim]]))
339
340 conclude subset-antisymmetry :=
341   (forall A B . A subset B & B subset A ==> A = B)
342 pick-any A B
343   assume hyp := (A subset B & B subset A)
344   (!set-identity-intro (A subset B) (B subset A))
345
346 conclude subset-transitivity :=
347   (forall A B C . A subset B & B subset C ==> A subset C)
348 pick-any A B C

```

```

349   assume (A subset B & B subset C)
350     (!subset-intro
351       pick-any x
352         (!chain [(x in A)
353                 ==> (x in B) [subset-characterization]
354                 ==> (x in C) [subset-characterization]]))
355
356
357 conclude subset-lemma-1 :=
358   (forall A B x . A subset B & x in B ==> x ++ A subset B)
359 pick-any A B x
360   assume hyp := (A subset B & x in B)
361   (!subset-intro
362     pick-any y
363       (!chain [(y in x ++ A)
364               ==> (y = x | y in A) [in-def]
365               ==> (y in B | y in A) [(x in B)]
366               ==> (y in B | y in B) [SC]
367               ==> (y in B) [prop-taut]]))
368
369 conclude subset-lemma-2 :=
370   (forall h t A . h ++ t subset A ==> t subset A)
371 pick-any h t A
372   assume (h ++ t subset A)
373   (!subset-intro
374     pick-any x
375       (!chain [(x in t)
376               ==> (x = h | x in t) [alternate]
377               ==> (x in h ++ t) [in-def]
378               ==> (x in A) [SC]]))
379
380
381
382 declare remove: (S) [(Set S) S] -> (Set S) [- [lst->set id]]
383
384 assert* remove-def :=
385   [(null - _ = null)
386    (h ++ t - x = t - x <== x = h)
387    (h ++ t - x = h ++ (t - x) <== x /= h)]
388
389 (eval [1 2 3 2 5] - 2)
390
391 conclude remove-characterization :=
392   (forall A x y . y in A - x <==> y in A & y /= x)
393 by-induction remove-characterization {
394   null => pick-any x y
395     (!chain [(y in null - x)
396             <==> (y in null)
397             <==> false
398             <==> (y in null & y /= x)])
399 | (A as (insert h t)) =>
400   let {IH := (forall x y . y in t - x <==> y in t & y /= x)}
401     pick-any x y
402     (!two-cases
403       assume case-1 := (x = h)
404         (!chain [(y in A - x)
405                 <==> (y in t - x) [remove-def]
406                 <==> (y in t & y /= x) [IH]
407                 <==> ((y = x | y in t) & y /= x) [prop-taut]
408                 <==> ((y = h | y in t) & y /= x) [case-1]
409                 <==> (y in A & y /= x) [in-def]])
410       assume case-2 := (x /= h)
411         let {lemma := assume (y = h)}
412           (!chain-> [case-2 ==> (y /= x) [(y = h)]])
413         (!chain [(y in A - x)
414                 <==> (y in h ++ (t - x)) [remove-def]
415                 <==> (y = h | y in t - x) [in-def]
416                 <==> (y = h | (y in t & y /= x)) [IH]
417                 <==> ((y = h | y in t) & (y = h | y /= x)) [prop-taut]
418                 <==> (y in A & (y = h | y /= x)) [in-def]

```

```

419         <==> (y in A & (y != x | y != x))           [prop-taut lemma]
420         <==> (y in A & y != x)                       [prop-taut]])
421     }
422
423 conclude remove-corollary := (forall A x . ~ x in A - x)
424 pick-any A x
425   (!by-contradiction (~ x in A - x)
426     (!chain [(x in A - x)
427       ==> (x in A & x != x) [remove-characterization]
428       ==> (x != x)         [right-and]
429       ==> (x = x & x != x) [augment]
430       ==> false           [prop-taut]]))
431
432 conclude remove-corollary-2 :=
433   (forall A x . ~ x in A ==> A - x = A)
434 pick-any A x
435   assume hyp := (~ x in A)
436   (!set-identity-intro-direct
437     pick-any y
438     (!equiv
439       (!chain [(y in A - x)
440         ==> (y in A & y != x) [remove-characterization]
441         ==> (y in A)         [left-and]])
442       assume (y in A)
443       let { _ := (!by-contradiction (y != x)
444         assume (y = x)
445           (!absurd (y in A)
446             (!chain-> [hyp ==> (~ y in A) [(y = x)]])]);
447         lemma := (!both (y in A) (y != x))}
448       (!chain-> [lemma ==> (y in A - x) ]))
449
450 conclude remove-corollary-3 :=
451   (forall A x y . x in A & y != x ==> x in A - y)
452 pick-any A x y
453   assume hyp := (x in A & y != x)
454   let { _ := (!ineq-sym (y != x))}
455     (!chain-> [hyp ==> (x in A - y) [remove-characterization]])
456
457 conclude remove-corollary-4 :=
458   (forall A x y . ~ x in A ==> ~ x in A - y)
459 pick-any A x y
460   (!chain [(~ x in A) ==> (~ x in A - y) [remove-characterization]])
461
462 conclude remove-corollary-5 :=
463   (forall A B x . A subset B & ~ x in A ==> A subset B - x)
464 pick-any A B x
465   assume h := (A subset B & ~ x in A)
466   (!subset-intro
467     pick-any y
468     assume h2 := (in y A)
469     let { _ := (!chain-> [h2 ==> (in y B) [SC]]);
470       _ := (!by-contradiction (y != x)
471         assume (y = x)
472           (!absurd (in y A)
473             (!chain-> [(~ x in A) ==> (~ y in A) [(y = x)]])]);
474       S := (!both (in y B) (y != x))}
475     (!chain-> [S ==> (y in B - x) [remove-characterization]])
476
477
478 conclude remove-corollary-6 := (forall A h t . A subset h ++ t ==> A - h subset t)
479 pick-any A:(Set.Set 'S) h:'S t:(Set.Set 'S)
480   assume hyp := (A subset h ++ t)
481   (!subset-intro
482     pick-any x
483     assume hyp' := (x in A - h)
484     let { _ := (!chain-> [hyp' ==> (x in A & x != h) [remove-characterization]]);
485       disj := (!chain-> [(x in A) ==> (x in h ++ t) [SC]
486         ==> (x = h | x in t) [in-def]])}
487     (!cases disj
488       (!chain [(x = h)

```

```

489         ==> (x = h & x != h) [augment]
490         ==> false [prop-taut]
491         ==> (x in t) [prop-taut]]]
492     (!chain [(x in t) ==> (x in t) [claim]]))
493
494
495 conclude remove-corollary-7 := (forall A x . A - x subset A)
496 pick-any A:(Set.Set 'S) x:'S
497   (!subset-intro
498     pick-any y
499     (!chain [(y in A - x)
500       ==> (y in A) [remove-characterization]]))
501
502
503 conclude remove-corollary-8 :=
504   (forall A x . x in A ==> A = x ++ (A - x))
505 pick-any A:(Set.Set 'S) x:'S
506   assume (x in A)
507   let {p1 := (!subset-intro
508     pick-any y:'S
509     assume (y in A)
510     (!two-cases
511       assume (x = y)
512       (!chain-> [true ==> (x in x ++ (A - x)) [in-lemma-1]
513         ==> (y in x ++ (A - x)) [(x = y)]]])
514       assume (x != y)
515       (!chain-> [(x != y)
516         ==> (y in A & x != y) [augment]
517         ==> (y in A - x) [remove-corollary-3]
518         ==> (y in x ++ (A - x)) [in-def]]])});
519   p2 := (!subset-intro
520     pick-any y:'S
521     assume hyp := (y in x ++ (A - x))
522     (!cases (!chain<- [(y = x | y in A - x) <== hyp [in-def]]]
523       assume (y = x)
524       (!chain-> [(y = x) ==> (y in A) [(x in A)]]])
525       assume (y in A - x)
526       (!chain-> [(y in A - x) ==> (y in A) [remove-characterization]])))
527   (!set-identity-intro p1 p2)
528
529 conclude subset-lemma-3 :=
530   (forall A t h . A subset h ++ t & h in A ==> exists B . B subset t & A = h ++ B)
531 pick-any A:(Set.Set 'S) t h:'S
532   assume hyp := (A subset h ++ t & h in A)
533   let {p := (!chain-> [(A subset h ++ t) ==> (A - h subset t) [remove-corollary-6]]])
534   (!chain-> [(h in A)
535     ==> (A = h ++ (A - h)) [remove-corollary-8]
536     ==> (p & A = h ++ (A - h)) [augment]
537     ==> (exists B . B subset t & A = h ++ B) [existence]])
538
539
540 conclude subset-lemma-4 :=
541   (forall A h t . ~ h in A & A subset h ++ t ==> A subset t)
542 pick-any A h t
543   assume hyp := (~ h in A & A subset h ++ t)
544   (!subset-intro
545     pick-any x
546     assume (x in A)
547     (!cases (!chain<- [(x = h | x in t) <== (x in h ++ t) [in-def]
548       <== (x in A) [SC]]])
549     (!chain [(x = h)
550       ==> (~ x in A) [(~ h in A)]
551       ==> (x in A & ~ x in A) [augment]
552       ==> (in x t) [prop-taut]]])
553     (!chain [(x in t) ==> (x in t) [claim]]))
554
555
556 conclude subset-lemma-5 :=
557   (forall A t h . A subset t ==> A subset h ++ t)
558 pick-any A t h

```



```

559 assume hyp := (A subset t)
560 (!subset-intro
561   pick-any x
562     (!chain [(x in A) ==> (x in t) [SC]
563             ==> (x in h ++ t) [in-def]]))
564
565 conclude subset-lemma-6 :=
566 (forall A . A subset null <==> A = null)
567 pick-any A
568 (!equiv assume (A subset null)
569         (!by-contradiction (A = null)
570         assume (A != null)
571           pick-witness x for (!chain<- [(exists x . x in A) <== (A != null) [NC-3]])
572           (!chain-> [(x in A) ==> (x in null) [SC]
573                   ==> false [NC]]))
574         assume (A = null)
575         (!chain-> [true ==> (A subset A) [subset-reflexivity]
576                 ==> (A subset null) [(A = null)]]))
577
578
579 conclude subset-lemma-7 :=
580 (forall A B x . ~ x in A & B subset A ==> ~ x in B)
581 pick-any A B x
582 assume hyp := (~ x in A & B subset A)
583 (!by-contradiction (~ x in B)
584   (!chain [(x in B) ==> (x in A) [SC]
585           ==> (x in A & ~ x in A) [augment]
586           ==> false [prop-taut]]))
587
588
589
590 declare union, intersection, diff: (S) [(Set S) (Set S)] -> (Set S) [120 [lst->set lst->set]]
591
592 define [\ / \ \] := [union intersection diff]
593
594 assert* union-def :=
595 [([] \ / s = s)
596  (h ++ t \ / s = h ++ (t \ / s))]
597
598 transform-output eval [set->lst]
599
600 (eval [1 2 3] \ / [4 5 6])
601
602 (eval [1 2] \ / [1 2])
603
604 conclude union-characterization-1 :=
605 (forall A B x . x in A \ / B ==> x in A | x in B)
606 by-induction union-characterization-1 {
607   null => pick-any B x
608     (!chain [(x in null \ / B)
609             ==> (x in B) [union-def]
610             ==> (x in null | x in B) [alternate]])
611   | (A as (h insert t)) =>
612     let {IH := (forall B x . x in t \ / B ==> x in t | x in B)}
613     pick-any B x
614     (!chain [(x in A \ / B)
615             ==> (x in h ++ (t \ / B)) [union-def]
616             ==> (x = h | x in t \ / B) [in-def]
617             ==> (x = h | x in t | x in B) [IH]
618             ==> ((x = h | x in t) | x in B) [prop-taut]
619             ==> (x in A | x in B) [in-def]])
620   }
621
622 conclude union-characterization-2 :=
623 (forall A B x . x in A | x in B ==> x in A \ / B)
624 by-induction union-characterization-2 {
625   (A as null) =>
626     pick-any B x
627     (!chain [(x in null | x in B)
628             ==> (false | x in B) [NC]

```

```

629         ==> (x in B)           [prop-taut]
630         ==> (x in null  $\vee$  B) [union-def]]
631
632 | (A as (insert h t)) =>
633   pick-any B x
634   let {IH := (forall B x . x in t | x in B ==> x in t  $\vee$  B)}
635     (!chain [(x in A | x in B)
636             ==> ((x = h | x in t) | x in B)   [in-def]
637             ==> (x = h | (x in t | x in B)) [prop-taut]
638             ==> (x = h | x in t  $\vee$  B)      [IH]
639             ==> (x in h ++ (t  $\vee$  B))      [in-def]
640             ==> (x in A  $\vee$  B)              [union-def]])
641   }
642
643
644 conclude union-characterization :=
645   (forall A B x . x in A  $\vee$  B <==> x in A | x in B)
646   pick-any A B x
647   (!chain [(x in A  $\vee$  B)
648           <==> (x in A | x in B) [union-characterization-1
649                                   union-characterization-2]])
650
651
652 define UC := union-characterization
653
654 assert* intersection-def :=
655   [(null  $\wedge$  s = null)
656    (h ++ t  $\wedge$  A = h ++ (t  $\wedge$  A) <== h in A)
657    (h ++ t  $\wedge$  A = t  $\wedge$  A <==  $\sim$  h in A)]
658
659 (eval [1 2 1]  $\wedge$  [5 1 3])
660
661 (eval [1 2 1]  $\wedge$  [5])
662
663 conclude intersection-characterization-1 :=
664   (forall A B x . x in A  $\wedge$  B ==> x in A & x in B)
665 by-induction intersection-characterization-1 {
666   null => pick-any B x
667     (!chain [(x in null  $\wedge$  B)
668             ==> (x in null)           [intersection-def]
669             ==> false                 [NC]
670             ==> (x in null & x in B) [prop-taut]])
671 | (A as (insert h t)) =>
672   let {IH := (forall B x . x in t  $\wedge$  B ==> x in t & x in B)}
673     pick-any B x
674     (!two-cases
675       assume (h in B)
676         (!chain [(x in (h ++ t)  $\wedge$  B)
677                 ==> (x in h ++ (t  $\wedge$  B)) [intersection-def]
678                 ==> (x = h | x in t  $\wedge$  B) [in-def]
679                 ==> (x = h | x in t & x in B) [IH]
680                 ==> ((x = h | x in t) & (x = h | x in B)) [prop-taut]
681                 ==> (x in A & (x in B | x in B)) [in-def (h in B)]
682                 ==> (x in A & x in B) [prop-taut]
683               ])
684       assume ( $\sim$  h in B)
685         (!chain [(x in A  $\wedge$  B)
686                 ==> (x in t  $\wedge$  B) [intersection-def]
687                 ==> (x in t & x in B) [IH]
688                 ==> (x in A & x in B) [in-def]]))
689     }
690
691 conclude intersection-characterization-2 :=
692   (forall A B x . x in A & x in B ==> x in A  $\wedge$  B)
693 by-induction intersection-characterization-2 {
694   (A as null) =>
695     pick-any B x
696     (!chain [(x in null & x in B)
697             ==> (x in null)           [left-and]
698             ==> false                 [NC]

```

```

699     ==> (x in null /\ B)      [prop-taut]])
700 | (A as (insert h t)) =>
701   let {IH := (forall B x . x in t & x in B ==> x in t /\ B)}
702     pick-any B x
703     (!two-cases
704       assume (h in B)
705         (!chain [(x in A & x in B)
706           ==> ((x = h | x in t) & x in B)      [in-def]
707           ==> ((x = h & x in B) | (x in t & x in B)) [prop-taut]
708           ==> (x = h | x in t & x in B)      [prop-taut]
709           ==> (x = h | x in t /\ B)          [IH]
710           ==> (x in h ++ (t /\ B))          [in-def]
711           ==> (x in A /\ B)                  [intersection-def]])
712       assume case2 := (~ h in B)
713         (!chain [(x in A & x in B)
714           ==> ((x = h | x in t) & x in B)      [in-def]
715           ==> ((~ x in B | x in t) & x in B)    [case2]
716           ==> ((~ x in B & x in B) | (x in t & x in B)) [prop-taut]
717           ==> (false | x in t & x in B)        [prop-taut]
718           ==> (x in t & x in B)                [prop-taut]
719           ==> (x in t /\ B)                    [IH]
720           ==> (x in A /\ B)                    [intersection-def]))
721 }
722
723
724 conclude intersection-characterization :=
725 (forall A B x . x in A /\ B <==> x in A & x in B)
726 pick-any A B x
727 (!equiv
728   (!chain [(x in A /\ B)
729     ==> (x in A & x in B) [intersection-characterization-1]])
730   (!chain [(x in A & x in B)
731     ==> (x in A /\ B)    [intersection-characterization-2]]))
732
733 define IC := intersection-characterization
734
735 conclude intersection-subset-theorem :=
736 (forall A B . A /\ B subset A)
737 pick-any A B
738 (!subset-intro
739   pick-any x
740     (!chain [(x in A /\ B)
741       ==> (x in A)      [IC]]))
742
743 assert* diff-def :=
744 [(null \ _ = null)
745 (h ++ t \ A = t \ A <== h in A)
746 (h ++ t \ A = h ++ (t \ A) <== ~ h in A)]
747
748 (eval [1 2 3] \ [3 1])
749
750 conclude diff-characterization-1 :=
751 (forall A B x . x in A \ B ==> x in A & ~ x in B)
752 by-induction diff-characterization-1 {
753   (A as null) =>
754     pick-any B x
755       (!chain [(x in A \ B)
756         ==> (x in null)      [diff-def]
757         ==> false           [null-characterization]
758         ==> (x in null & ~ x in B) [prop-taut]])
759 | (A as (insert h t)) =>
760   pick-any B x
761     let {ih := (forall B x . x in t \ B ==> x in t & ~ x in B)}
762       assume hyp := (x in A \ B)
763       (!two-cases
764         assume case1 := (h in B)
765           (!chain-> [hyp
766             ==> (x in t \ B)      [diff-def]
767             ==> (x in t & ~ x in B) [ih]
768             ==> (x in A & ~ x in B) [in-def]])

```

```

769         assume case2 := (~ h in B)
770         (!chain<- [(x = h | x in t \ B)
771                   <== (x in h ++ (t \ B)) [in-def]
772                   <== hyp [diff-def]])
773         assume case2-1 := (x = h)
774         (!chain-> [(h = h)
775                   ==> (h in h ++ t) [in-def]
776                   ==> (h in h ++ t & ~ h in B) [augment]
777                   ==> (x in A & ~ x in B) [case2-1]])
778         assume case2-2 := (x in t \ B)
779         (!chain-> [case2-2
780                   ==> (x in t & ~ x in B) [ih]
781                   ==> (x in h ++ t & ~ x in B) [in-def]]))
782     }
783
784 conclude diff-characterization-2 :=
785   (forall A B x . x in A & ~ x in B ==> x in A \ B)
786 by-induction diff-characterization-2 {
787   (A as null) =>
788     pick-any B x
789     (!chain [(x in A & ~ x in B)
790             ==> (x in A) [left-and]
791             ==> false [null-characterization]
792             ==> (x in A \ B) [prop-taut]])
793   | (A as (h insert t)) =>
794     pick-any B x
795     assume hyp := (x in A & ~ x in B)
796     let {ih := (forall B x . x in t & ~ x in B ==> x in t \ B)}
797     (!cases (!chain-> [(x in A) ==> (x = h | x in t) [in-def]])
798           assume case-1 := (x = h)
799           (!chain<- [(x in A \ B)
800                     <== (x in h ++ (t \ B)) [diff-def case-1]
801                     <== (h in h ++ (t \ B)) [case-1]
802                     <== true [in-lemma-1]])
803           assume case-2 := (x in t)
804           (!two-cases
805             assume (h in B)
806             (!chain<- [(x in A \ B)
807                       <== (x in t \ B) [diff-def]
808                       <== case-2 [ih]])
809             assume (~ h in B)
810             (!chain<- [(x in A \ B)
811                       <== (x in h ++ (t \ B)) [diff-def]
812                       <== (x in t \ B) [in-def]
813                       <== case-2 [ih]]))
814           )
815
816 conclude diff-characterization :=
817   (forall A B x . x in A \ B <==> x in A & ~ x in B)
818 pick-any A B x
819   (!equiv
820     (!chain [(x in A \ B)
821             ==> (x in A & ~ x in B) [diff-characterization-1]])
822     (!chain [(x in A & ~ x in B)
823             ==> (x in A \ B) [diff-characterization-2]]))
824
825 define DC := diff-characterization
826
827
828 conclude intersection-commutes := (forall A B . A /\ B = B /\ A)
829 pick-any A B
830   (!set-identity-intro-direct
831     pick-any x
832     (!chain [(x in A /\ B) <==> (x in A & x in B) [IC]
833             <==> (x in B & x in A) [prop-taut]
834             <==> (x in B /\ A) [IC]]))
835
836
837
838 conclude intersection-commutes := (forall A B . A /\ B = B /\ A)

```

```

839 pick-any A B
840   let {A-subset-of-B :=
841     (!subset-intro
842       pick-any x
843         (!chain [(x in A /\ B)
844                 ==> (x in A & x in B) [IC]
845                 ==> (x in B & x in A) [prop-taut]
846                 ==> (x in B /\ A) [IC]]));
847     B-subset-of-A :=
848     (!subset-intro
849       pick-any x
850         (!chain [(x in B /\ A)
851                 ==> (x in B & x in A) [IC]
852                 ==> (x in A & x in B) [prop-taut]
853                 ==> (x in A /\ B) [IC]]))
854   }
855   (!set-identity-intro A-subset-of-B B-subset-of-A)
856
857 conclude intersection-commutes := (forall A B . A /\ B = B /\ A)
858 let {M := method (A B) # derive (A /\ B subset B /\ A)
859     (!subset-intro
860       pick-any x
861         (!chain [(x in A /\ B)
862                 ==> (x in A & x in B) [IC]
863                 ==> (x in B & x in A) [prop-taut]
864                 ==> (x in B /\ A) [IC]]))
865   pick-any A B
866   (!set-identity-intro (!M A B) (!M B A))
867
868 conclude intersection-subset-theorem-2 :=
869   (forall A B . A /\ B subset B)
870 pick-any A B
871   (!chain-> [true ==> (B /\ A subset B) [intersection-subset-theorem]
872             ==> (A /\ B subset B) [intersection-commutes]])
873
874 conclude intersection-subset-theorem' :=
875   (forall A B C . A subset B /\ C <==> A subset B & A subset C)
876 pick-any A B C
877   (!equiv assume (A subset B /\ C)
878     (!both (!subset-intro
879       pick-any x
880         (!chain [(x in A) ==> (x in B /\ C) [SC]
881                 ==> (x in B) [IC]]))
882     (!subset-intro
883       pick-any x
884         (!chain [(x in A) ==> (x in B /\ C) [SC]
885                 ==> (x in C) [IC]])))
886   assume (A subset B & A subset C)
887   (!subset-intro
888     pick-any x
889     assume (x in A)
890     let {_ := (!chain-> [(x in A) ==> (x in B) [SC]]);
891         _ := (!chain-> [(x in A) ==> (x in C) [SC]]);
892         p := (!both (x in B) (x in C))}
893     (!chain-> [p ==> (x in B /\ C) [IC]]))
894
895 conclude union-subset-theorem :=
896   (forall A B C . A subset B | A subset C ==> A subset B \/ C)
897 pick-any A B C
898 assume hyp := (A subset B | A subset C)
899   (!cases hyp
900     assume (A subset B)
901     (!subset-intro
902       pick-any x
903         (!chain [(x in A) ==> (x in B) [SC]
904                 ==> (x in B | x in C) [alternate]
905                 ==> (x in B \/ C) [UC]]))
906     assume (A subset C)
907     (!subset-intro
908       pick-any x

```

```

909             (!chain [(x in A) ==> (x in C)           [SC]
910                    ==> (x in B | x in C) [alternate]
911                    ==> (x in B \ / C)   [UC]])))
912
913 conclude union-commutes := (forall A B . A \ / B = B \ / A)
914 pick-any A B
915   (!set-identity-intro-direct
916     pick-any x
917       (!chain [(x in A \ / B) <==> (x in A | x in B) [UC]
918              <==> (x in B | x in A) [prop-taut]
919              <==> (x in B \ / A)   [UC]]))
920
921 conclude intersection-associativity :=
922   (forall A B C . A /\ (B /\ C) = (A /\ B) /\ C)
923 pick-any A B C
924   (!set-identity-intro-direct
925     pick-any x
926       (!chain [(x in A /\ B /\ C)
927              <==> (x in A & x in B /\ C)           [IC]
928              <==> (x in A & x in B & x in C)       [IC]
929              <==> ((x in A & x in B) & x in C) [prop-taut]
930              <==> ((x in A /\ B) & x in C)         [IC]
931              <==> (x in (A /\ B) /\ C)             [IC]]))
932
933 conclude union-associativity :=
934   (forall A B C . A \ / B \ / C = (A \ / B) \ / C)
935 pick-any A B C
936   (!set-identity-intro-direct
937     pick-any x
938       (!chain [(x in A \ / B \ / C)
939              <==> (x in A | x in B \ / C)           [UC]
940              <==> (x in A | x in B | x in C)       [UC]
941              <==> ((x in A | x in B) | x in C) [prop-taut]
942              <==> (x in A \ / B | x in C)         [UC]
943              <==> (x in (A \ / B) \ / C)           [UC]]))
944
945 conclude /\-idempotence :=
946   (forall A . A /\ A = A)
947 pick-any A
948   (!set-identity-intro-direct
949     pick-any x
950       (!chain [(x in A /\ A)
951              <==> (x in A & x in A) [IC]
952              <==> (x in A)         [prop-taut]]))
953
954 conclude \/-idempotence :=
955   (forall A . A \ / A = A)
956 pick-any A
957   (!set-identity-intro-direct
958     pick-any x
959       (!chain [(x in A \ / A)
960              <==> (x in A | x in A) [UC]
961              <==> (x in A)         [prop-taut]]))
962
963 conclude union-null-theorem :=
964   (forall A B . A \ / B = null <==> A = null & B = null)
965 pick-any A B
966   (!chain [(A \ / B = null)
967          <==> (forall x . x in A \ / B <==> x in null) [SIC]
968          <==> (forall x . x in A \ / B <==> false)     [NC]
969          <==> (forall x . x in A | x in B <==> false) [UC]
970          <==> (forall x . ~ x in A & ~ x in B)         [prop-taut]
971          <==> ((forall x . ~ x in A) & (forall x ~ x in B)) [taut]
972          <==> (A = null & B = null)                   [NC-2]])
973
974
975 conclude distributivity-1 :=
976   (forall A B C . A \ / (B /\ C) = (A \ / B) /\ (A \ / C))
977 pick-any A B C
978   (!set-identity-intro-direct

```

```

979     pick-any x
980     (!chain [(x in A \ (B /\ C))
981             <==> (x in A | x in B /\ C)                [UC]
982             <==> (x in A | x in B & x in C)           [IC]
983             <==> ((x in A | x in B) & (x in A | x in C)) [prop-taut]
984             <==> (x in A \ B & x in A \ C)           [UC]
985             <==> (x in (A \ B) /\ (A \ C))           [IC]]))
986
987
988 conclude distributivity-2 :=
989   (forall A B C . A /\ (B \ C) = (A /\ B) \ (A /\ C))
990   pick-any A B C
991   (!set-identity-intro-direct
992     pick-any x
993     (!chain [(x in A /\ (B \ C))
994             <==> (x in A & x in B \ C)                [IC]
995             <==> (x in A & (x in B | x in C))         [UC]
996             <==> ((x in A & x in B) | (x in A & x in C)) [prop-taut]
997             <==> (x in A /\ B | x in A /\ C)         [IC]
998             <==> (x in (A /\ B) \ (A /\ C))         [UC]]))
999
1000 conclude diff-theorem-1 := (forall A . A \ A = null)
1001 pick-any A
1002   (!set-identity-intro-direct
1003     pick-any x
1004     (!chain [(x in A \ A)
1005             <==> (x in A & ~ x in A) [DC]
1006             <==> false [prop-taut]
1007             <==> (x in null) [NC]]))
1008
1009 conclude diff-theorem-2 :=
1010   (forall A B C . B subset C ==> A \ C subset A \ B)
1011   pick-any A B C
1012   assume (B subset C)
1013   (!subset-intro
1014     pick-any x
1015     (!chain [(x in A \ C)
1016             ==> (x in A & ~ x in C) [DC]
1017             ==> (x in A & ~ x in B) [SC]
1018             ==> (x in A \ B) [DC]]))
1019
1020
1021 define p := (forall A B C . B subset C ==> A \ B subset A \ C)
1022
1023 (falsify p 20)
1024
1025 conclude diff-theorem-3 :=
1026   (forall A B . A \ (A /\ B) = A \ B)
1027   pick-any A B
1028   (!set-identity-intro-direct
1029     pick-any x
1030     (!chain [(x in A \ (A /\ B))
1031             <==> (x in A & ~ x in A /\ B)                [DC]
1032             <==> (x in A & ~ (x in A & x in B))         [IC]
1033             <==> (x in A & (~ x in A | ~ x in B))         [prop-taut]
1034             <==> ((x in A & ~ x in A) | (x in A & ~ x in B)) [prop-taut]
1035             <==> (false | x in A & ~ x in B)             [prop-taut]
1036             <==> (x in A & ~ x in B)                     [prop-taut]
1037             <==> (x in A \ B)                             [DC]]))
1038
1039 conclude diff-theorem-4 :=
1040   (forall A B . A /\ (A \ B) = A \ B)
1041   pick-any A B
1042   (!set-identity-intro-direct
1043     pick-any x
1044     (!chain [(x in A /\ (A \ B))
1045             <==> (x in A & x in A \ B)                [IC]
1046             <==> (x in A & x in A & ~ x in B)         [DC]
1047             <==> (x in A & ~ x in B)                   [prop-taut]
1048             <==> (x in A \ B)                           [DC]]))

```

```

1049
1050 conclude diff-theorem-5 :=
1051   (forall A B . (A \ B) \ / B = A \ / B)
1052   pick-any A B
1053     (!set-identity-intro-direct
1054       pick-any x
1055         (!chain [(x in (A \ B) \ / B)
1056                 <==> (x in A \ B | x in B) [UC]
1057                 <==> ((x in A & ~ x in B) | x in B) [DC]
1058                 <==> ((x in A | x in B) & (~ x in B | x in B)) [prop-taut]
1059                 <==> ((x in A | x in B) & true) [prop-taut]
1060                 <==> (x in A | x in B) [prop-taut]
1061                 <==> (x in A \ / B) [UC]]))
1062
1063 conclude diff-theorem-6 :=
1064   (forall A B . (A \ / B) \ B = A \ B)
1065   pick-any A B
1066     (!set-identity-intro-direct
1067       pick-any x
1068         (!chain [(x in (A \ / B) \ B)
1069                 <==> (x in A \ / B & ~ x in B) [DC]
1070                 <==> ((x in A | x in B) & ~ x in B) [UC]
1071                 <==> (x in A & ~ x in B | x in B & ~ x in B) [prop-taut]
1072                 <==> (x in A & ~ x in B | false) [prop-taut]
1073                 <==> (x in A \ B | false) [DC]
1074                 <==> (x in A \ B) [prop-taut]]))
1075
1076 conclude diff-theorem-7 :=
1077   (forall A B . (A / \ B) \ B = null)
1078   pick-any A B
1079     (!set-identity-intro-direct
1080       pick-any x
1081         (!chain [(x in (A / \ B) \ B)
1082                 <==> (x in A / \ B & ~ x in B) [DC]
1083                 <==> ((x in A & x in B) & ~ x in B) [IC]
1084                 <==> false [prop-taut]
1085                 <==> (x in null) [NC]]))
1086
1087 conclude diff-theorem-8 :=
1088   (forall A B . (A \ B) / \ B = null)
1089   pick-any A B
1090     (!set-identity-intro-direct
1091       pick-any x
1092         (!chain [(x in (A \ B) / \ B)
1093                 <==> (x in A \ B & x in B) [IC]
1094                 <==> ((x in A & ~ x in B) & x in B) [DC]
1095                 <==> false [prop-taut]
1096                 <==> (x in null) [NC]]))
1097
1098 conclude diff-theorem-8 :=
1099   (forall A B C . A \ (B \ / C) = (A \ B) / \ (A \ C))
1100   pick-any A B C
1101     (!set-identity-intro-direct
1102       pick-any x
1103         (!chain [(x in A \ (B \ / C))
1104                 <==> (x in A & ~ x in B \ / C) [DC]
1105                 <==> (x in A & ~ (x in B | x in C)) [UC]
1106                 <==> (x in A & ~ x in B & ~ x in C) [prop-taut]
1107                 <==> ((x in A & ~ x in B) & (x in A & ~ x in C)) [prop-taut]
1108                 <==> (x in A \ B & x in A \ C) [DC]
1109                 <==> (x in (A \ B) / \ (A \ C)) [IC]]))
1110
1111 conclude diff-theorem-9 :=
1112   (forall A B C . A \ (B / \ C) = (A \ B) \ / (A \ C))
1113   pick-any A B C
1114     (!set-identity-intro-direct
1115       pick-any x
1116         (!chain [(x in A \ (B / \ C))
1117                 <==> (x in A & ~ x in B / \ C) [DC]
1118                 <==> (x in A & ~ (x in B & x in C)) [IC]

```



```

1119         <==> (x in A & (~ x in B | ~ x in C)) [prop-taut]
1120         <==> ((x in A & ~ x in B) | (x in A & ~ x in C)) [prop-taut]
1121         <==> (x in A \ B | x in A \ C) [DC]
1122         <==> (x in (A \ B) \/ (A \ C)) [UC]])
1123
1124 conclude diff-theorem-10 := (forall A B . A \ (A \ B) = A /\ B)
1125 pick-any A B
1126   (!set-identity-intro-direct
1127     pick-any x
1128     (!chain [(x in A \ (A \ B))
1129               <==> (x in A & ~ x in A \ B) [DC]
1130               <==> (x in A & ~ (x in A & ~ x in B)) [DC]
1131               <==> (x in A & (~ x in A | ~ ~ x in B)) [prop-taut]
1132               <==> ((x in A & ~ x in A) | (x in A & x in B)) [prop-taut]
1133               <==> (false | x in A & x in B) [prop-taut]
1134               <==> (x in A & x in B) [prop-taut]
1135               <==> (x in A /\ B) [IC]]))
1136
1137 conclude diff-theorem-11 := (forall A B . A subset B ==> A \ (B \ A) = B)
1138 pick-any A B
1139 assume hyp := (A subset B)
1140   (!set-identity-intro-direct
1141     pick-any x
1142     (!chain
1143       [(x in A \ (B \ A))
1144         <==> (x in A | x in B \ A) [UC]
1145         <==> (x in A | x in B & ~ x in A) [DC]
1146         <==> ((x in A | x in B) & (x in A | ~ x in A)) [prop-taut]
1147         <==> (x in A | x in B) [prop-taut]
1148         <==> (x in B | x in B) [SC prop-taut]
1149         <==> (x in B) [prop-taut]]))
1150
1151
1152 conclude diff-theorem-12 :=
1153   (forall A B . A = (A \ B) \/ (A /\ B))
1154 pick-any A B
1155   (!comm
1156     (!set-identity-intro-direct
1157       pick-any x
1158       (!chain [(x in (A \ B) \/ (A /\ B))
1159                 <==> (x in A \ B | x in A /\ B) [UC]
1160                 <==> (x in A & ~ x in B | x in A & x in B) [DC IC]
1161                 <==> (x in A) [prop-taut]]))
1162
1163 conclude diff-theorem-13 :=
1164   (forall A B . (A \ B) /\ (A /\ B) = null)
1165 pick-any A B
1166   (!set-identity-intro-direct
1167     pick-any x
1168     (!chain [(x in (A \ B) /\ (A /\ B))
1169               <==> (x in (A \ B) & x in A /\ B) [IC]
1170               <==> ((x in A & ~ x in B) & (x in A & x in B)) [DC IC]
1171               <==> false [prop-taut]
1172               <==> (x in null) [NC]]))
1173
1174
1175 #define diff-remove-theorem := (forall A x . A - x = A \ singleton x)
1176 #(mark 'A)
1177 # START_LOAD
1178 # datatype-cases diff-remove-theorem {
1179 # null => pick-any x
1180 # (!set-identity-intro-direct
1181 # pick-any y
1182 # (!chain [(y in null - x)
1183             <==> (y in null)
1184             <==> false
1185             <==> (y in null & ~ y in singleton x)
1186             <==> (y in null \ singleton x)])
1187 # | (A as (insert h t)) =>
1188 # pick-any x

```

```

1189 #           (!set-identity-intro
1190 #             (!subset-intro
1191 #               pick-any y
1192 #                 assume hyp := (y in A - x)
1193 #                   (!two-cases
1194 #                     assume case-1 := (x = h)
1195 #                       let {y/=x := (!chain [(y in A - x)
1196 #                                             ==> (y in t - x)
1197 #                                             ==> (y in t \ singleton x)
1198 #                                             ==> (y in t & ~ y in singleton x)
1199 #                                             ==> (y /= x)]])
1200 #                 }
1201 #END_LOAD
1202
1203 #(!induction* diff-remove-theorem)
1204
1205 conclude absorption-1 :=
1206   (forall x A . x in A <==> x ++ A = A)
1207   pick-any x A
1208     (!equiv
1209       assume hyp := (x in A)
1210       (!set-identity-intro-direct
1211         pick-any y
1212           (!chain [(y in x ++ A)
1213                   <==> (y = x | y in A)           [in-def]
1214                   <==> (y in A | y in A)         [hyp prop-taut]
1215                   <==> (y in A)                 [prop-taut]]))
1216         assume (x ++ A = A)
1217         (!chain-> [true ==> (x in x ++ A) [in-lemma-1]
1218                  ==> (x in A)          [set-identity-characterization]]))
1219
1220 conclude subset-theorem-1 :=
1221   (forall A B . A subset B ==> A \ / B = B)
1222   pick-any A B
1223     assume (A subset B)
1224     (!set-identity-intro-direct
1225       pick-any x
1226         (!chain [(x in A \ / B)
1227                 <==> (x in A | x in B) [UC]
1228                 <==> (x in B | x in B) [prop-taut SC]
1229                 <==> (x in B)         [prop-taut]]))
1230
1231
1232 conclude subset-theorem-2 :=
1233   (forall A B . A subset B ==> A /\ B = A)
1234   pick-any A B
1235     assume (A subset B)
1236     (!set-identity-intro-direct
1237       pick-any x
1238         (!chain [(x in A /\ B)
1239                 <==> (x in A & x in B) [IC]
1240                 <==> (x in A & x in A) [prop-taut SC]
1241                 <==> (x in A)         [prop-taut]]))
1242
1243
1244 conclude intersection-lemma-1 :=
1245   (forall A B x . x in B & x in A ==> A /\ B = (x ++ A) /\ B)
1246   pick-any A B x
1247     assume hyp := (x in B & x in A)
1248     (!set-identity-intro-direct
1249       pick-any y
1250         (!chain [(y in A /\ B)
1251                 <==> (y in A & y in B)           [IC]
1252                 <==> ((y = x | y in A) & y in B) [(y in A <==> y = x | y in A) <== (x in A) [in-lemma-4]]
1253                 <==> ((y in x ++ A) & y in B)   [in-def]
1254                 <==> (y in (x ++ A) /\ B)       [IC]]))
1255
1256 conclude intersection-lemma-2 :=
1257   (forall A B x . ~ x in A ==> ~ x in A /\ B)
1258   pick-any A B x

```

```

1259 assume hyp := (~ x in A)
1260   (!by-contradiction (~ x in A /\ B)
1261     (!chain [(x in A /\ B)
1262       ==> (x in A) [IC]
1263       ==> (x in A & ~ x in A) [augment]
1264       ==> false [prop-taut]]))
1265
1266
1267 conclude intersection-lemma-3 :=
1268   (forall A . A /\ A = A)
1269 pick-any A
1270   (!set-identity-intro-direct
1271     pick-any x
1272     (!chain [(x in A /\ A)
1273       <==> (x in A & x in A) [IC]
1274       <==> (x in A) [prop-taut]]))
1275
1276 declare insert-in-all: (S) [S (Set (Set S))] -> (Set (Set S)) [[id lst->set]]
1277
1278 assert* insert-in-all-def :=
1279   [(x insert-in-all null = null)
1280   (x insert-in-all A ++ t = (x ++ A) ++ (x insert-in-all t))]
1281
1282 define in-all := insert-in-all
1283
1284 conclude insert-in-all-characterization :=
1285   (forall U s x . s in x in-all U <==> exists B . B in U & s = x ++ B)
1286 by-induction insert-in-all-characterization {
1287   (U as null) => pick-any s x
1288     (!equiv (!chain [(s in x in-all U)
1289       ==> (s in null) [insert-in-all-def]
1290       ==> false [NC]
1291       ==> (exists B . B in U & s = x ++ B) [prop-taut]]))
1292     assume hyp := (exists B . B in U & s = x ++ B)
1293     pick-witness B for hyp
1294     (!chain-> [(B in U)
1295       ==> false [NC]
1296       ==> (s in x in-all U) [prop-taut]]))
1297 | (U as (insert A more)) =>
1298   let {IH := (forall s x . s in x in-all more <==> exists B . B in more & s = x ++ B)}
1299   pick-any s x
1300   let {
1301     G := (exists B . B in U & s = x ++ B);
1302     L := conclude ((s = x ++ A | exists B . B in more & s = x ++ B) <==> G)
1303     (!equiv
1304       assume hyp := (s = x ++ A | exists B . B in more & s = x ++ B)
1305       (!cases hyp
1306         assume (s = x ++ A)
1307           (!chain-> [true ==> (A in U) [in-lemma-1]
1308             ==> (s = x ++ A & A in U) [augment]
1309             ==> (A in U & s = x ++ A) [comm]
1310             ==> G [existence]]))
1311           (!chain [(exists B . B in more & s = x ++ B)
1312             ==> (exists B . B in U & s = x ++ B) [in-def]]))
1313         assume hyp := (exists B . B in U & s = x ++ B)
1314         let {goal := (s = x ++ A | exists B . B in more & s = x ++ B)}
1315         pick-witness B for hyp
1316         (!cases (!chain-> [(B = A | B in more) <== (B in U) [in-def]]))
1317         assume (B = A)
1318         (!chain-> [(s = x ++ B) ==> (s = x ++ A) [(B = A)]
1319           ==> goal [alternate]]))
1320         assume (B in more)
1321         (!chain-> [(B in more)
1322           ==> (B in more & s = x ++ B) [augment]
1323           ==> (exists B . B in more & s = x ++ B) [existence]
1324           ==> goal [alternate]]))
1325       })
1326   (!chain [(s in x in-all U)
1327     <==> (s in (x ++ A) ++ (x in-all more)) [insert-in-all-def]
1328     <==> (s = x ++ A | s in x in-all more) [in-def]

```

```

1329         <==> (s = x ++ A | exists B . B in more & s = x ++ B) [IH]
1330         <==> G [L]
1331     ])
1332 }
1333
1334 declare powerset: (S) [(Set S)] -> (Set (Set S)) [[lst->set]]
1335
1336 assert* powerset-def :=
1337   [(powerset null = singleton null)
1338     (powerset x ++ t = (powerset t) \ / (x insert-in-all (powerset t)))]
1339
1340 conclude powerset-characterization :=
1341   (forall A B . B in powerset A <==> B subset A)
1342 by-induction powerset-characterization {
1343   (A as Set.null) =>
1344     pick-any B
1345       (!chain [(B in powerset A)
1346               <==> (B in singleton null) [powerset-def]
1347               <==> (B = null) [singleton-characterization]
1348               <==> (B subset null) [subset-lemma-6]])
1349 | (A as (Set.insert h t:(Set.Set 'S))) =>
1350   let {IH := (forall B . B in powerset t <==> B subset t)}
1351   pick-any B:(Set.Set 'S)
1352     let {e1 := (!chain [(B in powerset A)
1353                       <==> (B in (powerset t) \ / (h in-all powerset t)) [powerset-def]
1354                       <==> (B in powerset t | B in h in-all powerset t) [UC]
1355                       <==> (B subset t | B in h in-all powerset t) [IH]
1356                       <==> (B subset t | exists s . s in powerset t & B = h ++ s) [insert-in-all-characterization]
1357                       <==> (B subset t | exists s . s subset t & B = h ++ s) [IH]]];
1358     lemma := (!chain-> [true ==> (h in h ++ t) [in-lemma-1]]);
1359     p3 := (assume hyp := (B subset t | exists s . s subset t & B = h ++ s)
1360           (!cases hyp
1361             (!chain [(B subset t) ==> (B subset A) [subset-lemma-5]]
1362               (assume ehyp := (exists s . s subset t & B = h ++ s)
1363                 pick-witness s for ehyp
1364                   (!subset-intro
1365                     pick-any x
1366                       assume (x in B)
1367                         (!chain-> [(x in B) ==> (x in h ++ s) [(B = h ++ s)]
1368                                   ==> (x = h | x in s) [in-def]
1369                                   ==> (x in h ++ t | x in s) [lemma]
1370                                   ==> (x in A | x in t) [SC]
1371                                   ==> (x in A | x in A) [in-def]
1372                                   ==> (x in A) [prop-taut]])))));
1373     p4 := (assume (B subset A)
1374           (!two-cases
1375             assume case1 := (h in B)
1376               (!chain-> [(B subset A)
1377                       ==> (B subset A & h in B) [augment]
1378                       ==> (exists s . s subset t & B = h ++ s) [subset-lemma-3]
1379                       ==> (B subset t | exists s . s subset t & B = h ++ s) [alternate]])
1380             assume case2 := (~ h in B)
1381               (!chain-> [case2 ==> (~ h in B & B subset A) [augment]
1382                       ==> (B subset t) [subset-lemma-4]
1383                       ==> (B subset t | exists s . s subset t & B = h ++ s) [alternate]]));
1384     p3<=>p4 := (!equiv p3 p4)}
1385   (!equiv-tran e1 p3<=>p4)
1386 }
1387
1388 define POSC := powerset-characterization
1389
1390 conclude ps-theorem-1 := (forall A . null in powerset A)
1391 pick-any A
1392   (!chain-> [true ==> (null subset A) [subset-def]
1393             ==> (null in powerset A) [POSC]])
1394
1395 conclude ps-theorem-2 := (forall A . A in powerset A)
1396 pick-any A
1397   (!chain-> [true ==> (A subset A) [subset-reflexivity]
1398             ==> (A in powerset A) [POSC]])
1399

```

```

1399
1400 conclude ps-theorem-3 :=
1401   (forall A B . A subset B <==> powerset A subset powerset B)
1402 pick-any A B
1403   (!equiv assume (A subset B)
1404     (!subset-intro
1405       pick-any C
1406       (!chain [(C in powerset A)
1407         ==> (C subset A)           [POSC]
1408         ==> (C subset B)           [subset-transitivity]
1409         ==> (C in powerset B)      [POSC]]))
1410     assume (powerset A subset powerset B)
1411     (!chain-> [true ==> (A in powerset A) [ps-theorem-2]
1412       ==> (A in powerset B) [SC]
1413       ==> (A subset B) [POSC]]))
1414
1415 conclude ps-theorem-4 :=
1416   (forall A B . powerset A /\ B = (powerset A) /\ (powerset B))
1417 pick-any A B
1418   (!set-identity-intro-direct
1419     pick-any C
1420     (!chain
1421       [(C in powerset A /\ B)
1422       <==> (C subset A /\ B)           [POSC]
1423       <==> (C subset A & C subset B)   [intersection-subset-theorem']
1424       <==> (C in powerset A & C in powerset B) [POSC]
1425       <==> (C in (powerset A) /\ (powerset B)) [IC]]))
1426
1427 conclude ps-theorem-5 :=
1428   (forall A B . (powerset A) \/ (powerset B) subset powerset A \/ B)
1429 pick-any A B
1430   (!subset-intro
1431     pick-any C
1432     (!chain [(C in (powerset A) \/ (powerset B))
1433       ==> (C in powerset A | C in powerset B) [UC]
1434       ==> (C subset A | C subset B) [POSC]
1435       ==> (C subset A \/ B) [union-subset-theorem]
1436       ==> (C in powerset A \/ B) [POSC]]))
1437
1438 declare paired-with: (S, T) [S (Set T)] -> (Set (Pair S T))
1439                                     [130 [id lst->set]]
1440
1441 assert* paired-with-def :=
1442   [(_ paired-with null = null)
1443   (x paired-with h ++ t = x @ h ++ (x paired-with t))]
1444
1445 (eval 3 paired-with [2 8])
1446
1447 conclude paired-with-characterization :=
1448   (forall B x y a . x @ y in a paired-with B <==> x = a & y in B)
1449 by-induction paired-with-characterization {
1450   null => pick-any x y a
1451     (!chain [(x @ y in a paired-with null)
1452       <==> (x @ y in null) [paired-with-def]
1453       <==> false [null-characterization]
1454       <==> (x = a & false) [prop-taut]
1455       <==> (x = a & y in null) [null-characterization]])
1456 | (B as (insert h t)) =>
1457   pick-any x y a
1458     let {IH := (forall x y a . x @ y in a paired-with t <==> x = a & y in t)}
1459     (!chain
1460       [(x @ y in a paired-with h ++ t)
1461       <==> (x @ y in a @ h ++ (a paired-with t)) [paired-with-def]
1462       <==> (x @ y = a @ h | x @ y in a paired-with t) [in-def]
1463       <==> (x = a & y = h | x @ y in a paired-with t) [pair-axioms]
1464       <==> (x = a & y = h | x = a & y in t) [IH]
1465       <==> (x = a & (y = h | y in t)) [prop-taut]
1466       <==> (x = a & y in B) [in-def]])
1467   }
1468

```

```

1469 conclude paired-with-lemma-1 :=
1470   (forall A x . x paired-with A = null ==> A = null)
1471 datatype-cases paired-with-lemma-1 {
1472   null => pick-any x
1473     (!chain [(x paired-with null = null)
1474              ==> (null = null)] [paired-with-def])
1475 | (insert h t) =>
1476   pick-any x
1477     (!chain
1478       [(x paired-with h ++ t = null)
1479        ==> (x @ h ++ (x paired-with t) = null) [paired-with-def]
1480         ==> (forall z . ~ z in x @ h ++ (x paired-with t)) [NC-2]
1481         ==> (forall z . ~ (z = x @ h | z in x paired-with t)) [in-def]
1482
1483         ==> (forall z . z /= x @ h) [prop-taut]
1484         ==> (x @ h /= x @ h) [(uspec with x @ h)]
1485         ==> (x @ h /= x @ h & x @ h = x @ h) [augment]
1486         ==> false [prop-taut]
1487         ==> (h ++ t = null) [prop-taut]])
1488 }
1489
1490 declare product: (S, T) [(Set S) (Set T)] -> (Set (Pair S T)) [150 [lst->set lst->set]]
1491
1492 define X := product
1493
1494 assert* product-def :=
1495   [(null X _ = null)
1496    (h ++ t X A = h paired-with A \ / t X A)]
1497
1498 (eval [1 2] X ['foo 'bar 'car])
1499
1500
1501
1502 conclude cartesian-product-characterization :=
1503   (forall A B a b . a @ b in A X B <==> a in A & b in B)
1504 by-induction cartesian-product-characterization {
1505   null => pick-any B a b
1506     (!chain [(a @ b in null X B)
1507              <==> (a @ b in null) [product-def]
1508              <==> false [null-characterization]
1509              <==> (a in null & b in B) [prop-taut null-characterization]])
1510 | (A as (insert h t)) =>
1511   let {IH := (forall B a b . a @ b in t X B <==> a in t & b in B)}
1512     pick-any B a b
1513       (!chain [(a @ b in h ++ t X B)
1514                <==> (a @ b in h paired-with B \ / t X B) [product-def]
1515                <==> (a @ b in h paired-with B | a @ b in t X B) [UC]
1516                <==> (a = h & b in B | a in t & b in B) [paired-with-characterization IH]
1517                <==> ((a = h | a in t) & b in B) [prop-taut]
1518                <==> (a in A & b in B) [in-def]])
1519 }
1520
1521 define CPC := cartesian-product-characterization
1522
1523 conclude cartesian-product-characterization-2 :=
1524   (forall x A B . x in A X B <==> exists a b . x = a @ b & a in A & b in B)
1525 pick-any x A B
1526   (!equiv
1527     assume hyp := (x in A X B)
1528     let {p := (!chain-> [true ==> (exists a b . x = a @ b) [pair-axioms]])}
1529     pick-witnesses a b for p x=a@b
1530       (!chain-> [x=a@b ==> (a @ b in A X B) [hyp]
1531                ==> (a in A & b in B) [CPC]
1532                ==> (x=a@b & a in A & b in B) [augment]
1533                ==> (exists a b . x = a @ b & a in A & b in B) [existence]])
1534     assume hyp := (exists a b . x = a @ b & a in A & b in B)
1535     pick-witnesses a b for hyp spec-premise
1536       (!chain-> [spec-premise
1537                ==> (a in A & b in B) [prop-taut]
1538                ==> (a @ b in A X B) [CPC]

```

```

1539         ==> (x in A X B)          [(x = a @ b)]))
1540
1541 define CPC-2 := cartesian-product-characterization-2
1542
1543 define taut := (method (p q) (!sprove-from q [p]))
1544
1545 conclude product-theorem-1 :=
1546   (forall A B . A X B = null ==> A = null | B = null)
1547 datatype-cases product-theorem-1 {
1548   null => pick-any B
1549     (!chain [(null X B = null)
1550             ==> (null = null)          [product-def]
1551             ==> (null = null | B = null) [alternate]])
1552 | (A as (insert h t)) =>
1553   pick-any B
1554     (!chain [(h ++ t X B = null)
1555             ==> (h paired-with B \ / t X B = null)      [product-def]
1556             ==> (h paired-with B = null & t X B = null) [union-null-theorem]
1557             ==> (B = null)                               [paired-with-lemma-1]
1558             ==> (h ++ t = null | B = null)              [alternate]])
1559 }
1560
1561 conclude product-theorem-2 :=
1562   (forall A B . A X B = null <==> A = null | B = null)
1563   pick-any A:(Set 'T1) B:(Set 'T2)
1564     (!chain [(A X B = null)
1565             <==> (forall x . ~ x in A X B)                [NC-2]
1566             <==> (forall x . ~ exists a b . x = a @ b & a in A & b in B) [CPC-2]
1567             <==> (forall x a b . a in A & b in B ==> x != a @ b) [taut]
1568             <==> (forall a b . a in A & b in B ==> forall x . x != a @ b) [taut]
1569             <==> (forall a b . a in A & b in B ==> false) [taut]
1570             <==> (forall a b . ~ a in A | ~ b in B) [taut]
1571             <==> ((forall a . ~ a in A) | (forall b . ~ b in B)) [taut]
1572             <==> (A = null | B = null)                  [NC-2]])
1573
1574 conclude product-theorem-3 :=
1575   (forall A B . non-empty A & non-empty B ==> A X B = B X A <==> A = B)
1576   pick-any A:(Set 'S) B:(Set 'T)
1577     assume hyp := (non-empty A & non-empty B)
1578     let {p1 := (!chain-> [(non-empty A) ==> (exists a . a in A) [NC-3]]);
1579         p2 := (!chain-> [(non-empty B) ==> (exists b . b in B) [NC-3]]);
1580         M := method (S1 S2 c2) # assumes c2 in S2, S1 X S2 = S2 X S1,
1581             (!subset-intro # and derives (S1 subset S2))
1582         pick-any x
1583           (!chain [(x in S1)
1584                 ==> (x in S1 & c2 in S2) [augment]
1585                 ==> (x @ c2 in S1 X S2) [CPC]
1586                 ==> (x @ c2 in S2 X S1) [SIC]
1587                 ==> (x in S2 & c2 in S1) [CPC]
1588                 ==> (x in S2) [left-and]])
1589     }
1590     pick-witness a for p1 # (a in A)
1591     pick-witness b for p2 # (b in B)
1592     (!equiv
1593       assume hyp := (A X B = B X A)
1594       (!set-identity-intro (!M A B b) (!M B A a))
1595       assume hyp := (A = B)
1596       (!chain-> [(A X A = A X A) ==> (A X B = B X A) [hyp]]))
1597
1598 conclude product-theorem-4 :=
1599   (forall A B C . non-empty A & A X B subset A X C ==> B subset C)
1600   pick-any A B C
1601     assume hyp := (non-empty A & A X B subset A X C)
1602     pick-witness a for (!chain-> [hyp ==> (exists a . a in A) [NC-3]])
1603     (!subset-intro
1604       pick-any b
1605         (!chain [(b in B)
1606                 ==> (a in A & b in B) [augment]
1607                 ==> (a @ b in A X B) [CPC]
1608                 ==> (a @ b in A X C) [SC]

```

```

1609         ==> (a in A & b in C) [CPC]
1610         ==> (b in C)           [right-and]]))
1611
1612 define pair-converter :=
1613   method (premise)
1614     match premise {
1615       (forall u:'S (forall v:'T body)) =>
1616         pick-any p:(Pair 'S 'T)
1617         let {E := (!chain-> [true ==> (exists ?x:'S ?y:'T .
1618                                     p = ?x @ ?y) [pair-axioms]])}
1619           pick-witnesses x y for E
1620           let {body' := (!uspec* premise [x y])}
1621             (!chain-> [body'
1622                       ==> (replace-term-in-sentence (x @ y) body' p)
1623                       [(p = x @ y)]])
1624         }
1625
1626 conclude product-theorem-5 :=
1627   (forall A B C . B subset C ==> A X B subset A X C)
1628 pick-any A B C
1629   assume (B subset C)
1630   (!subset-intro
1631     (!pair-converter
1632       pick-any a b
1633       (!chain [(a @ b in A X B)
1634               ==> (a in A & b in B)   [CPC]
1635               ==> (a in A & b in C)   [SC]
1636               ==> (a @ b in A X C)   [CPC]])))
1637
1638 conclude product-theorem-6 :=
1639   (forall A B C . A X (B /\ C) = A X B /\ A X C)
1640 pick-any A B C
1641   (!set-identity-intro-direct
1642     (!pair-converter
1643       pick-any x y
1644       (!chain [(x @ y in A X (B /\ C))
1645               <==> (x in A & y in B /\ C)           [CPC]
1646               <==> (x in A & y in B & y in C)       [IC]
1647               <==> ((x in A & y in B) & (x in A & y in C)) [prop-taut]
1648               <==> (x @ y in A X B & x @ y in A X C) [CPC]
1649               <==> (x @ y in A X B /\ A X C)         [IC]])))
1650
1651 # Theorem 103:
1652 conclude product-theorem-7 :=
1653   (forall A B C . A X (B \ C) = A X B \ A X C)
1654 pick-any A B C
1655   (!set-identity-intro-direct
1656     (!pair-converter
1657       pick-any x y
1658       (!chain [(x @ y in A X (B \ C))
1659               <==> (x in A & y in B \ C)           [CPC]
1660               <==> (x in A & (y in B | y in C))       [UC]
1661               <==> ((x in A & y in B) | (x in A & y in C)) [prop-taut]
1662               <==> (x @ y in A X B | x @ y in A X C) [CPC]
1663               <==> (x @ y in A X B \ A X C)         [UC]])))
1664
1665 # Theorem 104:
1666 conclude product-theorem-8 :=
1667   (forall A B C . A X (B \ C) = A X B \ A X C)
1668 pick-any A B C
1669   (!set-identity-intro-direct
1670     (!pair-converter
1671       pick-any x y
1672       (!chain [(x @ y in A X (B \ C))
1673               <==> (x in A & y in B \ C)           [CPC]
1674               <==> (x in A & y in B & ~ y in C)       [DC]
1675               <==> ((x in A & y in B) & (~x in A | ~ y in C)) [prop-taut]
1676               <==> ((x in A & y in B) & ~ (x in A & y in C)) [prop-taut]
1677               <==> (x @ y in A X B & ~ x @ y in A X C) [CPC]
1678               <==> (x @ y in A X B \ A X C)         [DC]])))

```



```

1679
1680 define [R R1 R2 R3 R4] :=
1681     [?R:(Set (Pair 'T14 'T15)) ?R1:(Set (Pair 'T16 'T17))
1682       ?R2:(Set (Pair 'T18 'T19)) ?R3:(Set (Pair 'T20 'T21))
1683       ?R4:(Set (Pair 'T22 'T23))]
1684
1685 ===== RELATION DOMAINS AND RANGES
1686
1687 declare dom: (S, T) [(Set (Pair S T))] -> (Set S) [150 [1st->set]]
1688
1689 assert* dom-def :=
1690     [(dom null = null)
1691       (dom x @ _ ++ t = x ++ dom t)]
1692
1693 (eval dom [( 'a @ 1) ('b @ 2) ('c @ 98)])
1694
1695 declare range: (S, T) [(Set (Pair S T))] -> (Set T) [150 [1st->set]]
1696
1697 assert* range-def :=
1698     [(range null = null)
1699       (range _ @ y ++ t = y ++ range t)]
1700
1701 (eval range [( 'a @ 1) ('b @ 2) ('c @ 98)])
1702
1703 conclude in-dom-lemma-1 :=
1704     (forall R a x y . a = x ==> a in dom x @ y ++ R)
1705 pick-any R a x y
1706     (!chain [(a = x) ==> (a in x ++ dom R) [in-def]
1707             ==> (a in dom x @ y ++ R) [dom-def]])
1708
1709 conclude in-range-lemma-1 :=
1710     (forall R a x y . a = y ==> a in range x @ y ++ R)
1711 pick-any R a x y
1712     (!chain [(a = y) ==> (a in y ++ range R) [in-def]
1713             ==> (a in range x @ y ++ R) [range-def]])
1714
1715 conclude in-dom-lemma-2 :=
1716     (forall R x a b . x in dom R ==> x in dom a @ b ++ R)
1717 pick-any R x a b
1718     (!chain [(x in dom a @ b ++ R)
1719             <== (x in a ++ dom R) [dom-def]
1720             <== (x in dom R) [in-def]])
1721
1722 conclude in-range-lemma-2 :=
1723     (forall R y a b . y in range R ==> y in range a @ b ++ R)
1724 pick-any R y a b
1725     (!chain [(y in range a @ b ++ R)
1726             <== (y in b ++ range R) [range-def]
1727             <== (y in range R) [in-def]])
1728
1729
1730 conclude dom-characterization :=
1731     (forall R x . x in dom R <==> exists y . x @ y in R)
1732 by-induction dom-characterization {
1733     null => pick-any x
1734             (!chain [(x in dom null)
1735                     <==> (x in null) [dom-def]
1736                     <==> false [NC]
1737                     <==> (exists y . false) [taut]
1738                     <==> (exists y . x @ y in null) [NC]])
1739
1740 | (R as (insert (pair a:'S b) t)) =>
1741     let {IH := (forall x . x in dom t <==> exists y . x @ y in t)}
1742         pick-any x:'S
1743         let {p1 := assume hyp := (x in dom R)
1744             (!cases (!chain<- [(x = a | x in dom t)
1745                               <== (x in a ++ dom t) [in-def]
1746                               <== hyp [dom-def]])
1747
1748         assume casel := (x = a)

```

```

1749         (!chain-> [true ==> (a @ b in R) [in-lemma-1]
1750                   ==> (x @ b in R) [case1]
1751                   ==> (exists y . x @ y in R) [existence]]))
1752
1753         assume case2 := (x in dom t)
1754         (!chain-> [case2 ==> (exists y . x @ y in t) [IH]
1755                   ==> (exists y . x @ y in R) [in-def]]));
1756     p2 := (!chain [(exists y . x @ y in R)
1757                  ==> (exists y . x @ y = a @ b | x @ y in t) [in-def]
1758                  ==> (exists y . x = a | x @ y in t) [pair-axioms]
1759                  ==> (exists y . x in dom R | x @ y in t) [in-dom-lemma-1]
1760                  ==> (exists y . x in dom R | exists z . x @ z in t) [in-dom-lemma-1 taut]
1761                  ==> (exists y . x in dom R | x in dom t) [IH]
1762                  ==> (exists y . x in dom R | x in dom R) [in-dom-lemma-2]
1763                  ==> (x in dom R) [taut]])
1764     }
1765     (!equiv p1 p2)
1766 }
1767
1768 define DOMC := dom-characterization
1769
1770 conclude range-characterization :=
1771 (forall R y . y in range R <==> exists x . x @ y in R)
1772 by-induction range-characterization {
1773   null => pick-any y
1774         (!chain [(y in range null)
1775                  <==> (y in null) [range-def]
1776                  <==> false [NC]
1777                  <==> (exists y . false) [taut]
1778                  <==> (exists x . x @ y in null) [NC]])
1779
1780 | (R as (insert (pair a b:'T) t)) =>
1781   let {IH := (forall y . y in range t <==> exists x . x @ y in t)}
1782     pick-any y:'T
1783     let {p1 := assume hyp := (y in range R)
1784          (|cases (!chain<- [(y = b | y in range t)
1785                            <== (y in b ++ range t) [in-def]
1786                            <== hyp [range-def]])
1787
1788          assume case1 := (y = b)
1789          (!chain-> [true ==> (a @ b in R) [in-lemma-1]
1790                  ==> (a @ y in R) [case1]
1791                  ==> (exists x . x @ y in R) [existence])])
1792
1793          assume case2 := (y in range t)
1794          (!chain-> [case2 ==> (exists x . x @ y in t) [IH]
1795                  ==> (exists x . x @ y in R) [in-def]]));
1796   p2 := (!chain [(exists x . x @ y in R)
1797                  ==> (exists x . x @ y = a @ b | x @ y in t) [in-def]
1798                  ==> (exists x . y = b | x @ y in t) [pair-axioms]
1799                  ==> (exists x . y in range R | x @ y in t) [in-range-lemma-1]
1800
1801                  ==> (exists x . y in range R | exists z . z @ y in t) [in-range-lemma-1 taut]
1802
1803                  ==> (exists x . y in range R | y in range t) [IH]
1804
1805                  ==> (exists x . y in range R | y in range R) [in-range-lemma-2]
1806                  ==> (y in range R) [taut]])
1807   }
1808   (!equiv p1 p2)
1809 }
1810
1811 define RANC := range-characterization
1812
1813 conclude dom-theorem-1 :=
1814 (forall R1 R2 . dom (R1 \ / R2) = dom R1 \ / dom R2)
1815 pick-any R1 R2
1816 (!set-identity-intro-direct
1817  pick-any x
1818  (!chain

```

```

1819      [(x in dom (R1 \ / R2))
1820 <==> (exists y . x @ y in R1 \ / R2)           [DOMC]
1821 <==> (exists y . x @ y in R1 | x @ y in R2)     [UC]
1822 <==> ((exists y . x @ y in R1) | (exists y . x @ y in R2)) [taut]
1823 <==> (x in dom R1 | x in dom R2)               [DOMC]
1824 <==> (x in dom R1 \ / dom R2)                 [UC]])
1825
1826
1827 conclude range-theorem-1 :=
1828   (forall R1 R2 . range (R1 \ / R2) = range R1 \ / range R2)
1829 pick-any R1 R2
1830   (!set-identity-intro-direct
1831     pick-any y
1832     (!chain [(y in range (R1 \ / R2))
1833 <==> (exists x . x @ y in R1 \ / R2)           [RANC]
1834 <==> (exists x . x @ y in R1 | x @ y in R2)     [UC]
1835 <==> ((exists x . x @ y in R1) | (exists x . x @ y in R2)) [taut]
1836 <==> (y in range R1 | y in range R2)           [RANC]
1837 <==> (y in range R1 \ / range R2)             [UC]]))
1838
1839
1840 conclude dom-theorem-2 :=
1841   (forall R1 R2 . dom (R1 /\ R2) subset dom R1 /\ dom R2)
1842 pick-any R1 R2
1843   (!subset-intro
1844     pick-any x
1845     (!chain [(x in dom (R1 /\ R2))
1846 ==> (exists y . x @ y in R1 /\ R2) [DOMC]
1847 ==> (exists y . x @ y in R1 & x @ y in R2) [IC]
1848 ==> ((exists y . x @ y in R1) & (exists y . x @ y in R2)) [taut]
1849 ==> (x in dom R1 & x in dom R2) [DOMC]
1850 ==> (x in dom R1 /\ dom R2) [IC]]))
1851
1852 (falsify (forall R1 R2 . dom (R1 /\ R2) = dom R1 /\ dom R2) 10)
1853
1854 conclude range-theorem-2 :=
1855   (forall R1 R2 . range (R1 /\ R2) subset range R1 /\ range R2)
1856 pick-any R1 R2
1857   (!subset-intro
1858     pick-any y
1859     (!chain [(y in range (R1 /\ R2))
1860 ==> (exists x . x @ y in R1 /\ R2) [RANC]
1861 ==> (exists x . x @ y in R1 & x @ y in R2) [IC]
1862 ==> ((exists x . x @ y in R1) & (exists x . x @ y in R2)) [taut]
1863 ==> (y in range R1 & y in range R2) [RANC]
1864 ==> (y in range R1 /\ range R2) [IC]]))
1865
1866
1867 conclude dom-theorem-3 :=
1868   (forall R1 R2 . dom R1 \ dom R2 subset dom (R1 \ R2))
1869 pick-any R1 R2
1870   (!subset-intro
1871     pick-any x
1872     assume hyp := (x in dom R1 \ dom R2)
1873     let {lemma := (!chain-> [hyp ==> (x in dom R1 & ~ x in dom R2) [DC]])}
1874     pick-witness w for (!chain-> [lemma ==> (x in dom R1) [left-and]
1875 ==> (exists y . x @ y in R1) [DOMC]])
1876     (!chain-> [lemma ==> (~ x in dom R2) [right-and]
1877 ==> (~ exists y . x @ y in R2) [DOMC]
1878 ==> (forall y . ~ x @ y in R2) [qn]
1879 ==> (~ x @ w in R2) [(uspec with w)]
1880 ==> (x @ w in R1 & ~ x @ w in R2) [augment]
1881 ==> (exists y . x @ y in R1 & ~ x @ y in R2) [existence]
1882 ==> (exists y . x @ y in R1 \ R2) [DC]
1883 ==> (x in dom (R1 \ R2)) [DOMC]]))
1884
1885
1886 conclude range-theorem-3 :=
1887   (forall R1 R2 . range R1 \ range R2 subset range (R1 \ R2))
1888 pick-any R1 R2

```

```

1889 (!subset-intro
1890   pick-any y
1891   assume hyp := (y in range R1 \ range R2)
1892   let {lemma := (!chain-> [hyp ==> (y in range R1 & ~ y in range R2) [DC]])}
1893   pick-witness w for (!chain-> [lemma ==> (y in range R1) [left-and]
1894                               ==> (exists x . x @ y in R1) [RANC]])
1895     (!chain-> [lemma ==> (~ y in range R2) [right-and]
1896               ==> (~ exists x . x @ y in R2) [RANC]
1897               ==> (forall x . ~ x @ y in R2) [qn]
1898               ==> (~ w @ y in R2) [(uspec with w)]
1899               ==> (w @ y in R1 & ~ w @ y in R2) [augment]
1900               ==> (exists x . x @ y in R1 & ~ x @ y in R2) [existence]
1901               ==> (exists x . x @ y in R1 \ R2) [DC]
1902               ==> (y in range (R1 \ R2)) [RANC]]))
1903
1904
1905
1906 declare conv: (S, T) [(Set (Pair S T))] -> (Set (Pair T S)) [210 [lst->set]]
1907 define -- := conv
1908
1909 assert* conv-def :=
1910   [(-- null = null)
1911     (-- x @ y ++ t = y @ x ++ -- t)]
1912
1913
1914 define pair-lemma-1 := Pair.pair-theorem-2
1915
1916 conclude converse-characterization :=
1917   (forall R x y . x @ y in -- R <==> y @ x in R)
1918 by-induction converse-characterization {
1919   null => pick-any x y
1920     (!chain [(x @ y in -- null)
1921              <==> (x @ y in null) [conv-def]
1922              <==> false [NC]
1923              <==> (y @ x in null) [NC]])
1924
1925 | (R as (insert (pair a b) t)) =>
1926   let {
1927     IH := (forall x y . x @ y in -- t <==> y @ x in t)
1928     pick-any x y
1929     (!chain [(x @ y in -- R)
1930              <==> (x @ y in b @ a ++ -- t) [conv-def]
1931              <==> (x @ y = b @ a | x @ y in -- t) [in-def]
1932              <==> (y @ x = a @ b | x @ y in -- t) [pair-lemma-1]
1933              <==> (y @ x = a @ b | y @ x in t) [IH]
1934              <==> (y @ x in R) [in-def]])
1935   }
1936
1937
1938 conclude converse-theorem-1 :=
1939   (forall R . -- -- R = R)
1940 by-induction converse-theorem-1 {
1941   null => (!chain [(-- -- null) = (-- null) [conv-def]
1942                  = null [conv-def]])
1943 | (R as (insert (pair x y) t)) =>
1944   let {IH := (-- -- t = t)}
1945     (!chain [(-- -- x @ y ++ t)
1946              = (-- (y @ x ++ -- t)) [conv-def]
1947              = (x @ y ++ -- -- t) [conv-def]
1948              = (x @ y ++ t) [IH]])
1949 }
1950
1951 conclude converse-theorem-2 :=
1952   (forall R1 R2 . -- (R1 /\ R2) = -- R1 /\ -- R2)
1953 pick-any R1 R2
1954 (!set-identity-intro-direct
1955  (!pair-converter
1956   pick-any x y
1957   (!chain [(x @ y in -- (R1 /\ R2))
1958            <==> (y @ x in R1 /\ R2) [converse-characterization]

```

```

1959         <==> (y @ x in R1 & y @ x in R2)           [IC]
1960         <==> (x @ y in -- R1 & x @ y in -- R2) [converse-characterization]
1961         <==> (x @ y in -- R1 /\ -- R2)           [IC]))))
1962
1963
1964 conclude converse-theorem-3 :=
1965   (forall R1 R2 . -- (R1 \/ R2) = -- R1 \/ -- R2)
1966   pick-any R1 R2
1967     (!set-identity-intro-direct
1968       (!pair-converter
1969         pick-any x y
1970           (!chain [(x @ y in -- (R1 \/ R2))
1971             <==> (y @ x in R1 \/ R2)           [converse-characterization]
1972             <==> (y @ x in R1 | y @ x in R2)     [UC]
1973             <==> (x @ y in -- R1 | x @ y in -- R2) [converse-characterization]
1974             <==> (x @ y in -- R1 \/ -- R2)     [UC]])))
1975
1976
1977 conclude converse-theorem-4 :=
1978   (forall R1 R2 . -- (R1 \ R2) = -- R1 \ -- R2)
1979   pick-any R1 R2
1980     (!set-identity-intro-direct
1981       (!pair-converter
1982         pick-any x y
1983           (!chain [(x @ y in -- (R1 \ R2))
1984             <==> (y @ x in R1 \ R2)           [converse-characterization]
1985             <==> (y @ x in R1 & ~ y @ x in R2) [DC]
1986             <==> (x @ y in -- R1 & ~ x @ y in -- R2) [converse-characterization]
1987             <==> (x @ y in -- R1 \ -- R2)     [DC]])))
1988
1989
1990 declare composed-with: (S1, S2, S3) [(Pair S1 S2) (Set (Pair S2 S3))] -> (Set (Pair S1 S3)) [200 [id lst->set]]
1991
1992 assert* composed-with-def :=
1993   [(_ composed-with null = null)
1994     (x @ y composed-with z @ w ++ t = x @ w ++ (x @ y composed-with t) <== y = z)
1995     (x @ y composed-with z @ w ++ t = x @ y composed-with t <== y /= z)]
1996
1997
1998
1999 (eval 1 @ 2 composed-with [(2 @ 5) (7 @ 8) (2 @ 3)])
2000 (eval 1 @ 2 composed-with [(7 @ 8) (9 @ 10)])
2001 (eval 1 @ 2 composed-with [])
2002
2003 conclude composed-with-characterization :=
2004   (forall R x y z w . w @ z in x @ y composed-with R <==> w = x & y @ z in R)
2005 by-induction composed-with-characterization {
2006   (R as null) => pick-any x y z w
2007     (!chain [(w @ z in x @ y composed-with null)
2008       <==> (w @ z in null) [composed-with-def]
2009       <==> false [NC]
2010       <==> (w = x & y @ z in null) [prop-taut NC]])
2011
2012 | (R as (insert (pair a b) t)) =>
2013   pick-any x y z w
2014     let {IH := (forall x y z w . w @ z in x @ y composed-with t <==> w = x & y @ z in t)}
2015     (!two-cases
2016       assume case1 := (y = a)
2017         (!chain [(w @ z in x @ y composed-with a @ b ++ t)
2018           <==> (w @ z in x @ b ++ (x @ y composed-with t)) [composed-with-def]
2019           <==> (w @ z = x @ b | w @ z in x @ y composed-with t) [in-def]
2020           <==> (w @ z = x @ b | (w = x & y @ z in t)) [IH]
2021           <==> (w = x & z = b | w = x & y @ z in t) [pair-axioms]
2022           <==> (w = x & y = a & z = b | w = x & y @ z in t) [augment]
2023           <==> (w = x & y @ z = a @ b | w = x & y @ z in t) [pair-axioms]
2024           <==> (w = x & (y @ z = a @ b | y @ z in t)) [prop-taut]
2025           <==> (w = x & y @ z in R) [in-def]])
2026       assume case2 := (y /= a)
2027         (!iff-comm
2028           (!chain [(w = x & y @ z in R)

```

```

2029 <==> (w = x & (y @ z = a @ b | y @ z in t)) [in-def]
2030 <==> (w = x & (y = a & z = b | y @ z in t)) [pair-axioms]
2031 <==> (w = x & (case2 & y = a & z = b | y @ z in t)) [augment]
2032 <==> (w = x & (false | y @ z in t)) [prop-taut]
2033 <==> (w = x & y @ z in t) [prop-taut]
2034 <==> (w @ z in x @ y composed-with t) [IH]
2035 <==> (w @ z in x @ y composed-with R) [composed-with-def]))
2036 }
2037
2038 conclude composed-with-characterization' :=
2039   (forall R x y z . x @ z in x @ y composed-with R <==> y @ z in R)
2040 pick-any R x y z
2041   (!chain [(x @ z in x @ y composed-with R)
2042     <==> (x = x & y @ z in R) [composed-with-characterization]
2043     <==> (y @ z in R) [augment]])]
2044
2045
2046 declare o: (S1, S2, S3) [(Set (Pair S1 S2)) (Set (Pair S2 S3))] -> (Set (Pair S1 S3)) [200 [1st->set 1st->set]]
2047
2048 assert* o-def :=
2049   [(null o _ = null)
2050     (x @ y ++ t o R = x @ y composed-with R \ / t o R)]
2051
2052 (eval [(nyc @ 'boston) ('houston @ 'dallas) ('austin @ 'dc)] o
2053
2054   [(('boston @ 'montreal) ('dallas @ 'chicago) ('dc @ 'nyc)] o
2055   [(('chicago @ 'seattle) ('montreal @ 'london)])]
2056
2057 let {R1 := [(('nyc @ 'boston) ('austin @ 'dc)];
2058   R2 := [(('boston @ 'montreal) ('dc @ 'chicago) ('chicago @ 'seattle))]}
2059   (eval R1 o R2)
2060
2061 conclude o-characterization :=
2062   (forall R1 R2 x z . x @ z in R1 o R2 <==> exists y . x @ y in R1 & y @ z in R2)
2063 by-induction o-characterization {
2064   (R1 as null) => pick-any R2 x z
2065     (!chain [(x @ z in R1 o R2)
2066       <==> (x @ z in null) [o-def]
2067       <==> false [NC]
2068       <==> (exists y . false & y @ z in R2) [(method (p q) (!force q))]
2069       <==> (exists y . x @ y in null & y @ z in R2) [NC (method (p q) (!force q))]])]
2070 | (R1 as (insert (pair a b) t)) =>
2071   pick-any R2 x z
2072     let {IH := (forall R2 x z . x @ z in t o R2 <==> exists y . x @ y in t & y @ z in R2)}
2073       let {dir1 := assume hyp := (x @ z in R1 o R2)
2074         (!cases (!chain-> [hyp
2075           ==> (x @ z in a @ b composed-with R2 \ / t o R2) [o-def]
2076           ==> (x @ z in a @ b composed-with R2 | x @ z in t o R2) [UC]
2077           ==> (x @ z in a @ b composed-with R2 | exists y . x @ y in t & y @ z in R2) [IH]
2078           ==> (x @ z in a @ b composed-with R2 | exists y . x @ y in R1 & y @ z in R2) [in]
2079           ==> (x = a & b @ z in R2 | exists y . x @ y in R1 & y @ z in R2) [composed-with]
2080           assume case1 := (x = a & b @ z in R2)
2081             (!chain-> [true ==> (a @ b in R1) [in-lemma-1]
2082               ==> (x @ b in R1) [case1]
2083               ==> (x @ b in R1 & b @ z in R2) [augment]
2084               ==> (exists y . x @ y in R1 & y @ z in R2) [taut]])]
2085           assume case2 := (exists y . x @ y in R1 & y @ z in R2)
2086             (!claim case2));
2087         dir2 := assume hyp := (exists y . x @ y in R1 & y @ z in R2)
2088           pick-witness y for hyp
2089             (!cases (!chain-> [(x @ y in R1)
2090               ==> (x @ y = a @ b | x @ y in t) [in-def]])]
2091           assume case1 := (x @ y = a @ b)
2092             let {_ := (!chain-> [case1 ==> (x = a) [pair-axioms]]);
2093               _ := (!chain-> [case1 ==> (y = b) [pair-axioms]])]
2094             (!chain-> [(x = a)
2095               ==> (x = a & y @ z in R2) [augment]
2096               ==> (x = a & b @ z in R2) [(y = b)]
2097               ==> (x @ z in a @ b composed-with R2) [composed-with-characterization]
2098               ==> (x @ z in a @ b composed-with R2 \ / t o R2) [UC]

```

```

2099             ==> (x @ z in R1 o R2) [o-def]])
2100   assume case2 := (x @ y in t
2101     (!chain-> [case2
2102       ==> (x @ y in t & y @ z in R2) [augment]
2103       ==> (exists y . x @ y in t & y @ z in R2) [existence]
2104       ==> (x @ z in t o R2) [IH]
2105       ==> (x @ z in a @ b composed-with R2 | x @ z in t o R2) [prop-taut]
2106       ==> (x @ z in a @ b composed-with R2 \ / t o R2) [UC]
2107       ==> (x @ z in R1 o R2) [o-def]])
2108     )
2109   (!equiv dir1 dir2)
2110 }
2111
2112
2113
2114
2115 conclude compose-theorem-1 :=
2116   (forall R1 R2 . dom R1 o R2 subset dom R1)
2117 pick-any R1 R2
2118   (!subset-intro
2119     pick-any x
2120     (!chain [(x in dom R1 o R2)
2121       ==> (exists y . x @ y in R1 o R2) [dom-characterization]
2122       ==> (exists y . exists z . x @ z in R1 & z @ y in R2) [o-characterization]
2123       ==> (exists y . exists z . x @ z in R1) [taut]
2124       ==> (exists y . x in dom R1) [dom-characterization]
2125       ==> (x in dom R1) [taut]]))
2126
2127 conclude compose-theorem-2 :=
2128   (forall R1 R2 R3 R4 . R1 subset R2 & R3 subset R4 ==> R1 o R3 subset R2 o R4)
2129 pick-any R1:(Set (Pair 'S 'T)) R2:(Set (Pair 'S 'T))
2130   R3:(Set (Pair 'T 'U)) R4:(Set (Pair 'T 'U))
2131 assume hyp := (R1 subset R2 & R3 subset R4)
2132   (!subset-intro
2133     (!pair-converter
2134       pick-any x y
2135       (!chain [(x @ y in R1 o R3)
2136         ==> (exists z . x @ z in R1 & z @ y in R3) [o-characterization]
2137         ==> (exists z . x @ z in R2 & z @ y in R3) [SC]
2138         ==> (exists z . x @ z in R2 & z @ y in R4) [SC]
2139         ==> (x @ y in R2 o R4) [o-characterization]]))
2140
2141 conclude compose-theorem-3 :=
2142   (forall R1 R2 R3 . R1 o (R2 \ / R3) = R1 o R2 \ / R1 o R3)
2143 pick-any R1 R2 R3
2144   (!set-identity-intro-direct
2145     (!pair-converter
2146       pick-any x y
2147       (!chain [(x @ y in R1 o (R2 \ / R3))
2148         <==> (exists z . x @ z in R1 & z @ y in R2 \ / R3) [o-characterization]
2149         <==> (exists z . x @ z in R1 & (z @ y in R2 | z @ y in R3)) [UC]
2150         <==> (exists z . x @ z in R1 & z @ y in R2 | x @ z in R1 & z @ y in R3) [prop-taut]
2151         <==> ((exists z . x @ z in R1 & z @ y in R2) | (exists z . x @ z in R1 & z @ y in R3)) [taut]
2152         <==> (x @ y in R1 o R2 | x @ y in R1 o R3) [o-characterization]
2153         <==> (x @ y in R1 o R2 \ / R1 o R3) [UC]]))
2154
2155 conclude compose-theorem-4 :=
2156   (forall R1 R2 R3 . R1 o (R2 /\ R3) subset R1 o R2 /\ R1 o R3)
2157 pick-any R1 R2 R3
2158   (!subset-intro
2159     (!pair-converter
2160       pick-any x y
2161       (!chain [(x @ y in R1 o (R2 /\ R3))
2162         ==> (exists z . x @ z in R1 & z @ y in R2 /\ R3) [o-characterization]
2163         ==> (exists z . x @ z in R1 & (z @ y in R2 & z @ y in R3)) [IC]
2164         ==> (exists z . (x @ z in R1 & z @ y in R2) & (x @ z in R1 & z @ y in R3)) [prop-taut]
2165         ==> ((exists z . x @ z in R1 & z @ y in R2) & (exists z . x @ z in R1 & z @ y in R3)) [taut]
2166         ==> (x @ y in R1 o R2 & x @ y in R1 o R3) [o-characterization]
2167         ==> (x @ y in R1 o R2 /\ R1 o R3) [IC]]))
2168

```

```

2169
2170 conclude compose-theorem-5 :=
2171   (forall R1 R2 R3 . R1 o R2 \ R1 o R3 subset R1 o (R2 \ R3))
2172 pick-any R1 R2 R3
2173   (!subset-intro
2174     (!pair-converter
2175       pick-any x y
2176         (!chain [(x @ y in R1 o R2 \ R1 o R3)
2177           ==> (x @ y in R1 o R2 & ~ x @ y in R1 o R3) [DC]
2178           ==> ((exists z . x @ z in R1 & z @ y in R2) & ~ (exists z . x @ z in R1 & z @ y in R3)) [o-characterization]
2179           ==> (exists z . x @ z in R1 & z @ y in R2 & ~ z @ y in R3) [taut]
2180           ==> (exists z . x @ z in R1 & z @ y in R2 \ R3) [DC]
2181           ==> (x @ y in R1 o (R2 \ R3)) [o-characterization]])))
2182
2183 conclude composition-assoc :=
2184   (forall R1 R2 R3 . R1 o R2 o R3 = (R1 o R2) o R3)
2185 pick-any R1 R2 R3
2186   (!set-identity-intro-direct
2187     (!pair-converter
2188       pick-any x y
2189         (!chain [(x @ y in R1 o R2 o R3)
2190           <==> (exists z . x @ z in R1 & z @ y in R2 o R3) [o-characterization]
2191           <==> (exists z . x @ z in R1 & exists w . z @ w in R2 & w @ y in R3) [o-characterization]
2192           <==> (exists w z . x @ z in R1 & z @ w in R2 & w @ y in R3) [taut]
2193           <==> (exists w . (exists z . x @ z in R1 & z @ w in R2) & w @ y in R3) [taut]
2194           <==> (exists w . x @ w in R1 o R2 & w @ y in R3) [o-characterization]
2195           <==> (x @ y in (R1 o R2) o R3) [o-characterization]])))
2196
2197 conclude compose-theorem-6 :=
2198   (forall R1 R2 . -- (R1 o R2) = -- R2 o -- R1)
2199 pick-any R1 R2
2200   (!set-identity-intro-direct
2201     (!pair-converter
2202       pick-any x y
2203         (!chain [(x @ y in -- (R1 o R2))
2204           <==> (y @ x in R1 o R2) [converse-characterization]
2205           <==> (exists z . y @ z in R1 & z @ x in R2) [o-characterization]
2206           <==> (exists z . z @ y in -- R1 & x @ z in -- R2) [converse-characterization]
2207           <==> (exists z . x @ z in -- R2 & z @ y in -- R1) [prop-taut]
2208           <==> (x @ y in -- R2 o -- R1) [o-characterization]])))
2209
2210
2211 declare restrict1: (S, T) [(Set (Pair S T)) S] -> (Set (Pair S T)) [200 [lst->set id]]
2212
2213 assert* restrict1-def :=
2214   [(null restrict1 _ = null)
2215     (x @ y ++ t restrict1 z = x @ y ++ (t restrict1 z) <== x = z)
2216     (x @ y ++ t restrict1 z = t restrict1 z <== x /= z)]
2217
2218 (eval [(1 @ 'foo) (2 @ 'b) (1 @ 'bar)] restrict1 1)
2219
2220 define restrict1-characterization :=
2221   (forall R x y a . x @ y in R restrict1 a <==> x @ y in R & x = a)
2222
2223 (define ^1 restrict1)
2224
2225 conclude restrict1-lemma :=
2226   (forall R x y a . x @ y in R & x = a ==> x @ y in R ^1 a)
2227 by-induction restrict1-lemma {
2228   (R as null) => pick-any x y a
2229     (!chain [(x @ y in R & x = a)
2230       ==> (x @ y in R) [left-and]
2231       ==> false [NC]
2232       ==> (x @ y in R ^1 a) [prop-taut]])
2233 | (R as (insert (pair x' y') t)) =>
2234   let {IH := (forall x y a . x @ y in t & x = a ==> x @ y in t ^1 a)}
2235     pick-any x y a
2236     assume hyp := (x @ y in R & x = a)
2237     (!two-cases
2238       assume casel := (x' = a)

```



```

2239     (!chain-> [hyp
2240       ==> ((x @ y = x' @ y' | x @ y in t) & x = a) [in-def]
2241       ==> (x @ y = x' @ y' & x = a | x @ y in t & x = a) [prop-taut]
2242       ==> (x @ y in x' @ y' ++ (t ^1 a) & x = a | x @ y in t & x = a) [in-def]
2243       ==> (x @ y in R ^1 a & x = a | x @ y in t & x = a) [restrict1-def]
2244       ==> (x @ y in R ^1 a & x = a | x @ y in t ^1 a) [IH]
2245       ==> (x @ y in R ^1 a & x = a | x @ y in x' @ y' ++ (t ^1 a)) [in-def]
2246       ==> (x @ y in R ^1 a & x = a | x @ y in R ^1 a) [restrict1-def]
2247       ==> (x @ y in R ^1 a) [prop-taut]])
2248   assume case2 := (x' /= a)
2249   (!cases (!chain-> [hyp
2250     ==> ((x @ y = x' @ y' | x @ y in t) & x = a) [in-def]
2251     ==> ((x = x' & y = y' | x @ y in t) & x = a) [pair-axioms]
2252     ==> (x = x' & y = y' & x = a | x @ y in t & x = a) [prop-taut]])
2253   assume hyp1 := (x = x' & y = y' & x = a)
2254   let { _ := (!absurd (!chain-> [hyp1 ==> (x = a)
2255     ==> (x' = a)])
2256     case2)}
2257   (!from-false (x @ y in R ^1 a))
2258   assume hyp2 := (x @ y in t & x = a)
2259   (!chain-> [hyp2 ==> (x @ y in t ^1 a) [IH]
2260     ==> (x @ y in R ^1 a) [restrict1-def]]))
2261 }
2262
2263 by-induction restrict1-characterization {
2264   (R as null) => pick-any x y a
2265     (!chain [(x @ y in R ^1 a)
2266       <==> (x @ y in null) [restrict1-def]
2267       <==> false [NC]
2268       <==> (false & x = a) [prop-taut]
2269       <==> (x @ y in R & x = a) [NC]])
2270 | (R as (insert (pair x' y') t)) =>
2271   pick-any x y a
2272   let {IH := (forall x y a . x @ y in t ^1 a <==> x @ y in t & x = a);
2273     goal := (x @ y in R ^1 a <==> x @ y in R & x = a);
2274     dir1 := assume hyp := (x @ y in R ^1 a)
2275       (!two-cases
2276         assume case1 := (x' = a)
2277         (!cases (!chain-> [hyp
2278           ==> (x @ y in x' @ y' ++ (t ^1 a)) [restrict1-def]
2279           ==> (x @ y = x' @ y' | x @ y in t ^1 a) [in-def]])
2280         assume hyp1a := (x @ y = x' @ y')
2281           (!both (!chain-> [hyp1a ==> (x @ y in R) [in-def]])
2282             (!chain-> [hyp1a ==> (x = x') [pair-axioms]
2283               ==> (x = a) [case1]]))
2284           (!chain [(x @ y in t ^1 a) ==> (x @ y in t & x = a) [IH]
2285             ==> (x @ y in R & x = a) [in-def]])
2286         assume case2 := (x' /= a)
2287           (!chain-> [hyp ==> (x @ y in t ^1 a) [restrict1-def]
2288             ==> (x @ y in t & x = a) [IH]
2289             ==> (x @ y in R & x = a) [in-def]]));
2290     dir2 := (!chain [(x @ y in R & x = a) ==> (x @ y in R ^1 a) [restrict1-lemma]])
2291     (!equiv dir1 dir2)
2292 }
2293
2294
2295 declare restrict: (S, T) [(Set (Pair S T)) (Set S)] -> (Set (Pair S T)) [200 [lst->set lst->set]]
2296
2297 define ^ := restrict
2298 assert* restrict-def :=
2299 [(R restrict null = null)
2300 (R restrict h ++ t = R restrict1 h \ / R restrict t)]
2301
2302 (eval [(1 @ 'foo) (2 @ 'b) (3 @ 'c) (4 @ 'd) (1 @ 'bar)] ^ [1 2])
2303
2304 conclude restrict-characterization :=
2305 (forall A R x y . x @ y in R restrict A <==> x @ y in R & x in A)
2306 by-induction restrict-characterization {
2307   (A as null) => pick-any R x y
2308     (!chain [(x @ y in R restrict A)

```

```

2309             <==> (x @ y in null)           [restrict-def]
2310             <==> false                       [NC]
2311             <==> (x @ y in R & false)        [prop-taut]
2312             <==> (x @ y in R & x in A)      [NC]])
2313 | (A as (insert h t)) =>
2314   let {IH := (forall R x y . x @ y in R restrict t <==> x @ y in R & x in t)}
2315     pick-any R x y
2316     (!chain [(x @ y in R restrict A)
2317             <==> (x @ y in R ^1 h \ / R restrict t)           [restrict-def]
2318             <==> (x @ y in R ^1 h | x @ y in R restrict t) [UC]
2319             <==> ((x @ y in R & x = h) | x @ y in R restrict t) [restrict1-characterization]
2320             <==> ((x @ y in R & x = h) | x @ y in R & x in t) [IH]
2321             <==> ((x @ y in R) & (x = h | x in t))           [prop-taut]
2322             <==> (x @ y in R & x in A)                       [in-def]])
2323 }
2324
2325 conclude restriction-theorem-1 :=
2326 (forall R A B . A subset B ==> R ^ A subset R ^ B)
2327 pick-any R A B
2328   assume (A subset B)
2329     (!subset-intro
2330       (!pair-converter
2331         pick-any x y
2332         (!chain [(x @ y in R ^ A)
2333                 ==> (x @ y in R & x in A) [restrict-characterization]
2334                 ==> (x @ y in R & x in B) [SC]
2335                 ==> (x @ y in R ^ B)   [restrict-characterization]])))
2336
2337
2338 conclude restriction-theorem-2 :=
2339 (forall R A B . R ^ (A /\ B) = R ^ A /\ R ^ B)
2340 pick-any R A B
2341   (!set-identity-intro-direct
2342     (!pair-converter
2343       pick-any x y
2344       (!chain [(x @ y in R ^ (A /\ B))
2345               <==> (x @ y in R & x in A /\ B)           [restrict-characterization]
2346               <==> (x @ y in R & x in A & x in B)       [IC]
2347               <==> ((x @ y in R & x in A) & (x @ y in R & x in B)) [prop-taut]
2348               <==> (x @ y in R ^ A & x @ y in R ^ B)   [restrict-characterization]
2349               <==> (x @ y in R ^ A /\ R ^ B)           [IC]])))
2350
2351
2352 conclude restriction-theorem-3 :=
2353 (forall R A B . R ^ (A \ / B) = R ^ A \ / R ^ B)
2354 pick-any R A B
2355   (!set-identity-intro-direct
2356     (!pair-converter
2357       pick-any x y
2358       (!chain [(x @ y in R ^ (A \ / B))
2359               <==> (x @ y in R & x in A \ / B)           [restrict-characterization]
2360               <==> (x @ y in R & (x in A | x in B))       [UC]
2361               <==> ((x @ y in R & x in A) | (x @ y in R & x in B)) [prop-taut]
2362               <==> (x @ y in R ^ A | x @ y in R ^ B)   [restrict-characterization]
2363               <==> (x @ y in R ^ A \ / R ^ B)           [UC]])))
2364
2365
2366 conclude restriction-theorem-4 :=
2367 (forall R A B . R ^ (A \ B) = R ^ A \ R ^ B)
2368 pick-any R A B
2369   (!set-identity-intro-direct
2370     (!pair-converter
2371       pick-any x y
2372       (!chain [(x @ y in R ^ (A \ B))
2373               <==> (x @ y in R & x in A \ B)           [restrict-characterization]
2374               <==> (x @ y in R & (x in A & ~ x in B))       [DC]
2375               <==> ((x @ y in R & x in A) & ~ (x @ y in R & x in B)) [prop-taut]
2376               <==> (x @ y in R ^ A & ~ x @ y in R ^ B)   [restrict-characterization]
2377               <==> (x @ y in R ^ A \ R ^ B)           [DC]])))
2378

```

```

2379
2380
2381 conclude restriction-theorem-5 :=
2382   (forall R1 R2 A . (R1 o R2) ^ A = (R1 ^ A) o R2)
2383 pick-any R1 R2 A
2384   (!set-identity-intro-direct
2385     (!pair-converter
2386       pick-any x y
2387         (!chain [(x @ y in (R1 o R2) ^ A)
2388                 <==> (x @ y in R1 o R2 & x in A) [restrict-characterization]
2389                 <==> ((exists z . x @ z in R1 & z @ y in R2) & x in A) [o-characterization]
2390                 <==> (exists z . x @ z in R1 & z @ y in R2 & x in A) [taut]
2391                 <==> (exists z . (x @ z in R1 & x in A) & z @ y in R2) [prop-taut]
2392                 <==> (exists z . x @ z in R1 ^ A & z @ y in R2) [restrict-characterization]
2393                 <==> (x @ y in (R1 ^ A) o R2) [o-characterization]]))
2394
2395
2396 declare image: (S, T) [(Set (Pair S T)) (Set S)] -> (Set T) [** 200 [lst->set lst->set]]
2397
2398 #define ** := image
2399
2400 assert* image-def := [(R ** A = range R ^ A)]
2401
2402 (eval [(1 @ 'a) (2 @ 'b) (3 @ 'c)] ** [1 3])
2403
2404 conclude image-characterization :=
2405   (forall R A y . y in R ** A <==> exists x . x @ y in R & x in A)
2406 pick-any R A y
2407   (!chain [(y in R ** A)
2408             <==> (y in range R ^ A) [image-def]
2409             <==> (exists x . x @ y in R ^ A) [range-characterization]
2410             <==> (exists x . x @ y in R & x in A) [restrict-characterization]])
2411
2412 conclude image-lemma :=
2413   (forall R A x y . x @ y in R & x in A ==> y in R ** A)
2414 pick-any R A x y
2415   (!chain [(x @ y in R & x in A)
2416             ==> (exists x . x @ y in R & x in A) [existence]
2417             ==> (y in R ** A) [image-characterization]])
2418
2419 conclude image-theorem-1 :=
2420   (forall R A B . R ** (A \ / B) = R ** A \ / R ** B)
2421 pick-any R A B
2422   (!set-identity-intro-direct
2423     pick-any y
2424       (!chain [(y in R ** (A \ / B))
2425                 <==> (exists x . x @ y in R & x in A \ / B) [image-characterization]
2426                 <==> (exists x . x @ y in R & (x in A | x in B)) [UC]
2427                 <==> (exists x . (x @ y in R & x in A) | (x @ y in R & x in B)) [prop-taut]
2428                 <==> ((exists x . x @ y in R & x in A) | (exists x . x @ y in R & x in B)) [taut]
2429                 <==> (y in R ** A | y in R ** B) [image-characterization]
2430                 <==> (y in R ** A \ / R ** B) [UC]]))
2431
2432
2433 conclude image-theorem-2 :=
2434   (forall R A B . R ** (A / \ B) subset R ** A / \ R ** B)
2435 pick-any R A B
2436   (!subset-intro
2437     pick-any y
2438       (!chain [(y in R ** (A / \ B))
2439                 ==> (exists x . x @ y in R & x in A / \ B) [image-characterization]
2440                 ==> (exists x . x @ y in R & x in A & x in B) [IC]
2441                 ==> (exists x . (x @ y in R & x in A) & (x @ y in R & x in B)) [prop-taut]
2442                 ==> ((exists x . x @ y in R & x in A) & (exists x . x @ y in R & x in B)) [taut]
2443                 ==> (y in R ** A & y in R ** B) [image-characterization]
2444                 ==> (y in R ** A / \ R ** B) [IC]]))
2445
2446
2447 conclude image-theorem-3 :=
2448   (forall R A B . R ** A \ R ** B subset R ** (A \ B))

```

```

2449 pick-any R A B
2450   (!subset-intro
2451     pick-any y
2452       (!chain [(y in R ** A \ R ** B)
2453         ==> (y in R ** A & ~ y in R ** B) [DC]
2454         ==> ((exists x . x @ y in R & x in A) & ~ (exists x . x @ y in R & x in B)) [image-characterization]
2455         ==> ((exists x . x @ y in R & x in A) & (forall x . x @ y in R ==> ~ x in B)) [taut]
2456         ==> (exists x . x @ y in R & x in A & ~ x in B) [taut]
2457         ==> (exists x . x @ y in R & x in A \ B) [DC]
2458         ==> (y in R ** (A \ B)) [image-characterization]]))
2459
2460 conclude image-theorem-4 :=
2461   (forall R A B . A subset B ==> R ** A subset R ** B)
2462 pick-any R A B
2463   assume hyp := (A subset B)
2464   (!subset-intro
2465     pick-any y
2466       (!chain [(y in R ** A)
2467         ==> (exists x . x @ y in R & x in A) [image-characterization]
2468         ==> (exists x . x @ y in R & x in B) [SC]
2469         ==> (y in R ** B) [image-characterization]]))
2470
2471 conclude image-theorem-5 :=
2472   (forall R A . R ** A = null <==> dom R /\ A = null)
2473 pick-any R A
2474   (!chain [(R ** A = null)
2475     <==> (forall y . ~ y in R ** A) [null-characterization-2]
2476     <==> (forall y . ~ exists x . x @ y in R & x in A) [image-characterization]
2477     <==> (forall x . ~ exists y . x @ y in R & x in A) [taut]
2478     <==> (forall x . ~ ((exists y . x @ y in R) & x in A)) [taut]
2479     <==> (forall x . ~ (x in dom R & x in A)) [dom-characterization]
2480     <==> (forall x . ~ (x in dom R /\ A)) [IC]
2481     <==> (dom R /\ A = null) [null-characterization-2]])
2482
2483
2484 conclude image-theorem-6 :=
2485   (forall R A . dom R /\ A subset -- R ** R ** A)
2486 pick-any R A
2487   (!subset-intro
2488     pick-any x
2489       (!chain [(x in dom R /\ A)
2490         ==> (x in dom R & x in A) [IC]
2491         ==> ((exists y . x @ y in R) & x in A) [dom-characterization]
2492         ==> (exists y . x @ y in R & x @ y in R & x in A) [taut]
2493         ==> (exists y . x @ y in R & y in R ** A) [image-lemma]
2494         ==> (exists y . y @ x in -- R & y in R ** A) [converse-characterization]
2495         ==> (x in -- R ** R ** A) [image-characterization]]))
2496
2497 (falsify (forall R A . dom R /\ A = -- R ** R ** A) 20)
2498
2499 conclude image-theorem-7 :=
2500   (forall R A B . (R ** A) /\ B subset R ** (A /\ -- R ** B))
2501 pick-any R A B
2502   (!subset-intro
2503     pick-any y
2504       (!chain [(y in (R ** A) /\ B)
2505         ==> (y in R ** A & y in B) [IC]
2506         ==> ((exists x . x @ y in R & x in A) & y in B) [image-characterization]
2507         ==> (exists x . x @ y in R & x in A & y in B) [taut]
2508         ==> (exists x . y @ x in -- R & x in A & x @ y in R & y in B) [converse-characterization augment]
2509         ==> (exists x . (y @ x in -- R & y in B) & x in A & x @ y in R) [prop-taut]
2510         ==> (exists x . x in -- R ** B & x in A & x @ y in R) [image-lemma]
2511         ==> (exists x . x @ y in R & (x in A & x in -- R ** B)) [prop-taut]
2512         ==> (exists x . x @ y in R & x in A /\ -- R ** B) [IC]
2513         ==> (y in R ** (A /\ -- R ** B)) [image-characterization]]))
2514
2515
2516 define lemma := (close t /\ (x insert-in-all t) = null)
2517 define lemma2 := (close (forall y . y in t ==> ~ x in y) ==> t /\ (x insert-in-all t) = null)
2518

```

```

2519 declare card: (S) [(Set S)] -> N [[lst->set]]
2520
2521 define S := N.S
2522
2523 assert* card-def :=
2524   [(card null = zero)
2525    (card h ++ t = card t <== h in t)
2526    (card h ++ t = S card t <== ~ h in t)]
2527
2528 transform-output eval [nat->int]
2529
2530 (eval card [1 2 3] \ / [4 7 8])
2531
2532 define [< <=] := [N.< N.<=]
2533
2534 overload + N.+
2535
2536 define card-theorem-1 :=
2537   (card singleton _ = S zero)
2538
2539 conclude card-theorem-2 :=
2540   (forall A x . ~ x in A ==> card A < card x ++ A)
2541 pick-any A x
2542   assume hyp := (~ x in A)
2543     (!chain-> [true ==> (card A < S card A) [N.Less.<S]
2544               ==> (card A < card x ++ A) [card-def]])
2545
2546 conclude minus-card-theorem :=
2547   (forall A x . x in A ==> card A = N.S card A - x)
2548 by-induction minus-card-theorem {
2549   (A as null:(Set.Set 'S)) =>
2550     pick-any x
2551       (!chain [(x in A)
2552               ==> false [NC]
2553               ==> (card A = N.S card A - x) [prop-taut]])
2554 | (A as (insert h:'S t:(Set.Set 'S))) =>
2555   let {IH := (forall x . x in t ==> card t = N.S card (t - x))}
2556     pick-any x:'S
2557       assume hyp := (x in A)
2558         (!two-cases
2559           assume case1 := (x = h)
2560             (!two-cases
2561               assume (h in t)
2562                 let {_ := (!chain-> [(h in t) ==> (x in t) [case1]])}
2563                   (!chain [(card A)
2564                             = (card t) [card-def]
2565                             = (N.S (card t - x)) [IH]
2566                             = (N.S (card A - x)) [remove-def]])
2567                 assume (~ h in t)
2568                   let {_ := (!chain-> [(~ h in t) ==> (~ x in t) [case1]])}
2569                     (!combine-equations
2570                       (!chain [(card A) = (N.S card t)])
2571                       (!chain [(N.S card (A - x))
2572                                 = (N.S card (t - x))
2573                                 = (N.S card t)])))
2574                 assume case2 := (x /= h)
2575                   let {_ := (!chain-> [(x in A)
2576                                       ==> (x = h | x in t) [in-def]
2577                                       ==> (x in t) [(dsyl with case2)]])}
2578                     (!two-cases
2579                       assume (h in t)
2580                         let {_ := (!chain-> [(h in t)
2581                                             ==> (h in t & x /= h) [augment]
2582                                             ==> (h in t - x) [remove-corollary-3]])}
2583                           (!chain [(card A)
2584                                     = (card t) [card-def]
2585                                     = (N.S card (t - x)) [IH]
2586                                     = (N.S card h ++ (t - x)) [card-def]
2587                                     = (N.S card (A - x)) [remove-def]])
2588                           assume (~ h in t)

```

```

2589         let { _ := (!chain-> [(~ h in t) ==> (~ h in t - x) [remove-corollary-4]]) }
2590             (!chain-> [(card t) = (N.S card t - x) [IH]
2591                 ==> (N.S card t = N.S N.S card t - x)
2592                 ==> (card A = N.S N.S card t - x) [card-def]
2593                 ==> (card A = N.S card h ++ (t - x)) [card-def]
2594                 ==> (card A = N.S card A - x) [remove-def]]))
2595     }
2596
2597 define subset-card-theorem :=
2598   (forall A B . A subset B ==> card A <= card B)
2599
2600
2601 by-induction subset-card-theorem {
2602   null => pick-any B:(Set.Set 'S)
2603     assume hyp := (null subset B)
2604       (!chain-> [true ==> (zero <= card B) [N.Less=.zero<=]
2605               ==> (card null:(Set.Set 'S) <= card B) [card-def]])
2606 | (A as (insert h:'S t:(Set.Set 'S))) =>
2607   let {IH := (forall B . t subset B ==> card t <= card B)}
2608     pick-any B:(Set.Set 'S)
2609       assume hyp := (A subset B)
2610         (!two-cases
2611           assume case1 := (in h t)
2612             (!chain-> [hyp ==> (t subset B) [subset-lemma-2]
2613                   ==> (card t <= card B) [IH]
2614                   ==> (card A <= card B) [card-def]])
2615           assume case2 := (~ in h t)
2616             let {t-sub-B := (!chain-> [hyp ==> (t subset B) [subset-lemma-2]]);
2617                 _ := (!chain-> [true
2618                             ==> (in h A) [in-lemma-1]
2619                             ==> (in h B) [SC]])}
2620             (!chain-> [t-sub-B ==> (t subset B & case2) [augment]
2621                   ==> (t subset B - h) [remove-corollary-5]
2622                   ==> (card t <= card B - h) [IH]
2623                   ==> (S card t <= S card B - h)
2624                   ==> (S card t <= card B) [minus-card-theorem]
2625                   ==> (card A <= card B) [card-def]])
2626   }
2627
2628 conclude proper-subset-card-theorem :=
2629   (forall A B . A proper-subset B ==> card A < card B)
2630 pick-any A B
2631   assume hyp := (A proper-subset B)
2632     pick-witness x for (!chain-> [hyp ==> (A subset B & exists x . x in B & ~ x in A) [PSC]
2633                               ==> (exists x . x in B & ~ x in A) [right-and]])
2634     let {L1 := (!chain-> [hyp ==> (A subset B) [PSC]
2635                       ==> (x ++ A subset B) [subset-lemma-1]
2636                       ==> (card x ++ A <= card B) [subset-card-theorem]]);
2637         L2 := (!chain-> [(~ x in A) ==> (card A < card x ++ A) [card-theorem-2]]);
2638         (!chain-> [L1 ==> (L1 & L2) [augment]
2639               ==> (card A < card B) [N.Less=.transitive1]])
2640
2641 conclude intersection-card-theorem-1 :=
2642   (forall A B . card A /\ B <= card A)
2643 pick-any A B
2644   (!chain-> [true ==> (A /\ B subset A) [intersection-subset-theorem]
2645           ==> (card A /\ B <= card A) [subset-card-theorem]])
2646
2647 conclude intersection-card-theorem-2 :=
2648   (forall A B . card A /\ B <= card B)
2649 pick-any A B
2650   (!chain-> [true ==> (A /\ B subset B) [intersection-subset-theorem-2]
2651           ==> (card A /\ B <= card B) [subset-card-theorem]])
2652
2653 conclude intersection-card-theorem-3 :=
2654   (forall A B x . ~ x in A & x in B ==> card (x ++ A) /\ B = N.S card A /\ B)
2655 pick-any A B x
2656   assume hyp := (~ x in A & x in B)
2657   let { _ := (!chain-> [(~ x in A) ==> (~ x in A /\ B) [intersection-lemma-2]]);
2658       (!chain [(card (x ++ A) /\ B)

```

```

2659         = (card x ++ (A /\ B))    [intersection-def]
2660         = (S card A /\ B)        [card-def]])
2661
2662 # by-induction card-lemma-1 {
2663 #   (A as (insert h t)) =>
2664 #     let { _ := (mark 'A) }
2665 #       (!vpf (forall B x . ~ x in A & x in B ==> card (x ++ A) /\ B = N.S card A /\ B) (ab))
2666 # | (A as Set.null) => let { _ := (mark 'B) } (!dhalt)
2667 # }
2668
2669 define card-lemma-2 :=
2670   (forall A B . card A /\ B = ((card A) + (card B)) N.- (card A /\ B))
2671
2672 overload - N.-
2673
2674 conclude num-lemma :=
2675   (forall x y z . (x + y) - z = (S x + y) - S z)
2676 pick-any x:N y:N z:N
2677   (!chain-> [(S x + y) - S z
2678             = (S (x + y) - S z)           [N.Plus.left-nonzero]
2679             = ((x + y) - z)               [N.Minus.axioms]
2680             ==> ((x + y) - z = (S x + y) - S z) [sym]])
2681
2682
2683 conclude lemma-p1 :=
2684   (forall A B x . ~ x in A and x in B ==> card (x ++ A) /\ B = S card A /\ B)
2685 pick-any A B x
2686   assume hyp := (~ x in A & x in B)
2687   let { _ := (!chain-> [ (~ x in A)
2688                       ==> (~ x in A /\ B) [intersection-lemma-2]] ) }
2689   (!chain [(card x ++ A /\ B)
2690           = (card x ++ (A /\ B)) [intersection-def]
2691           = (S card A /\ B)      [card-def]])
2692
2693
2694 conclude lemma-p2 :=
2695   (forall A B x . ~ x in A & ~ x in B ==> A /\ B = (x ++ A) /\ B)
2696 pick-any A B x
2697   assume hyp := (~ x in A & ~ x in B)
2698   (!set-identity-intro-direct
2699     pick-any y
2700       (!equiv assume hyp1 := (y in A /\ B)
2701         let {L1 := (!chain-> [hyp1 ==> (y in A) [IC]
2702                             ==> (y = x | y in A) [alternate]
2703                             ==> (y in x ++ A) [in-def]])};
2704           L2 := (!chain-> [hyp1 ==> (y in B) [IC]])}
2705         (!chain-> [L1
2706                 ==> (L1 & L2) [augment]
2707                 ==> (y in (x ++ A) /\ B) [IC]])
2708         assume hyp2 := (y in (x ++ A) /\ B)
2709         let {L1 := (!chain-> [hyp2 ==> (y in B) [IC]])};
2710           L2 := (!by-contradiction (y =/= x)
2711             assume (y = x)
2712               (!chain-> [(y in B) ==> (x in B) [ (y = x) ]
2713                       ==> (x in B & ~ x in B) [augment]
2714                       ==> false [prop-taut]]))}
2715         (!chain-> [hyp2 ==> (y in x ++ A) [IC]
2716                 ==> (y = x | y in A) [in-def]
2717                 ==> ((y = x | y in A) & y =/= x) [augment]
2718                 ==> ((y = x) & (y =/= x)) | (y in A & y =/= x) [prop-taut]
2719                 ==> (false | y in A & y =/= x) [prop-taut]
2720                 ==> (y in A) [prop-taut]
2721                 ==> (y in A & y in B) [augment]
2722                 ==> (y in A /\ B) [IC]]))
2723
2724 # by-induction card-lemma-2 {
2725 #   null => (!vpf (forall B . card null /\ B = (card null) + (card B) N.- card null /\ B) (ab))
2726 # | (A as (insert h t)) =>
2727 #   let { _ := (mark 'A) }
2728 #     (!vpf (forall B . card A /\ B = (card A) + (card B) N.- card A /\ B) (ab))

```

```

2729
2730 # }
2731
2732 #(falsify card-lemma-1 10)
2733
2734 conclude union-lemma-2 :=
2735   (forall A B x . x ++ (A \ / B) = A \ / x ++ B)
2736 pick-any A B x
2737   (!chain [(x ++ (A \ / B))
2738            = (x ++ (B \ / A)) [union-commutes]
2739            = ((x ++ B) \ / A) [union-def]
2740            = (A \ / (x ++ B)) [union-commutes]])
2741
2742 conclude union-subset-lemma-1 := (forall A B . A subset A \ / B)
2743 pick-any A B
2744   (!subset-intro
2745     pick-any x
2746       (!chain [(x in A) ==> (x in A \ / B) [UC]]))
2747
2748 conclude union-subset-lemma-2 := (forall A B . B subset A \ / B)
2749 pick-any A B
2750   (!subset-intro
2751     pick-any x
2752       (!chain [(x in B) ==> (x in A \ / B) [UC]]))
2753
2754 conclude leq-lemma-1 := (forall x y . x <= x + y)
2755 pick-any x y
2756   (!by-contradiction (x <= x + y)
2757     let {-y<zero := (!chain-> [true ==> (~ y < zero) [N.Less.not-zero]])}
2758     (!chain [(~ x <= x + y)
2759              ==> (x + y < x) [N.Less=.trichotomy1]
2760              ==> (x + y < x + zero) [N.Plus.right-zero]
2761              ==> (y + x < zero + x) [N.Plus.commutative]
2762              ==> (y < zero) [N.Less.Plus-cancellation]
2763              ==> (y < zero & -y<zero) [augment]
2764              ==> false [prop-taut]]))
2765
2766 conclude leq-lemma :=
2767   (forall x y z . x <= y ==> x <= y + z)
2768 pick-any x y z
2769   assume hyp := (x <= y)
2770   (!chain-> [true ==> (y <= y + z) [leq-lemma-1]
2771            ==> (x <= y & y <= y + z) [augment]
2772            ==> (x <= y + z) [N.Less=.transitive]])
2773
2774 conclude minus-lemma :=
2775   (forall x y . y <= x ==> S (x - y) = (S x) - y)
2776 pick-any x:N y:N
2777   assume (y <= x)
2778   let {_ := (!chain-> [(y <= x) ==> (y <= S x) [N.Less=.S2]])}
2779   (!chain-> [(S x) = (S x)
2780            ==> (S ((x - y) + y) = S x) [N.Minus.Plus-Cancel]
2781            ==> (S (x - y) + y = S x) [N.Plus.left-nonzero]
2782            ==> (S (x - y) + y = (S x - y) + y) [N.Minus.Plus-Cancel]
2783            ==> (S (x - y) = S x - y) [N.Plus.=-cancellation]])
2784
2785 conclude union-card :=
2786   (forall A B . card A \ / B = ((card A) + (card B)) - card A /\ B)
2787 by-induction union-card {
2788   null => pick-any B
2789     let {ns := null:(Set.Set 'S)}
2790     (!chain [(card ns \ / B)
2791              = (card B) [union-def]
2792              = ((card B) - zero) [N.Minus.axioms]
2793              = ((card B) - (card ns)) [card-def]
2794              = ((card B) - card ns /\ B) [intersection-def]
2795              = ((zero + card B) - card ns /\ B) [N.Plus.left-zero]
2796              = ((card ns) + (card B)) - card ns /\ B [card-def]])
2797 | (A as (insert h t:(Set.Set 'S))) =>
2798   let {IH := (forall B . card t \ / B = ((card t) + (card B)) - card t /\ B)}

```



```

2799 pick-any B: (Set.Set 'S)
2800   (!two-cases
2801     assume case1 := (h in t)
2802     let { _ := (!chain-> [(h in t) ==> (h in t \ / B) [UC]]);
2803         L1 := (!chain [(card A \ / B)
2804                   = (card h ++ (t \ / B)) [union-def]
2805                   = (card t \ / B) [card-def]
2806                   = (((card t) + (card B)) - (card t /\ B)) [IH]
2807                   = (((card A) + (card B)) - (card t /\ B)) [card-def]])}
2808     (!two-cases
2809       assume (h in B)
2810       let { _ := (!both (h in B) (h in t)) }
2811         (!chain [(card A \ / B)
2812                 = (((card A) + (card B)) - (card t /\ B)) [L1]
2813                 = (((card A) + (card B)) - (card A /\ B)) [intersection-lemma-1]])
2814       assume (~ h in B)
2815         (!chain [(card A \ / B)
2816                 = (((card A) + (card B)) - (card t /\ B)) [L1]
2817                 = (((card A) + (card B)) - (card A /\ B)) [intersection-def]])
2818     assume case2 := (~ h in t)
2819     (!two-cases
2820       assume (h in B)
2821       let { _ := (!chain-> [(h in B) ==> (h in t \ / B) [UC]]);
2822           _ := (!chain-> [(~ h in t) ==> (~ h in t /\ B) [IC]]);
2823           (!chain [(card A \ / B)
2824                   = (card h ++ (t \ / B)) [union-def]
2825                   = (card t \ / B) [card-def]
2826                   = (((card t) + (card B)) - (card t /\ B)) [IH]
2827                   = (((S card t) + (card B)) - (S (card t /\ B))) [num-lemma]
2828                   = (((card A) + (card B)) - (S (card t /\ B))) [card-def]
2829                   = (((card A) + (card B)) - (S (card t /\ B))) [card-def]
2830                   = (((card A) + (card B)) - (card h ++ (t /\ B))) [card-def]
2831                   = (((card A) + (card B)) - (card A /\ B)) [intersection-def]])
2832       assume (~ h in B)
2833       let { _ := (!chain-> [(~ h in t)
2834                           ==> (~ h in t & ~ h in B) [augment]
2835                           ==> (~ (h in t | h in B)) [dm]
2836                           ==> (~ h in t \ / B) [UC]]);
2837           _ := (!chain-> [true
2838                       ==> (card t /\ B <= card t) [intersection-card-theorem-1]
2839                       ==> (card t /\ B <= (card t) + (card B)) [leq-lemma]])}
2840       (!chain [(card A \ / B)
2841               = (card h ++ (t \ / B)) [union-def]
2842               = (S card t \ / B) [card-def]
2843               = (S (((card t) + (card B)) - card t /\ B)) [IH]
2844               = ((S ((card t) + (card B))) - (card t /\ B)) [minus-lemma]
2845               = ((S ((card t) + (card B))) - (card A /\ B)) [lemma-p2]
2846               = (((S card t) + card B) - (card A /\ B)) [N.Plus.left-nonzero]
2847               = (((card A) + card B) - (card A /\ B)) [card-def]])
2848   }
2849
2850
2851 conclude diff-card-lemma :=
2852   (forall A B . card A = (card A \ B) + (card A /\ B))
2853 pick-any A B
2854   (!chain-> [true ==> (A = (A \ B) \ / (A /\ B)) [diff-theorem-12]
2855             ==> (card A = card (A \ B) \ / (A /\ B))
2856             ==> (card A = ((card A \ B) + (card A /\ B)) - (card (A \ B) /\ (A /\ B))) [union-card]
2857             ==> (card A = ((card A \ B) + (card A /\ B)) - (card null)) [diff-theorem-13]
2858             ==> (card A = ((card A \ B) + (card A /\ B)) - zero) [card-def]
2859             ==> (card A = (card A \ B) + card A /\ B) [N.Minus.axioms]])
2860
2861 conclude diff-card-theorem :=
2862   (forall A B . card A \ B = (card A) - card A /\ B)
2863 pick-any A B
2864   (!chain-> [true ==> (card A = (card A \ B) + card A /\ B) [diff-card-lemma]
2865             ==> ((card A \ B) + card A /\ B = card A) [sym]
2866             ==> (card A \ B = (card A) - card A /\ B) [N.Minus.Plus-Minus-properties]])
2867
2868 declare fun: (S, T) [(Set (Pair S T))] -> Boolean [210 [lst->set]]

```

```
2869
2870 assert* fun-def :=
2871   [(fun null)
2872     (fun x @ y ++ t = fun t <== ~ x in dom t | t ** singleton x = singleton y)
2873     (~ fun x @ y ++ t <== ~ (~ x in dom t | t ** singleton x = singleton y))]
2874
2875
2876 (eval fun [(1 @ 'a) (2 @ 'b)])
2877
2878 (eval fun [(1 @ 'a) (2 @ 'b) (1 @ 'c)])
2879
2880 (eval fun [(1 @ 'a) (2 @ 'b) (3 @ 'c) (2 @ 'd)])
2881
2882 (eval fun [(1 @ 'a) (2 @ 'b) (3 @ 'c) (8 @ 'd)])
2883
2884 (eval fun [])
2885
2886 (eval fun [(1 @ 'a)])
2887
2888 (eval fun [(1 @ 'a) (1 @ 'a)])
2889
2890 } # close Set
2891
2892 EOF
2893 ()
2894
2895 (load "sets")
```