

lib/main/set.ath

```

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

```

```

69 conclude singleton-characterization :=
70   (forall x y . x in singleton y <==> x = y)
71   pick-any x y
72   (!chain [(x in singleton y)
73     <==> (x in y ++ null) [singleton-axiom]
74     <==> (x = y | x in null) [in-def]
75     <==> (x = y | false) [null-characterization]
76     <==> (x = y) [prop-taut]])
77
78 define singleton-lemma := (forall x . x in singleton x)
79   pick-any x
80   (!chain-> [(x = x)
81     ==> (x in singleton x) [singleton-characterization]])
82
83 declare subset, proper-subset: (S) [(Set S) (Set S)] -> Boolean [[alist->set alist->set]]
84
85 assert* subset-def :=
86   [([]) subset _]
87   (h ++ t subset A <==> h in A & t subset A)]
88
89 define subset-characterization-1 :=
90   by-induction (forall A B . A subset B ==> forall x . x in A ==> x in B) {
91     null => pick-any B
92       assume (null subset B)
93       pick-any x
94       (!chain [(x in null) ==> false [NC]
95         ==> (x in B) [prop-taut]])
96   | (A as (insert h t)) =>
97     pick-any B
98     assume hyp := (A subset B)
99     pick-any x
100     let {ih := (forall B . t subset B ==>
101       forall x . x in t ==> x in B);
102       _ := (!chain-> [hyp ==> (t subset B) [subset-def]])}
103     assume hyp' := (x in A)
104     (!cases (!chain<- [(x = h | x in t) <== hyp' [in-def]])
105       assume (x = h)
106       (!chain-> [hyp ==> (h in B) [subset-def]
107         ==> (x in B) [(x = h)]]))
108     (!chain [(x in t) ==> (x in B) [ih]]))
109   }
110
111 define subset-characterization-2 :=
112   by-induction (forall A B . (forall x . x in A ==> x in B) ==> A subset B) {
113     null => pick-any B
114       assume (forall x . x in null ==> x in B)
115       (!chain-> [true ==> (null subset B) [subset-def]])
116   | (A as (insert h t)) =>
117     pick-any B
118     assume hyp := (forall x . x in A ==> x in B)
119     let {ih := (forall B . (forall x . x in t ==> x in B)
120       ==> t subset B);
121       goal := (A subset B);
122       ih-cond := pick-any x
123         (!chain [(x in t) ==> (x in A) [in-def]
124           ==> (x in B) [hyp]]);
125       _ := (!chain-> [ih-cond ==> (t subset B) [ih]])}
126     (!chain-> [(h = h)
127       ==> (h in A) [in-def]
128       ==> (h in B) [hyp]
129       ==> (h in B & t subset B) [augment]
130       ==> goal [subset-def]])
131   }
132
133 conclude subset-characterization :=
134   (forall s1 s2 . s1 subset s2 <==> forall x . x in s1 ==> x in s2)
135   pick-any s1 s2
136   (!equiv (!chain [(s1 subset s2)
137     ==> (forall x . x in s1 ==> x in s2) [subset-characterization-1]])
137     (!chain [(forall x . x in s1 ==> x in s2)
138       ==> (forall x . x in s1 ==> x in s2) [subset-characterization-2]]))

```

```

139         ==> (s1 subset s2)                                [subset-characterization-2]])
140
141 define SC := subset-characterization
142
143 define subset-intro :=
144   method (p)
145     match p {
146       (forall (some-var x) ((x in (some-term A)) ==> (x in (some-term B)))) =>
147         (!chain-> [p ==> (A subset B) [subset-characterization]])
148     }
149
150 assert* set-identity :=
151   (A = B <==> A subset B & B subset A)
152
153 conclude set-identity-characterization :=
154   (forall A B . A = B <==> forall x . x in A <==> x in B)
155   pick-any A:(Set 'S) B
156     (!equiv
157       assume hyp := (A = B)
158       pick-any x
159         let {_ := (!chain-> [hyp ==> (A subset B) [set-identity]]);
160             _ := (!chain-> [hyp ==> (B subset A) [set-identity]])}
161         (!chain [(x in A) <==> (x in B) [subset-characterization]])
162       assume hyp := (forall x . x in A <==> x in B)
163       let {A-subset-B := (!subset-intro
164         pick-any x
165           (!chain [(x in A) ==> (x in B) [hyp]]));
166           B-subset-A := (!subset-intro
167             pick-any x
168               (!chain [(x in B) ==> (x in A) [hyp]]));
169           p := (!both A-subset-B B-subset-A)}
170         (!chain-> [p ==> (A = B) [set-identity]]))
171
172 define SIC := set-identity-characterization
173
174 define set-identity-intro :=
175   method (p1 p2)
176     match [p1 p2] {
177       [(A subset B) (B subset A)] =>
178         (!chain-> [p1 ==> (p1 & p2) [augment]
179           ==> (A = B) [set-identity]])
180     }
181
182 define set-identity-intro-direct :=
183   method (premise)
184     match premise {
185       (forall x ((x in A) <==> (x in B))) =>
186         (!chain-> [premise ==> (A = B) [set-identity-characterization]])
187     }
188
189
190 assert* proper-subset-def :=
191   [(s1 proper-subset s2 <==> s1 subset s2 & s1 != s2)]
192
193 conclude neg-set-identity-characterization-1 :=
194   (forall s1 s2 . s1 != s2 <==> ~ s1 subset s2 | ~ s2 subset s1)
195   pick-any s1 s2
196     (!chain [(s1 != s2)
197       <==> (~ (s1 subset s2 & s2 subset s1)) [set-identity]
198       <==> (~ s1 subset s2 | ~ s2 subset s1) [prop-taut]])
199
200 conclude neg-set-identity-characterization-2 :=
201   (forall s1 s2 . s1 != s2 <==>
202     (exists x . x in s1 & ~ x in s2) |
203     (exists x . x in s2 & ~ x in s1))
204   pick-any s1 s2
205     (!chain [(s1 != s2)
206       <==> (~ s1 subset s2 | ~ s2 subset s1) [neg-set-identity-characterization-1]
207       <==> (~ (forall x . x in s1 ==> x in s2) | ~ (forall x . x in s2 ==> x in s1)) [SC]
208       <==> ((exists x . ~ (x in s1 ==> x in s2)) | (exists x . ~ (x in s2 ==> x in s1))) [qn]

```

```

209     <==> ((exists x . x in s1 & ~ x in s2) | (exists x . x in s2 & ~ x in s1)) [prop-taut]]
210
211 define proper-subset-characterization :=
212   (forall s1 s2 . s1 proper-subset s2 <==> s1 subset s2 & exists x . x in s2 & ~ x in s1)
213
214 conclude PSC := proper-subset-characterization
215   pick-any s1 s2
216     (!chain [(s1 proper-subset s2)
217       <==> (s1 subset s2 & s1 != s2) [proper-subset-def]
218       <==> (s1 subset s2 & ((exists x . x in s1 & ~ x in s2) |
219         (exists x . x in s2 & ~ x in s1))) [neg-set-identity-characterization-2]
220       <==> (s1 subset s2 & ((s1 subset s2) & (exists x . x in s1 & ~ x in s2)) |
221         (exists x . x in s2 & ~ x in s1))) [prop-taut]
222       <==> (s1 subset s2 & (((forall x . x in s1 ==> x in s2) & (exists x . x in s1 & ~ x in s2)) |
223         (exists x . x in s2 & ~ x in s1))) [SC]
224       <==> (s1 subset s2 & ((~ (forall x . x in s1 ==> x in s2) & (exists x . x in s1 & ~ x in s2)) |
225         (exists x . x in s2 & ~ x in s1))) [bdn]
226       <==> (s1 subset s2 & ((~ (exists x . ~ (x in s1 ==> x in s2)) & (exists x . x in s1 & ~ x in s2)) |
227         (exists x . x in s2 & ~ x in s1))) [qn]
228       <==> (s1 subset s2 & ((~ (exists x . x in s1 & ~ x in s2) & (exists x . x in s1 & ~ x in s2)) |
229         (exists x . x in s2 & ~ x in s1))) [prop-taut]
230       <==> (s1 subset s2 & (false | (exists x . x in s2 & ~ x in s1))) [prop-taut]
231       <==> (s1 subset s2 & (exists x . x in s2 & ~ x in s1)) [prop-taut]])]
232
233 conclude proper-subset-lemma :=
234   (forall A B x . A subset B & x in B & ~ x in A ==> A proper-subset B)
235   pick-any A B x
236     assume h := (A subset B & x in B & ~ x in A)
237     (!chain-> [(x in B)
238       ==> (x in B & ~ x in A) [augment]
239       ==> (exists x . x in B & ~ x in A) [existence]
240       ==> (A subset B & exists x . x in B & ~ x in A) [augment]
241       ==> (A proper-subset B) [PSC]])]
242
243 conclude in-lemma-2 := (forall h t . h in t ==> h ++ t = t)
244   pick-any h t
245     assume hyp := (h in t)
246     (!set-identity-intro-direct
247       pick-any x
248         (!chain [(x in h ++ t)
249           <==> (x = h | x in t) [in-def]
250           <==> (x in t | x in t) [hyp prop-taut]
251           <==> (x in t) [prop-taut]]))
252
253 conclude in-lemma-3 := (forall x h t . x in t ==> x in h ++ t)
254   pick-any x h t
255     (!chain [(x in t)
256       ==> (x = h | x in t) [alternate]
257       ==> (x in h ++ t) [in-def]])]
258
259
260 conclude in-lemma-4 :=
261   (forall A x y . x in A ==> y in A <==> y = x | y in A)
262   pick-any A x y
263     assume (x in A)
264     (!equiv assume h := (y in A)
265       (!chain-> [h ==> (y = x | y in A) [alternate]])
266       assume h := (y = x | y in A)
267       (!cases h
268         (!chain [(y = x) ==> (y in A) [(x in A)]]
269         (!chain [(y in A) ==> (y in A) [claim]])))
270
271 conclude null-characterization-2 :=
272   (forall A . A = null <==> forall x . ~ x in A)
273   pick-any A
274     (!chain [(A = null)
275       <==> (forall x . x in A <==> x in null) [SIC]
276       <==> (forall x . x in A <==> false) [NC]
277       <==> (forall x . ~ x in A) [prop-taut]])]
278

```

```

279 define NC-2 := null-characterization-2
280
281 conclude NC-3 :=
282   (forall A . A != null <==> exists x . x in A)
283 pick-any A
284   (!chain [(A != null)
285     <==> (~ forall x . ~ x in A) [NC-2]
286     <==> (exists x . ~ ~ x in A) [qn-strict]
287     <==> (exists x . x in A) [bdn]])
288
289 define (non-empty S) := (S != null)
290
291 conclude subset-reflexivity := (forall A . A subset A)
292 pick-any A
293   (!subset-intro
294     pick-any x
295     (!chain [(x in A) ==> (x in A) [claim]]))
296
297 conclude subset-antisymmetry :=
298   (forall A B . A subset B & B subset A ==> A = B)
299 pick-any A B
300   assume hyp := (A subset B & B subset A)
301   (!set-identity-intro (A subset B) (B subset A))
302
303 conclude subset-transitivity :=
304   (forall A B C . A subset B & B subset C ==> A subset C)
305 pick-any A B C
306   assume (A subset B & B subset C)
307   (!subset-intro
308     pick-any x
309     (!chain [(x in A)
310       ==> (x in B) [subset-characterization]
311       ==> (x in C) [subset-characterization]]))
312
313 conclude subset-lemma-1 :=
314   (forall A B x . A subset B & x in B ==> x ++ A subset B)
315 pick-any A B x
316   assume hyp := (A subset B & x in B)
317   (!subset-intro
318     pick-any y
319     (!chain [(y in x ++ A)
320       ==> (y = x | y in A) [in-def]
321       ==> (y in B | y in A) [(x in B)]
322       ==> (y in B | y in B) [SC]
323       ==> (y in B) [prop-taut]]))
324
325 conclude subset-lemma-2 :=
326   (forall h t A . h ++ t subset A ==> t subset A)
327 pick-any h t A
328   assume (h ++ t subset A)
329   (!subset-intro
330     pick-any x
331     (!chain [(x in t)
332       ==> (x = h | x in t) [alternate]
333       ==> (x in h ++ t) [in-def]
334       ==> (x in A) [SC]]))
335
336
337 declare remove: (S) [(Set S) S] -> (Set S) [- [alist->set id]]
338
339 assert* remove-def :=
340   [(null - _ = null)
341     (h ++ t - x = t - x <== x = h)
342     (h ++ t - x = h ++ (t - x) <== x != h)]
343
344 conclude remove-characterization :=
345   (forall A x y . y in A - x <==> y in A & y != x)
346 by-induction remove-characterization {
347   null => pick-any x y
348     (!chain [(y in null - x)

```

```

349         <==> (y in null)
350         <==> false
351         <==> (y in null & y /= x)])
352 | (A as (insert h t)) =>
353   let {IH := (forall x y . y in t - x <==> y in t & y /= x)}
354   pick-any x y
355     (!two-cases
356       assume case-1 := (x = h)
357       (!chain [(y in A - x)
358         <==> (y in t - x) [remove-def]
359         <==> (y in t & y /= x) [IH]
360         <==> ((y = x | y in t) & y /= x) [prop-taut]
361         <==> ((y = h | y in t) & y /= x) [case-1]
362         <==> (y in A & y /= x) [in-def]])
363       assume case-2 := (x /= h)
364       let {lemma := assume (y = h)
365         (!chain-> [case-2 ==> (y /= x) [(y = h)])])
366       (!chain [(y in A - x)
367         <==> (y in h ++ (t - x)) [remove-def]
368         <==> (y = h | y in t - x) [in-def]
369         <==> (y = h | (y in t & y /= x)) [IH]
370         <==> ((y = h | y in t) & (y = h | y /= x)) [prop-taut]
371         <==> (y in A & (y = h | y /= x)) [in-def]
372         <==> (y in A & (y /= x | y /= x)) [prop-taut lemma]
373         <==> (y in A & y /= x) [prop-taut]])])
374 }
375
376 conclude remove-corollary := (forall A x . ~ x in A - x)
377 pick-any A x
378   (!by-contradiction (~ x in A - x)
379     (!chain [(x in A - x)
380       ==> (x in A & x /= x) [remove-characterization]
381       ==> (x /= x) [right-and]
382       ==> (x = x & x /= x) [augment]
383       ==> false [prop-taut]]))
384
385 conclude remove-corollary-2 :=
386   (forall A x . ~ x in A ==> A - x = A)
387 pick-any A x
388   assume hyp := (~ x in A)
389   (!set-identity-intro-direct
390     pick-any y
391       (!equiv
392         (!chain [(y in A - x)
393           ==> (y in A & y /= x) [remove-characterization]
394           ==> (y in A) [left-and]])
395         assume (y in A)
396           let {_ := (!by-contradiction (y /= x)
397             assume (y = x)
398               (!absurd (y in A)
399                 (!chain-> [hyp ==> (~ y in A) [(y = x)])])
400             lemma := (!both (y in A) (y /= x)))
401             (!chain-> [lemma ==> (y in A - x) ]))
402
403 conclude remove-corollary-3 :=
404   (forall A x y . x in A & y /= x ==> x in A - y)
405 pick-any A x y
406   assume hyp := (x in A & y /= x)
407   let {_ := (!ineq-sym (y /= x))}
408   (!chain-> [hyp ==> (x in A - y) [remove-characterization]])
409
410 conclude remove-corollary-4 :=
411   (forall A x y . ~ x in A ==> ~ x in A - y)
412 pick-any A x y
413   (!chain [(~ x in A) ==> (~ x in A - y) [remove-characterization]])
414
415 conclude remove-corollary-5 :=
416   (forall A B x . A subset B & ~ x in A ==> A subset B - x)
417 pick-any A B x
418   assume h := (A subset B & ~ x in A)

```

```

419 (!subset-intro
420   pick-any y
421   assume h2 := (in y A)
422   let { _ := (!chain-> [h2 ==> (in y B) [SC]]);
423       _ := (!by-contradiction (y /= x)
424         assume (y = x)
425           (!absurd (in y A)
426             (!chain-> [(~ x in A) ==> (~ y in A) [(y = x)]])));
427       S := (!both (in y B) (y /= x))}
428   (!chain-> [S ==> (y in B - x) [remove-characterization]]))
429
430 conclude remove-corollary-6 := (forall A h t . A subset h ++ t ==> A - h subset t)
431 pick-any A:(Set 'S) h:'S t:(Set 'S)
432 assume hyp := (A subset h ++ t)
433 (!subset-intro
434   pick-any x
435   assume hyp' := (x in A - h)
436   let { _ := (!chain-> [hyp' ==> (x in A & x /= h) [remove-characterization]]);
437       disj := (!chain-> [(x in A) ==> (x in h ++ t) [SC]
438         ==> (x = h | x in t) [in-def]])}
439   (!cases disj
440     (!chain [(x = h)
441       ==> (x = h & x /= h) [augment]
442       ==> false [prop-taut]
443       ==> (x in t) [prop-taut]])
444     (!chain [(x in t) ==> (x in t) [claim]])))
445
446 conclude remove-corollary-7 := (forall A x . A - x subset A)
447 pick-any A:(Set 'S) x:'S
448 (!subset-intro
449   pick-any y
450   (!chain [(y in A - x)
451     ==> (y in A) [remove-characterization]]))
452
453 conclude remove-corollary-8 :=
454   (forall A x . x in A ==> A = x ++ (A - x))
455 pick-any A:(Set 'S) x:'S
456 assume (x in A)
457 let {p1 := (!subset-intro
458   pick-any y:'S
459   assume (y in A)
460   (!two-cases
461     assume (x = y)
462     (!chain-> [true ==> (x in x ++ (A - x)) [in-lemma-1]
463       ==> (y in x ++ (A - x)) [(x = y)]])
464     assume (x /= y)
465     (!chain-> [(x /= y)
466       ==> (y in A & x /= y) [augment]
467       ==> (y in A - x) [remove-corollary-3]
468       ==> (y in x ++ (A - x)) [in-def]]))};
469   p2 := (!subset-intro
470     pick-any y:'S
471     assume hyp := (y in x ++ (A - x))
472     (!cases (!chain<- [(y = x | y in A - x) <== hyp [in-def]]
473       assume (y = x)
474       (!chain-> [(y = x) ==> (y in A) [(x in A)]]
475       assume (y in A - x)
476       (!chain-> [(y in A - x) ==> (y in A) [remove-characterization]])))
477     (!set-identity-intro p1 p2))
478
479 conclude subset-lemma-3 :=
480   (forall A t h . A subset h ++ t & h in A ==> exists B . B subset t & A = h ++ B)
481 pick-any A:(Set 'S) t h:'S
482 assume hyp := (A subset h ++ t & h in A)
483 let {p := (!chain-> [(A subset h ++ t) ==> (A - h subset t) [remove-corollary-6]]);
484   (!chain-> [(h in A)
485     ==> (A = h ++ (A - h)) [remove-corollary-8]
486     ==> (p & A = h ++ (A - h)) [augment]
487     ==> (exists B . B subset t & A = h ++ B) [existence]})}
488

```

```

489 conclude subset-lemma-4 :=
490   (forall A h t . ~ h in A & A subset h ++ t ==> A subset t)
491 pick-any A h t
492   assume hyp := (~ h in A & A subset h ++ t)
493   (!subset-intro
494     pick-any x
495     assume (x in A)
496       (!chain (!chain<- [(x = h | x in t) <== (x in h ++ t) [in-def]
497         <== (x in A) [SC]]))
498       (!chain [(x = h)
499         ==> (~ x in A) [(~ h in A)]
500         ==> (x in A & ~ x in A) [augment]
501         ==> (in x t) [prop-taut]])
502       (!chain [(x in t) ==> (x in t) [claim]])))
503
504 conclude subset-lemma-5 :=
505   (forall A t h . A subset t ==> A subset h ++ t)
506 pick-any A t h
507   assume hyp := (A subset t)
508   (!subset-intro
509     pick-any x
510     (!chain [(x in A) ==> (x in t) [SC]
511       ==> (x in h ++ t) [in-def]]))
512
513 conclude subset-lemma-6 :=
514   (forall A . A subset null <==> A = null)
515 pick-any A
516   (!equiv assume (A subset null)
517     (!by-contradiction (A = null)
518       assume (A /= null)
519       pick-witness x for (!chain<- [(exists x . x in A) <== (A /= null) [NC-3]])
520       (!chain-> [(x in A) ==> (x in null) [SC]
521         ==> false [NC]]))
522     assume (A = null)
523     (!chain-> [true ==> (A subset A) [subset-reflexivity]
524       ==> (A subset null) [(A = null)]])
525
526 conclude subset-lemma-7 :=
527   (forall A B x . ~ x in A & B subset A ==> ~ x in B)
528 pick-any A B x
529   assume hyp := (~ x in A & B subset A)
530   (!by-contradiction (~ x in B)
531     (!chain [(x in B) ==> (x in A) [SC]
532       ==> (x in A & ~ x in A) [augment]
533       ==> false [prop-taut]]))
534
535
536 declare union, intersection, diff: (S) [(Set S) (Set S)] -> (Set S) [120 [alist->set alist->set]]
537
538 define [/ \ / \ ] := [union intersection diff]
539
540 assert* union-def :=
541   [([ / \ / s = s)
542     (h ++ t \ / s = h ++ (t \ / s))]
543
544 transform-output eval [set->alist]
545
546 conclude union-characterization-1 :=
547   (forall A B x . x in A \ / B ==> x in A | x in B)
548 by-induction union-characterization-1 {
549   null => pick-any B x
550     (!chain [(x in null \ / B)
551       ==> (x in B) [union-def]
552       ==> (x in null | x in B) [alternate]])
553   | (A as (h insert t)) =>
554     let {IH := (forall B x . x in t \ / B ==> x in t | x in B)}
555     pick-any B x
556     (!chain [(x in A \ / B)
557       ==> (x in h ++ (t \ / B)) [union-def]
558       ==> (x = h | x in t \ / B) [in-def]

```



```

559         ==> (x = h | x in t | x in B) [IH]
560         ==> ((x = h | x in t) | x in B) [prop-taut]
561         ==> (x in A | x in B) [in-def]]
562     }
563
564 conclude union-characterization-2 :=
565   (forall A B x . x in A | x in B ==> x in A \ / B)
566   by-induction union-characterization-2 {
567     (A as null) =>
568       pick-any B x
569       (!chain [(x in null | x in B)
570         ==> (false | x in B) [NC]
571         ==> (x in B) [prop-taut]
572         ==> (x in null \ / B) [union-def]])
573
574     | (A as (insert h t)) =>
575       pick-any B x
576       let {IH := (forall B x . x in t | x in B ==> x in t \ / B)}
577       (!chain [(x in A | x in B)
578         ==> ((x = h | x in t) | x in B) [in-def]
579         ==> (x = h | (x in t | x in B)) [prop-taut]
580         ==> (x = h | x in t \ / B) [IH]
581         ==> (x in h ++ (t \ / B)) [in-def]
582         ==> (x in A \ / B) [union-def]])
583   }
584
585 conclude union-characterization :=
586   (forall A B x . x in A \ / B <==> x in A | x in B)
587   pick-any A B x
588   (!chain [(x in A \ / B)
589     <==> (x in A | x in B) [union-characterization-1
590       union-characterization-2]])
591
592 define UC := union-characterization
593
594 assert* intersection-def :=
595   [(null \ / s = null)
596     (h ++ t \ / A = h ++ (t \ / A) <== h in A)
597     (h ++ t \ / A = t \ / A <== ~ h in A)]
598
599 conclude intersection-characterization-1 :=
600   (forall A B x . x in A \ / B ==> x in A & x in B)
601   by-induction intersection-characterization-1 {
602     null => pick-any B x
603       (!chain [(x in null \ / B)
604         ==> (x in null) [intersection-def]
605         ==> false [NC]
606         ==> (x in null & x in B) [prop-taut]])
607     | (A as (insert h t)) =>
608       let {IH := (forall B x . x in t \ / B ==> x in t & x in B)}
609       pick-any B x
610       (!two-cases
611         assume (h in B)
612         (!chain [(x in (h ++ t) \ / B)
613           ==> (x in h ++ (t \ / B)) [intersection-def]
614           ==> (x = h | x in t \ / B) [in-def]
615           ==> (x = h | x in t & x in B) [IH]
616           ==> ((x = h | x in t) & (x = h | x in B)) [prop-taut]
617           ==> (x in A & (x in B | x in B)) [in-def (h in B)]
618           ==> (x in A & x in B) [prop-taut]
619         ])
620         assume (~ h in B)
621         (!chain [(x in A \ / B)
622           ==> (x in t \ / B) [intersection-def]
623           ==> (x in t & x in B) [IH]
624           ==> (x in A & x in B) [in-def]]))
625   }
626
627 conclude intersection-characterization-2 :=
628   (forall A B x . x in A & x in B ==> x in A \ / B)

```

```

629 by-induction intersection-characterization-2 {
630   (A as null) =>
631   pick-any B x
632     (!chain [(x in null & x in B)
633               ==> (x in null)           [left-and]
634               ==> false                 [NC]
635               ==> (x in null /\ B)      [prop-taut]])
636 | (A as (insert h t)) =>
637   let {IH := (forall B x . x in t & x in B ==> x in t /\ B)}
638   pick-any B x
639     (!two-cases
640       assume (h in B)
641         (!chain [(x in A & x in B)
642                   ==> ((x = h | x in t) & x in B)           [in-def]
643                   ==> ((x = h & x in B) | (x in t & x in B)) [prop-taut]
644                   ==> (x = h | x in t & x in B)             [prop-taut]
645                   ==> (x = h | x in t /\ B)                 [IH]
646                   ==> (x in h ++ (t /\ B))                 [in-def]
647                   ==> (x in A /\ B)                         [intersection-def]))
648       assume case2 := (~ h in B)
649         (!chain [(x in A & x in B)
650                   ==> ((x = h | x in t) & x in B)           [in-def]
651                   ==> ((~ x in B | x in t) & x in B)         [case2]
652                   ==> ((~ x in B & x in B) | (x in t & x in B)) [prop-taut]
653                   ==> (false | x in t & x in B)             [prop-taut]
654                   ==> (x in t & x in B)                     [prop-taut]
655                   ==> (x in t /\ B)                         [IH]
656                   ==> (x in A /\ B)                         [intersection-def]))
657 }
658
659 conclude intersection-characterization :=
660   (forall A B x . x in A /\ B <==> x in A & x in B)
661   pick-any A B x
662     (!equiv
663       (!chain [(x in A /\ B)
664                 ==> (x in A & x in B) [intersection-characterization-1]))
665       (!chain [(x in A & x in B)
666                 ==> (x in A /\ B)    [intersection-characterization-2]))
667     )
668 define IC := intersection-characterization
669
670 conclude intersection-subset-theorem :=
671   (forall A B . A /\ B subset A)
672   pick-any A B
673     (!subset-intro
674       pick-any x
675         (!chain [(x in A /\ B)
676                   ==> (x in A)      [IC]]))
677     )
678 assert* diff-def :=
679   [(null \ _ = null)
680    (h ++ t \ A = t \ A <== h in A)
681    (h ++ t \ A = h ++ (t \ A) <== ~ h in A)]
682
683 conclude diff-characterization-1 :=
684   (forall A B x . x in A \ B ==> x in A & ~ x in B)
685   by-induction diff-characterization-1 {
686     (A as null) =>
687     pick-any B x
688       (!chain [(x in A \ B)
689                 ==> (x in null)           [diff-def]
690                 ==> false                 [null-characterization]
691                 ==> (x in null & ~ x in B) [prop-taut]])
692 | (A as (insert h t)) =>
693   pick-any B x
694     let {ih := (forall B x . x in t \ B ==> x in t & ~ x in B)}
695     assume hyp := (x in A \ B)
696     (!two-cases
697       assume case1 := (h in B)
698       (!chain-> [hyp

```

```

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

```

```

769
770
771 conclude intersection-commutes := (forall A B . A /\ B = B /\ A)
772 pick-any A B
773   let {A-subset-of-B :=
774     (!subset-intro
775       pick-any x
776       (!chain [(x in A /\ B)
777         ==> (x in A & x in B) [IC]
778         ==> (x in B & x in A) [prop-taut]
779         ==> (x in B /\ A) [IC]]));
780     B-subset-of-A :=
781     (!subset-intro
782       pick-any x
783       (!chain [(x in B /\ A)
784         ==> (x in B & x in A) [IC]
785         ==> (x in A & x in B) [prop-taut]
786         ==> (x in A /\ B) [IC]]))
787   }
788   (!set-identity-intro A-subset-of-B B-subset-of-A)
789
790 conclude intersection-commutes := (forall A B . A /\ B = B /\ A)
791 let {M := method (A B) # derive (A /\ B subset B /\ A)
792   (!subset-intro
793     pick-any x
794     (!chain [(x in A /\ B)
795       ==> (x in A & x in B) [IC]
796       ==> (x in B & x in A) [prop-taut]
797       ==> (x in B /\ A) [IC]]))
798   pick-any A B
799   (!set-identity-intro (!M A B) (!M B A))
800
801 conclude intersection-subset-theorem-2 :=
802   (forall A B . A /\ B subset B)
803 pick-any A B
804   (!chain-> [true ==> (B /\ A subset B) [intersection-subset-theorem]
805     ==> (A /\ B subset B) [intersection-commutes]])
806
807 conclude intersection-subset-theorem' :=
808   (forall A B C . A subset B /\ C <==> A subset B & A subset C)
809 pick-any A B C
810   (!equiv assume (A subset B /\ C)
811     (!both (!subset-intro
812       pick-any x
813       (!chain [(x in A) ==> (x in B /\ C) [SC]
814         ==> (x in B) [IC]]))
815       (!subset-intro
816         pick-any x
817         (!chain [(x in A) ==> (x in B /\ C) [SC]
818           ==> (x in C) [IC]])))
819     assume (A subset B & A subset C)
820     (!subset-intro
821       pick-any x
822       assume (x in A)
823       let {_ := (!chain-> [(x in A) ==> (x in B) [SC]]);
824         _ := (!chain-> [(x in A) ==> (x in C) [SC]]);
825         p := (!both (x in B) (x in C))}
826       (!chain-> [p ==> (x in B /\ C) [IC]]))
827
828 conclude union-subset-theorem :=
829   (forall A B C . A subset B | A subset C ==> A subset B \/ C)
830 pick-any A B C
831   assume hyp := (A subset B | A subset C)
832   (!cases hyp
833     assume (A subset B)
834     (!subset-intro
835       pick-any x
836       (!chain [(x in A) ==> (x in B) [SC]
837         ==> (x in B | x in C) [alternate]
838         ==> (x in B \/ C) [UC]]))

```

```

839     assume (A subset C)
840     (!subset-intro
841       pick-any x
842       (!chain [(x in A) ==> (x in C)           [SC]
843                ==> (x in B | x in C) [alternate]
844                ==> (x in B \ / C)      [UC]])))
845
846 conclude union-commutes := (forall A B . A \ / B = B \ / A)
847 pick-any A B
848 (!set-identity-intro-direct
849   pick-any x
850   (!chain [(x in A \ / B) <==> (x in A | x in B) [UC]
851            <==> (x in B | x in A) [prop-taut]
852            <==> (x in B \ / A)   [UC]])))
853
854 conclude intersection-associativity :=
855   (forall A B C . A /\ (B /\ C) = (A /\ B) /\ C)
856 pick-any A B C
857 (!set-identity-intro-direct
858   pick-any x
859   (!chain [(x in A /\ B /\ C)
860            <==> (x in A & x in B /\ C) [IC]
861            <==> (x in A & x in B & x in C) [IC]
862            <==> ((x in A & x in B) & x in C) [prop-taut]
863            <==> ((x in A /\ B) & x in C) [IC]
864            <==> (x in (A /\ B) /\ C) [IC]])))
865
866 conclude union-associativity :=
867   (forall A B C . A \ / B \ / C = (A \ / B) \ / C)
868 pick-any A B C
869 (!set-identity-intro-direct
870   pick-any x
871   (!chain [(x in A \ / B \ / C)
872            <==> (x in A | x in B \ / C) [UC]
873            <==> (x in A | x in B | x in C) [UC]
874            <==> ((x in A | x in B) | x in C) [prop-taut]
875            <==> (x in A \ / B | x in C) [UC]
876            <==> (x in (A \ / B) \ / C) [UC]])))
877
878 conclude /\-idempotence :=
879   (forall A . A /\ A = A)
880 pick-any A
881 (!set-identity-intro-direct
882   pick-any x
883   (!chain [(x in A /\ A)
884            <==> (x in A & x in A) [IC]
885            <==> (x in A) [prop-taut]])))
886
887 conclude \/-idempotence :=
888   (forall A . A \ / A = A)
889 pick-any A
890 (!set-identity-intro-direct
891   pick-any x
892   (!chain [(x in A \ / A)
893            <==> (x in A | x in A) [UC]
894            <==> (x in A) [prop-taut]])))
895
896 conclude union-null-theorem :=
897   (forall A B . A \ / B = null <==> A = null & B = null)
898 pick-any A B
899 (!chain [(A \ / B = null)
900          <==> (forall x . x in A \ / B <==> x in null) [SIC]
901          <==> (forall x . x in A \ / B <==> false) [NC]
902          <==> (forall x . x in A | x in B <==> false) [UC]
903          <==> (forall x . ~ x in A & ~ x in B) [prop-taut]
904          <==> ((forall x . ~ x in A) & (forall x ~ x in B)) [taut]
905          <==> (A = null & B = null) [NC-2]])
906
907
908 conclude distributivity-1 :=

```

```

909 (forall A B C . A /\ (B /\ C) = (A /\ B) /\ (A /\ C))
910 pick-any A B C
911 (!set-identity-intro-direct
912   pick-any x
913   (!chain [(x in A /\ (B /\ C))
914     <==> (x in A | x in B /\ C) [UC]
915     <==> (x in A | x in B & x in C) [IC]
916     <==> ((x in A | x in B) & (x in A | x in C)) [prop-taut]
917     <==> (x in A /\ B & x in A /\ C) [UC]
918     <==> (x in (A /\ B) /\ (A /\ C)) [IC]]))
919
920
921 conclude distributivity-2 :=
922 (forall A B C . A /\ (B /\ C) = (A /\ B) /\ (A /\ 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)) [UC]
929     <==> ((x in A & x in B) | (x in A & x in C)) [prop-taut]
930     <==> (x in A /\ B | x in A /\ C) [IC]
931     <==> (x in (A /\ B) /\ (A /\ C)) [UC]]))
932
933 conclude diff-theorem-1 := (forall A . A \ A = null)
934 pick-any A
935 (!set-identity-intro-direct
936   pick-any x
937   (!chain [(x in A \ A)
938     <==> (x in A & ~ x in A) [DC]
939     <==> false [prop-taut]
940     <==> (x in null) [NC]]))
941
942 conclude diff-theorem-2 :=
943 (forall A B C . B subset C ==> A \ C subset A \ B)
944 pick-any A B C
945 assume (B subset C)
946 (!subset-intro
947   pick-any x
948   (!chain [(x in A \ C)
949     ==> (x in A & ~ x in C) [DC]
950     ==> (x in A & ~ x in B) [SC]
951     ==> (x in A \ B) [DC]]))
952
953
954 define p := (forall A B C . B subset C ==> A \ B subset A \ C)
955
956  #(falsify p 20) 
957
958 conclude diff-theorem-3 :=
959 (forall A B . A \ (A /\ B) = A \ B)
960 pick-any A B
961 (!set-identity-intro-direct
962   pick-any x
963   (!chain [(x in A \ (A /\ B))
964     <==> (x in A & ~ x in A /\ B) [DC]
965     <==> (x in A & ~ (x in A & x in B)) [IC]
966     <==> (x in A & (~ x in A | ~ x in B)) [prop-taut]
967     <==> ((x in A & ~ x in A) | (x in A & ~ x in B)) [prop-taut]
968     <==> (false | x in A & ~ x in B) [prop-taut]
969     <==> (x in A & ~ x in B) [prop-taut]
970     <==> (x in A \ B) [DC]]))
971
972 conclude diff-theorem-4 :=
973 (forall A B . A /\ (A \ B) = A \ B)
974 pick-any A B
975 (!set-identity-intro-direct
976   pick-any x
977   (!chain [(x in A /\ (A \ B))
978     <==> (x in A & x in A \ B) [IC]

```

```

979         <==> (x in A & x in A & ~ x in B) [DC]
980         <==> (x in A & ~ x in B) [prop-taut]
981         <==> (x in A \ B) [DC]]))
982
983 conclude diff-theorem-5 :=
984   (forall A B . (A \ B) \ / B = A \ / B)
985   pick-any A B
986   (!set-identity-intro-direct
987     pick-any x
988     (!chain [(x in (A \ B) \ / B)
989       <==> (x in A \ B | x in B) [UC]
990       <==> ((x in A & ~ x in B) | x in B) [DC]
991       <==> ((x in A | x in B) & (~ x in B | x in B)) [prop-taut]
992       <==> ((x in A | x in B) & true) [prop-taut]
993       <==> (x in A | x in B) [prop-taut]
994       <==> (x in A \ / B) [UC]]))
995
996 conclude diff-theorem-6 :=
997   (forall A B . (A \ / B) \ B = A \ B)
998   pick-any A B
999   (!set-identity-intro-direct
1000     pick-any x
1001     (!chain [(x in (A \ / B) \ B)
1002       <==> (x in A \ / B & ~ x in B) [DC]
1003       <==> ((x in A | x in B) & ~ x in B) [UC]
1004       <==> (x in A & ~ x in B | x in B & ~ x in B) [prop-taut]
1005       <==> (x in A & ~ x in B | false) [prop-taut]
1006       <==> (x in A \ B | false) [DC]
1007       <==> (x in A \ B) [prop-taut]]))
1008
1009 conclude diff-theorem-7 :=
1010   (forall A B . (A /\ B) \ B = null)
1011   pick-any A B
1012   (!set-identity-intro-direct
1013     pick-any x
1014     (!chain [(x in (A /\ B) \ B)
1015       <==> (x in A /\ B & ~ x in B) [DC]
1016       <==> ((x in A & x in B) & ~ x in B) [IC]
1017       <==> false [prop-taut]
1018       <==> (x in null) [NC]]))
1019
1020 conclude diff-theorem-8 :=
1021   (forall A B . (A \ B) /\ B = null)
1022   pick-any A B
1023   (!set-identity-intro-direct
1024     pick-any x
1025     (!chain [(x in (A \ B) /\ B)
1026       <==> (x in A \ B & x in B) [IC]
1027       <==> ((x in A & ~ x in B) & x in B) [DC]
1028       <==> false [prop-taut]
1029       <==> (x in null) [NC]]))
1030
1031 conclude diff-theorem-8 :=
1032   (forall A B C . A \ (B \ / C) = (A \ B) \ / (A \ C))
1033   pick-any A B C
1034   (!set-identity-intro-direct
1035     pick-any x
1036     (!chain [(x in A \ (B \ / C))
1037       <==> (x in A & ~ x in B \ / C) [DC]
1038       <==> (x in A & ~ (x in B | x in C)) [UC]
1039       <==> (x in A & ~ x in B & ~ x in C) [prop-taut]
1040       <==> ((x in A & ~ x in B) & (x in A & ~ x in C)) [prop-taut]
1041       <==> (x in A \ B & x in A \ C) [DC]
1042       <==> (x in (A \ B) \ / (A \ C)) [IC]]))
1043
1044 conclude diff-theorem-9 :=
1045   (forall A B C . A \ (B /\ C) = (A \ B) \ / (A \ C))
1046   pick-any A B C
1047   (!set-identity-intro-direct
1048     pick-any x

```

```

1049      (!chain [(x in A \ (B /\ C))
1050        <==> (x in A & ~ x in B /\ C) [DC]
1051        <==> (x in A & ~ (x in B & x in C)) [IC]
1052        <==> (x in A & (~ x in B | ~ x in C)) [prop-taut]
1053        <==> ((x in A & ~ x in B) | (x in A & ~ x in C)) [prop-taut]
1054        <==> (x in A \ B | x in A \ C) [DC]
1055        <==> (x in (A \ B) /\ (A \ C)) [UC]]))
1056
1057 conclude diff-theorem-10 := (forall A B . A \ (A \ B) = A /\ B)
1058 pick-any A B
1059   (!set-identity-intro-direct
1060     pick-any x
1061     (!chain [(x in A \ (A \ B))
1062       <==> (x in A & ~ x in A \ B) [DC]
1063       <==> (x in A & ~ (x in A & ~ x in B)) [DC]
1064       <==> (x in A & (~ x in A | ~ ~ x in B)) [prop-taut]
1065       <==> ((x in A & ~ x in A) | (x in A & x in B)) [prop-taut]
1066       <==> (false | x in A & x in B) [prop-taut]
1067       <==> (x in A & x in B) [prop-taut]
1068       <==> (x in A /\ B) [IC]]))
1069
1070 conclude diff-theorem-11 := (forall A B . A subset B ==> A /\ (B \ A) = B)
1071 pick-any A B
1072   assume hyp := (A subset B)
1073   (!set-identity-intro-direct
1074     pick-any x
1075     (!chain
1076       [(x in A /\ (B \ A))
1077       <==> (x in A | x in B \ A) [UC]
1078       <==> (x in A | x in B & ~ x in A) [DC]
1079       <==> ((x in A | x in B) & (x in A | ~ x in A)) [prop-taut]
1080       <==> (x in A | x in B) [prop-taut]
1081       <==> (x in B | x in B) [SC prop-taut]
1082       <==> (x in B) [prop-taut]]))
1083
1084
1085 conclude diff-theorem-12 :=
1086   (forall A B . A = (A \ B) /\ (A /\ B))
1087 pick-any A B
1088   (!comm
1089     (!set-identity-intro-direct
1090       pick-any x
1091       (!chain [(x in (A \ B) /\ (A /\ B))
1092         <==> (x in A \ B | x in A /\ B) [UC]
1093         <==> (x in A & ~ x in B | x in A & x in B) [DC IC]
1094         <==> (x in A) [prop-taut]])))
1095
1096 conclude diff-theorem-13 :=
1097   (forall A B . (A \ B) /\ (A /\ B) = null)
1098 pick-any A B
1099   (!set-identity-intro-direct
1100     pick-any x
1101     (!chain [(x in (A \ B) /\ (A /\ B))
1102       <==> (x in (A \ B) & x in A /\ B) [IC]
1103       <==> ((x in A & ~ x in B) & (x in A & x in B)) [DC IC]
1104       <==> false [prop-taut]
1105       <==> (x in null) [NC]]))
1106
1107
1108 #define diff-remove-theorem := (forall A x . A - x = A \ singleton x)
1109 #(mark 'A)
1110 # START_LOAD
1111 # datatype-cases diff-remove-theorem {
1112 # null => pick-any x
1113 #
1114   (!set-identity-intro-direct
1115     pick-any y
1116     (!chain [(y in null - x)
1117       <==> (y in null)
1118       <==> false
1119       <==> (y in null & ~ y in singleton x)

```



```

1119 #                                     <==> (y in null \ singleton x)))
1120 # | (A as (insert h t)) =>
1121 #   pick-any x
1122 #     (!set-identity-intro
1123 #       (!subset-intro
1124 #         pick-any y
1125 #           assume hyp := (y in A - x)
1126 #             (!two-cases
1127 #               assume case-1 := (x = h)
1128 #                 let {y/=x := (!chain [(y in A - x)
1129 #                                       ==> (y in t - x)
1130 #                                       ==> (y in t \ singleton x)
1131 #                                       ==> (y in t & ~ y in singleton x)
1132 #                                       ==> (y /= x)]])
1133 #             }
1134 #END_LOAD
1135
1136 #(!induction* diff-remove-theorem)
1137
1138 conclude absorption-1 :=
1139   (forall x A . x in A <==> x ++ A = A)
1140   pick-any x A
1141     (!equiv
1142       assume hyp := (x in A)
1143         (!set-identity-intro-direct
1144           pick-any y
1145             (!chain [(y in x ++ A)
1146                     <==> (y = x | y in A)      [in-def]
1147                     <==> (y in A | y in A)      [hyp prop-taut]
1148                     <==> (y in A)               [prop-taut]]))
1149           assume (x ++ A = A)
1150             (!chain-> [true ==> (x in x ++ A) [in-lemma-1]
1151                      ==> (x in A)           [set-identity-characterization]]))
1152
1153 conclude subset-theorem-1 :=
1154   (forall A B . A subset B ==> A /\ B = B)
1155   pick-any A B
1156     assume (A subset B)
1157       (!set-identity-intro-direct
1158         pick-any x
1159           (!chain [(x in A /\ B)
1160                   <==> (x in A | x in B) [UC]
1161                   <==> (x in B | x in B) [prop-taut SC]
1162                   <==> (x in B)         [prop-taut]]))
1163
1164
1165 conclude subset-theorem-2 :=
1166   (forall A B . A subset B ==> A /\ B = A)
1167   pick-any A B
1168     assume (A subset B)
1169       (!set-identity-intro-direct
1170         pick-any x
1171           (!chain [(x in A /\ B)
1172                   <==> (x in A & x in B) [IC]
1173                   <==> (x in A & x in A) [prop-taut SC]
1174                   <==> (x in A)         [prop-taut]]))
1175
1176
1177 conclude intersection-lemma-1 :=
1178   (forall A B x . x in B & x in A ==> A /\ B = (x ++ A) /\ B)
1179   pick-any A B x
1180     assume hyp := (x in B & x in A)
1181       (!set-identity-intro-direct
1182         pick-any y
1183           (!chain [(y in A /\ B)
1184                   <==> (y in A & y in B) [IC]
1185                   <==> ((y = x | y in A) & y in B) [(y in A <==> y = x | y in A) <== (x in A) [in-lemma-4]]
1186                   <==> ((y in x ++ A) & y in B) [in-def]
1187                   <==> (y in (x ++ A) /\ B) [IC]]))
1188

```

```

1189 conclude intersection-lemma-2 :=
1190   (forall A B x .  $\sim x$  in A ==>  $\sim x$  in A /\ B)
1191 pick-any A B x
1192   assume hyp := ( $\sim x$  in A)
1193   (!by-contradiction ( $\sim x$  in A /\ B)
1194     (!chain [(x in A /\ B)
1195       ==> (x in A) [IC]
1196       ==> (x in A &  $\sim x$  in A) [augment]
1197       ==> false [prop-taut]]))
1198
1199
1200 conclude intersection-lemma-3 :=
1201   (forall A . A /\ A = A)
1202 pick-any A
1203   (!set-identity-intro-direct
1204     pick-any x
1205     (!chain [(x in A /\ A)
1206       <==> (x in A & x in A) [IC]
1207       <==> (x in A) [prop-taut]]))
1208
1209 declare insert-in-all: (S) [S (Set (Set S))] -> (Set (Set S)) [[id alist->set]]
1210
1211 assert* insert-in-all-def :=
1212   [(x insert-in-all null = null)
1213     (x insert-in-all A ++ t = (x ++ A) ++ (x insert-in-all t))]
1214
1215 define in-all := insert-in-all
1216
1217 conclude insert-in-all-characterization :=
1218   (forall U s x . s in x in-all U <==> exists B . B in U & s = x ++ B)
1219 by-induction insert-in-all-characterization {
1220   (U as null) => pick-any s x
1221     (!equiv (!chain [(s in x in-all U)
1222       ==> (s in null) [insert-in-all-def]
1223       ==> false [NC]
1224       ==> (exists B . B in U & s = x ++ B) [prop-taut]]
1225       assume hyp := (exists B . B in U & s = x ++ B)
1226       pick-witness B for hyp
1227       (!chain-> [(B in U)
1228         ==> false [NC]
1229         ==> (s in x in-all U) [prop-taut]]))
1230 | (U as (insert A more)) =>
1231   let {IH := (forall s x . s in x in-all more <==> exists B . B in more & s = x ++ B)}
1232   pick-any s x
1233   let {
1234     G := (exists B . B in U & s = x ++ B);
1235     L := conclude ((s = x ++ A | exists B . B in more & s = x ++ B) <==> G)
1236     (!equiv
1237       assume hyp := (s = x ++ A | exists B . B in more & s = x ++ B)
1238       (!cases hyp
1239         assume (s = x ++ A)
1240         (!chain-> [true ==> (A in U) [in-lemma-1]
1241           ==> (s = x ++ A & A in U) [augment]
1242           ==> (A in U & s = x ++ A) [prop-taut]
1243           ==> G [existence]]))
1244         (!chain [(exists B . B in more & s = x ++ B)
1245           ==> (exists B . B in U & s = x ++ B) [in-def]]))
1246       assume hyp := (exists B . B in U & s = x ++ B)
1247       let {goal := (s = x ++ A | exists B . B in more & s = x ++ B)}
1248       pick-witness B for hyp
1249       (!cases (!chain<- [(B = A | B in more) <== (B in U) [in-def]]
1250         assume (B = A)
1251         (!chain-> [(s = x ++ B) ==> (s = x ++ A) [(B = A)]
1252           ==> goal [alternate]]))
1253         assume (B in more)
1254         (!chain-> [(B in more)
1255           ==> (B in more & s = x ++ B) [augment]
1256           ==> (exists B . B in more & s = x ++ B) [existence]
1257           ==> goal [alternate]]))
1258   }

```

```

1259     (!chain [(s in x in-all U)
1260       <==> (s in (x ++ A) ++ (x in-all more)) [insert-in-all-def]
1261       <==> (s = x ++ A | s in x in-all more) [in-def]
1262       <==> (s = x ++ A | exists B . B in more & s = x ++ B) [IH]
1263       <==> G [L]
1264     ])
1265   }
1266
1267   declare powerset: (S) [(Set S)] -> (Set (Set S)) [[alist->set]]
1268
1269   assert* powerset-def :=
1270     [(powerset null = singleton null)
1271     (powerset x ++ t = (powerset t) \\/ (x insert-in-all (powerset t)))]
1272
1273   conclude powerset-characterization :=
1274     (forall A B . B in powerset A <==> B subset A)
1275   by-induction powerset-characterization {
1276     (A as Set.null) =>
1277       pick-any B
1278       (!chain [(B in powerset A)
1279         <==> (B in singleton null) [powerset-def]
1280         <==> (B = null) [singleton-characterization]
1281         <==> (B subset null) [subset-lemma-6]])
1282 | (A as (Set.insert h t:(Set 'S))) =>
1283   let {IH := (forall B . B in powerset t <==> B subset t)}
1284   pick-any B:(Set 'S)
1285   let {e1 := (!chain [(B in powerset A)
1286     <==> (B in (powerset t) \\/ (h in-all powerset t)) [powerset-def]
1287     <==> (B in powerset t | B in h in-all powerset t) [UC]
1288     <==> (B subset t | B in h in-all powerset t) [IH]
1289     <==> (B subset t | exists s . s in powerset t & B = h ++ s) [insert-in-all-characterization]
1290     <==> (B subset t | exists s . s subset t & B = h ++ s) [IH]]];
1291   lemma := (!chain-> [true ==> (h in h ++ t) [in-lemma-1]]);
1292   p3 := (assume hyp := (B subset t | exists s . s subset t & B = h ++ s)
1293     (!cases hyp
1294       (!chain [(B subset t) ==> (B subset A) [subset-lemma-5]]
1295         (assume ehyp := (exists s . s subset t & B = h ++ s)
1296           pick-witness s for ehyp
1297             (!subset-intro
1298               pick-any x
1299               assume (x in B)
1300                 (!chain-> [(x in B) ==> (x in h ++ s) [(B = h ++ s)]
1301                   ==> (x = h | x in s) [in-def]
1302                   ==> (x in h ++ t | x in s) [lemma]
1303                   ==> (x in A | x in t) [SC]
1304                   ==> (x in A | x in A) [in-def]
1305                   ==> (x in A) [prop-taut]]))));
1306   p4 := (assume (B subset A)
1307     (!two-cases
1308       assume case1 := (h in B)
1309       (!chain-> [(B subset A)
1310         ==> (B subset A & h in B) [augment]
1311         ==> (exists s . s subset t & B = h ++ s) [subset-lemma-3]
1312         ==> (B subset t | exists s . s subset t & B = h ++ s) [alternate]])
1313       assume case2 := (~ h in B)
1314       (!chain-> [case2 ==> (~ h in B & B subset A) [augment]
1315         ==> (B subset t) [subset-lemma-4]
1316         ==> (B subset t | exists s . s subset t & B = h ++ s) [alternate]]));
1317   p3<=>p4 := (!equiv p3 p4)}
1318   (!equiv-tran e1 p3<=>p4)
1319 }
1320
1321 define POSC := powerset-characterization
1322
1323 conclude ps-theorem-1 := (forall A . null in powerset A)
1324   pick-any A
1325   (!chain-> [true ==> (null subset A) [subset-def]
1326     ==> (null in powerset A) [POSC]])
1327
1328 conclude ps-theorem-2 := (forall A . A in powerset A)

```

```

1329 pick-any A
1330   (!chain-> [true ==> (A subset A)          [subset-reflexivity]
1331             ==> (A in powerset A) [POSC]])
1332
1333 conclude ps-theorem-3 :=
1334   (forall A B . A subset B <==> powerset A subset powerset B)
1335 pick-any A B
1336   (!equiv assume (A subset B)
1337     (!subset-intro
1338       pick-any C
1339         (!chain [(C in powerset A)
1340                 ==> (C subset A)          [POSC]
1341                 ==> (C subset B)          [subset-transitivity]
1342                 ==> (C in powerset B)      [POSC]]))
1343       assume (powerset A subset powerset B)
1344         (!chain-> [true ==> (A in powerset A) [ps-theorem-2]
1345                   ==> (A in powerset B) [SC]
1346                   ==> (A subset B)      [POSC]]))
1347
1348 conclude ps-theorem-4 :=
1349   (forall A B . powerset A /\ B = (powerset A) /\ (powerset B))
1350 pick-any A B
1351   (!set-identity-intro-direct
1352     pick-any C
1353       (!chain
1354         [(C in powerset A /\ B)
1355          <==> (C subset A /\ B)          [POSC]
1356          <==> (C subset A & C subset B) [intersection-subset-theorem']
1357          <==> (C in powerset A & C in powerset B) [POSC]
1358          <==> (C in (powerset A) /\ (powerset B)) [IC]])
1359
1360 conclude ps-theorem-5 :=
1361   (forall A B . (powerset A) \/ (powerset B) subset powerset A \/ B)
1362 pick-any A B
1363   (!subset-intro
1364     pick-any C
1365       (!chain [(C in (powerset A) \/ (powerset B))
1366               ==> (C in powerset A | C in powerset B) [UC]
1367               ==> (C subset A | C subset B)          [POSC]
1368               ==> (C subset A \/ B)                  [union-subset-theorem]
1369               ==> (C in powerset A \/ B)              [POSC]]))
1370
1371 declare paired-with: (S, T) [S (Set T)] -> (Set (Pair S T))
1372                                     [130 [id alist->set]]
1373
1374 assert* paired-with-def :=
1375   [(_ paired-with null = null)
1376    (x paired-with h ++ t = x @ h ++ (x paired-with t))]
1377
1378 conclude paired-with-characterization :=
1379   (forall B x y a . x @ y in a paired-with B <==> x = a & y in B)
1380 by-induction paired-with-characterization {
1381   null => pick-any x y a
1382     (!chain [(x @ y in a paired-with null)
1383             <==> (x @ y in null)          [paired-with-def]
1384             <==> false                    [null-characterization]
1385             <==> (x = a & false)          [prop-taut]
1386             <==> (x = a & y in null) [null-characterization]])
1387 | (B as (insert h t)) =>
1388   pick-any x y a
1389   let {IH := (forall x y a . x @ y in a paired-with t <==> x = a & y in t)}
1390     (!chain
1391       [(x @ y in a paired-with h ++ t)
1392        <==> (x @ y in a @ h ++ (a paired-with t)) [paired-with-def]
1393        <==> (x @ y = a @ h | x @ y in a paired-with t) [in-def]
1394        <==> (x = a & y = h | x @ y in a paired-with t) [pair-axioms]
1395        <==> (x = a & y = h | x = a & y in t)          [IH]
1396        <==> (x = a & (y = h | y in t))                [prop-taut]
1397        <==> (x = a & y in B)                            [in-def]])
1398 }

```

```

1399
1400 conclude paired-with-lemma-1 :=
1401   (forall A x . x paired-with A = null ==> A = null)
1402 datatype-cases paired-with-lemma-1 {
1403   null => pick-any x
1404     (!chain [(x paired-with null = null)
1405               ==> (null = null)] [paired-with-def])
1406 | (insert h t) =>
1407   pick-any x
1408   (!chain
1409     [(x paired-with h ++ t = null)
1410      ==> (x @ h ++ (x paired-with t) = null) [paired-with-def]
1411      ==> (forall z . ~ z in x @ h ++ (x paired-with t)) [NC-2]
1412      ==> (forall z . ~ (z = x @ h | z in x paired-with t)) [in-def]
1413
1414      ==> (forall z . z != x @ h) [prop-taut]
1415      ==> (x @ h != x @ h) [(uspec with x @ h)]
1416      ==> (x @ h != x @ h & x @ h = x @ h) [augment]
1417      ==> false [prop-taut]
1418      ==> (h ++ t = null) [prop-taut]])
1419 }
1420
1421 declare product: (S, T) [(Set S) (Set T)] -> (Set (Pair S T)) [150 [alist->set alist->set]]
1422
1423 define X := product
1424
1425 assert* product-def :=
1426   [(null X _ = null)
1427    (h ++ t X A = h paired-with A \ / t X A)]
1428
1429 conclude cartesian-product-characterization :=
1430   (forall A B a b . a @ b in A X B <==> a in A & b in B)
1431 by-induction cartesian-product-characterization {
1432   null => pick-any B a b
1433     (!chain [(a @ b in null X B)
1434               <==> (a @ b in null) [product-def]
1435               <==> false [null-characterization]
1436               <==> (a in null & b in B) [prop-taut null-characterization]])
1437 | (A as (insert h t)) =>
1438   let {IH := (forall B a b . a @ b in t X B <==> a in t & b in B)}
1439   pick-any B a b
1440   (!chain [(a @ b in h ++ t X B)
1441             <==> (a @ b in h paired-with B \ / t X B) [product-def]
1442             <==> (a @ b in h paired-with B | a @ b in t X B) [UC]
1443             <==> (a = h & b in B | a in t & b in B) [paired-with-characterization IH]
1444             <==> ((a = h | a in t) & b in B) [prop-taut]
1445             <==> (a in A & b in B) [in-def]])
1446 }
1447
1448 define CPC := cartesian-product-characterization
1449
1450 conclude cartesian-product-characterization-2 :=
1451   (forall x A B . x in A X B <==> exists a b . x = a @ b & a in A & b in B)
1452 pick-any x A B
1453   (!equiv
1454     assume hyp := (x in A X B)
1455     let {p := (!chain-> [true ==> (exists a b . x = a @ b) [pair-axioms]])}
1456     pick-witnesses a b for p x=a@b
1457       (!chain-> [x=a@b ==> (a @ b in A X B) [hyp]
1458                 ==> (a in A & b in B) [CPC]
1459                 ==> (x=a@b & a in A & b in B) [augment]
1460                 ==> (exists a b . x = a @ b & a in A & b in B) [existence]])
1461     assume hyp := (exists a b . x = a @ b & a in A & b in B)
1462     pick-witnesses a b for hyp spec-premise
1463       (!chain-> [spec-premise
1464                 ==> (a in A & b in B) [prop-taut]
1465                 ==> (a @ b in A X B) [CPC]
1466                 ==> (x in A X B) [(x = a @ b)])])
1467
1468 define CPC-2 := cartesian-product-characterization-2

```

```

1469
1470 define taut := (method (p q) (!prove-from q [p]))
1471
1472 conclude product-theorem-1 :=
1473   (forall A B . A X B = null ==> A = null | B = null)
1474 datatype-cases product-theorem-1 {
1475   null => pick-any B
1476     (!chain [(null X B = null)
1477       ==> (null = null) [product-def]
1478       ==> (null = null | B = null) [alternate]])
1479   | (A as (insert h t)) =>
1480     pick-any B
1481     (!chain [(h ++ t X B = null)
1482       ==> (h paired-with B \ / t X B = null) [product-def]
1483       ==> (h paired-with B = null & t X B = null) [union-null-theorem]
1484       ==> (B = null) [paired-with-lemma-1]
1485       ==> (h ++ t = null | B = null) [alternate]])
1486 }
1487
1488 conclude product-theorem-2 :=
1489   (forall A B . A X B = null <==> A = null | B = null)
1490 pick-any A:(Set 'T1) B:(Set 'T2)
1491   (!chain [(A X B = null)
1492     <==> (forall x . ~ x in A X B) [NC-2]
1493     <==> (forall x . ~ exists a b . x = a @ b & a in A & b in B) [CPC-2]
1494     <==> (forall x a b . a in A & b in B ==> x /= a @ b) [taut]
1495     <==> (forall a b . a in A & b in B ==> forall x . x /= a @ b) [taut]
1496     <==> (forall a b . a in A & b in B ==> false) [taut]
1497     <==> (forall a b . ~ a in A | ~ b in B) [taut]
1498     <==> ((forall a . ~ a in A) | (forall b . ~ b in B)) [taut]
1499     <==> (A = null | B = null) [NC-2]])
1500
1501 conclude product-theorem-3 :=
1502   (forall A B . non-empty A & non-empty B ==> A X B = B X A <==> A = B)
1503 pick-any A:(Set 'S) B:(Set 'T)
1504 assume hyp := (non-empty A & non-empty B)
1505 let {p1 := (!chain-> [(non-empty A) ==> (exists a . a in A) [NC-3]]);
1506   p2 := (!chain-> [(non-empty B) ==> (exists b . b in B) [NC-3]]);
1507   M := method (S1 S2 c2) # assumes c2 in S2, S1 X S2 = S2 X S1,
1508     (!subset-intro # and derives (S1 subset S2)
1509       pick-any x
1510         (!chain [(x in S1)
1511           ==> (x in S1 & c2 in S2) [augment]
1512           ==> (x @ c2 in S1 X S2) [CPC]
1513           ==> (x @ c2 in S2 X S1) [SIC]
1514           ==> (x in S2 & c2 in S1) [CPC]
1515           ==> (x in S2) [left-and]]))
1516   }
1517 pick-witness a for p1 # (a in A)
1518 pick-witness b for p2 # (b in B)
1519   (!equiv
1520     assume hyp := (A X B = B X A)
1521     (!set-identity-intro (!M A B b) (!M B A a))
1522     assume hyp := (A = B)
1523     (!chain-> [(A X A = A X A) ==> (A X B = B X A) [hyp]]))
1524
1525 conclude product-theorem-4 :=
1526   (forall A B C . non-empty A & A X B subset A X C ==> B subset C)
1527 pick-any A B C
1528 assume hyp := (non-empty A & A X B subset A X C)
1529 pick-witness a for (!chain-> [hyp ==> (exists a . a in A) [NC-3]])
1530   (!subset-intro
1531     pick-any b
1532     (!chain [(b in B)
1533       ==> (a in A & b in B) [augment]
1534       ==> (a @ b in A X B) [CPC]
1535       ==> (a @ b in A X C) [SC]
1536       ==> (a in A & b in C) [CPC]
1537       ==> (b in C) [right-and]]))
1538

```

```

1539 define pair-converter :=
1540   method (premise)
1541     match premise {
1542       (forall u:'S (forall v:'T body)) =>
1543         pick-any p:(Pair 'S 'T)
1544         let {E := (!chain-> [true ==> (exists ?x:'S ?y:'T .
1545                                   p = ?x @ ?y) [pair-axioms]])}
1546         pick-witnesses x y for E
1547         let {body' := (!uspec* premise [x y])}
1548         (!chain-> [body'
1549                   ==> (replace-term-in-sentence (x @ y) body' p)
1550                     [(p = x @ y)])])
1551     }
1552
1553 conclude product-theorem-5 :=
1554   (forall A B C . B subset C ==> A X B subset A X C)
1555 pick-any A B C
1556   assume (B subset C)
1557   (!subset-intro
1558     (!pair-converter
1559       pick-any a b
1560       (!chain [(a @ b in A X B)
1561                ==> (a in A & b in B) [CPC]
1562                ==> (a in A & b in C) [SC]
1563                ==> (a @ b in A X C) [CPC]])))
1564
1565 conclude product-theorem-6 :=
1566   (forall A B C . A X (B /\ C) = A X B /\ A X C)
1567 pick-any A B C
1568   (!set-identity-intro-direct
1569     (!pair-converter
1570       pick-any x y
1571       (!chain [(x @ y in A X (B /\ C))
1572                <==> (x in A & y in B /\ C) [CPC]
1573                <==> (x in A & y in B & y in C) [IC]
1574                <==> ((x in A & y in B) & (x in A & y in C)) [prop-taut]
1575                <==> (x @ y in A X B & x @ y in A X C) [CPC]
1576                <==> (x @ y in A X B /\ A X C) [IC]])))
1577
1578 # Theorem 103:
1579 conclude product-theorem-7 :=
1580   (forall A B C . A X (B \ / C) = A X B \ / A X C)
1581 pick-any A B C
1582   (!set-identity-intro-direct
1583     (!pair-converter
1584       pick-any x y
1585       (!chain [(x @ y in A X (B \ / C))
1586                <==> (x in A & y in B \ / C) [CPC]
1587                <==> (x in A & (y in B | y in C)) [UC]
1588                <==> ((x in A & y in B) | (x in A & y in C)) [prop-taut]
1589                <==> (x @ y in A X B | x @ y in A X C) [CPC]
1590                <==> (x @ y in A X B \ / A X C) [UC]])))
1591
1592 # Theorem 104:
1593 conclude product-theorem-8 :=
1594   (forall A B C . A X (B \ C) = A X B \ A X C)
1595 pick-any A B C
1596   (!set-identity-intro-direct
1597     (!pair-converter
1598       pick-any x y
1599       (!chain [(x @ y in A X (B \ C))
1600                <==> (x in A & y in B \ C) [CPC]
1601                <==> (x in A & y in B & ~ y in C) [DC]
1602                <==> ((x in A & y in B) & (~x in A | ~ y in C)) [prop-taut]
1603                <==> ((x in A & y in B) & ~ (x in A & y in C)) [prop-taut]
1604                <==> (x @ y in A X B & ~ x @ y in A X C) [CPC]
1605                <==> (x @ y in A X B \ A X C) [DC]])))
1606
1607 define [R R1 R2 R3 R4] :=
1608   [?R:(Set (Pair 'T14 'T15)) ?R1:(Set (Pair 'T16 'T17))

```

```

1609      ?R2:(Set (Pair 'T18 'T19)) ?R3:(Set (Pair 'T20 'T21))
1610      ?R4:(Set (Pair 'T22 'T23))]
1611
1612 #===== RELATION DOMAINS AND RANGES
1613
1614 declare dom: (S, T) [(Set (Pair S T))] -> (Set S) [150 [alist->set]]
1615
1616 assert* dom-def :=
1617   [(dom null = null)
1618    (dom x @ _ ++ t = x ++ dom t)]
1619
1620 declare range: (S, T) [(Set (Pair S T))] -> (Set T) [150 [alist->set]]
1621
1622 assert* range-def :=
1623   [(range null = null)
1624    (range _ @ y ++ t = y ++ range t)]
1625
1626 conclude in-dom-lemma-1 :=
1627   (forall R a x y . a = x ==> a in dom x @ y ++ R)
1628 pick-any R a x y
1629   (!chain [(a = x) ==> (a in x ++ dom R) [in-def]
1630            ==> (a in dom x @ y ++ R) [dom-def]])
1631
1632 conclude in-range-lemma-1 :=
1633   (forall R a x y . a = y ==> a in range x @ y ++ R)
1634 pick-any R a x y
1635   (!chain [(a = y) ==> (a in y ++ range R) [in-def]
1636            ==> (a in range x @ y ++ R) [range-def]])
1637
1638 conclude in-dom-lemma-2 :=
1639   (forall R x a b . x in dom R ==> x in dom a @ b ++ R)
1640 pick-any R x a b
1641   (!chain [(x in dom a @ b ++ R)
1642            <== (x in a ++ dom R) [dom-def]
1643            <== (x in dom R) [in-def]])
1644
1645 conclude in-range-lemma-2 :=
1646   (forall R y a b . y in range R ==> y in range a @ b ++ R)
1647 pick-any R y a b
1648   (!chain [(y in range a @ b ++ R)
1649            <== (y in b ++ range R) [range-def]
1650            <== (y in range R) [in-def]])
1651
1652
1653 conclude dom-characterization :=
1654   (forall R x . x in dom R <==> exists y . x @ y in R)
1655 by-induction dom-characterization {
1656   null => pick-any x
1657     (!chain [(x in dom null)
1658              <==> (x in null) [dom-def]
1659              <==> false [NC]
1660              <==> (exists y . false) [taut]
1661              <==> (exists y . x @ y in null) [NC]])
1662 | (R as (insert (pair a:'S b) t)) =>
1663   let {IH := (forall x . x in dom t <==> exists y . x @ y in t)}
1664   pick-any x:'S
1665   let {p1 := assume hyp := (x in dom R)
1666       (!chain<- [(x = a | x in dom t)
1667                 <== (x in a ++ dom t) [in-def]
1668                 <== hyp [dom-def]])
1669
1670       assume case1 := (x = a)
1671       (!chain-> [true ==> (a @ b in R) [in-lemma-1]
1672                ==> (x @ b in R) [case1]
1673                ==> (exists y . x @ y in R) [existence]])
1674       assume case2 := (x in dom t)
1675       (!chain-> [case2 ==> (exists y . x @ y in t) [IH]
1676                ==> (exists y . x @ y in R) [in-def]])}
1677   p2 := (!chain [(exists y . x @ y in R)
1678                 ==> (exists y . x @ y = a @ b | x @ y in t) [in-def])

```



```

1679         ==> (exists y . x = a | x @ y in t)          [pair-axioms]
1680         ==> (exists y . x in dom R | x @ y in t)      [in-dom-lemma-1]
1681         ==> (exists y . x in dom R | exists z . x @ z in t) [in-dom-lemma-1 taut]
1682         ==> (exists y . x in dom R | x in dom t)      [IH]
1683         ==> (exists y . x in dom R | x in dom R)      [in-dom-lemma-2]
1684         ==> (x in dom R)                             [taut]]]
1685     }
1686     (!equiv p1 p2)
1687 }
1688
1689 define DOMC := dom-characterization
1690
1691 conclude range-characterization :=
1692   (forall R y . y in range R <==> exists x . x @ y in R)
1693 by-induction range-characterization {
1694   null => pick-any y
1695     (!chain [(y in range null)
1696       <==> (y in null)          [range-def]
1697       <==> false              [NC]
1698       <==> (exists y . false)  [taut]
1699       <==> (exists x . x @ y in null) [NC]])
1700 | (R as (insert (pair a b:'T) t)) =>
1701   let {IH := (forall y . y in range t <==> exists x . x @ y in t)}
1702   pick-any y:'T
1703   let {p1 := assume hyp := (y in range R)
1704     (!cases (!chain<- [(y = b | y in range t)
1705       <== (y in b ++ range t) [in-def]
1706       <== hyp                [range-def]])
1707     assume case1 := (y = b)
1708       (!chain-> [true ==> (a @ b in R) [in-lemma-1]
1709         ==> (a @ y in R) [case1]
1710         ==> (exists x . x @ y in R) [existence]])
1711     assume case2 := (y in range t)
1712       (!chain-> [case2 ==> (exists x . x @ y in t) [IH]
1713         ==> (exists x . x @ y in R) [in-def]]))};
1714   p2 := (!chain [(exists x . x @ y in R)
1715     ==> (exists x . x @ y = a @ b | x @ y in t) [in-def]
1716     ==> (exists x . y = b | x @ y in t)          [pair-axioms]
1717     ==> (exists x . y in range R | x @ y in t)    [in-range-lemma-1]
1718     ==> (exists x . y in range R | exists z . z @ y in t) [in-range-lemma-1 taut]
1719     ==> (exists x . y in range R | y in range t)    [IH]
1720     ==> (exists x . y in range R | y in range R)    [in-range-lemma-2]
1721     ==> (y in range R)                             [taut]])
1722   }
1723   (!equiv p1 p2)
1724 }
1725
1726 define RANC := range-characterization
1727
1728 conclude dom-theorem-1 :=
1729   (forall R1 R2 . dom (R1 \ / R2) = dom R1 \ / dom R2)
1730 pick-any R1 R2
1731   (!set-identity-intro-direct
1732     pick-any x
1733     (!chain
1734       [(x in dom (R1 \ / R2))
1735       <==> (exists y . x @ y in R1 \ / R2)          [DOMC]
1736       <==> (exists y . x @ y in R1 | x @ y in R2)    [UC]
1737       <==> ((exists y . x @ y in R1) | (exists y . x @ y in R2)) [taut]
1738       <==> (x in dom R1 | x in dom R2)              [DOMC]
1739       <==> (x in dom R1 \ / dom R2)                  [UC]]))
1740
1741 conclude range-theorem-1 :=
1742   (forall R1 R2 . range (R1 \ / R2) = range R1 \ / range R2)
1743 pick-any R1 R2

```

```

1749 (!set-identity-intro-direct
1750   pick-any y
1751   (!chain [(y in range (R1 \ R2))
1752            <==> (exists x . x @ y in R1 \ R2) [RANC]
1753            <==> (exists x . x @ y in R1 | x @ y in R2) [UC]
1754            <==> ((exists x . x @ y in R1) | (exists x . x @ y in R2)) [taut]
1755            <==> (y in range R1 | y in range R2) [RANC]
1756            <==> (y in range R1 \ range R2) [UC]]))
1757
1758
1759 conclude dom-theorem-2 :=
1760   (forall R1 R2 . dom (R1 /\ R2) subset dom R1 /\ dom R2)
1761 pick-any R1 R2
1762 (!subset-intro
1763   pick-any x
1764   (!chain [(x in dom (R1 /\ R2))
1765            ==> (exists y . x @ y in R1 /\ R2) [DOMC]
1766            ==> (exists y . x @ y in R1 & x @ y in R2) [IC]
1767            ==> ((exists y . x @ y in R1) & (exists y . x @ y in R2)) [taut]
1768            ==> (x in dom R1 & x in dom R2) [DOMC]
1769            ==> (x in dom R1 /\ dom R2) [IC]]))
1770
1771 #(falsify (forall R1 R2 . dom (R1 /\ R2) = dom R1 /\ dom R2) 10)
1772
1773 conclude range-theorem-2 :=
1774   (forall R1 R2 . range (R1 /\ R2) subset range R1 /\ range R2)
1775 pick-any R1 R2
1776 (!subset-intro
1777   pick-any y
1778   (!chain [(y in range (R1 /\ R2))
1779            ==> (exists x . x @ y in R1 /\ R2) [RANC]
1780            ==> (exists x . x @ y in R1 & x @ y in R2) [IC]
1781            ==> ((exists x . x @ y in R1) & (exists x . x @ y in R2)) [taut]
1782            ==> (y in range R1 & y in range R2) [RANC]
1783            ==> (y in range R1 /\ range R2) [IC]]))
1784
1785
1786 conclude dom-theorem-3 :=
1787   (forall R1 R2 . dom R1 \ dom R2 subset dom (R1 \ R2))
1788 pick-any R1 R2
1789 (!subset-intro
1790   pick-any x
1791   assume hyp := (x in dom R1 \ dom R2)
1792   let {lemma := (!chain-> [hyp ==> (x in dom R1 & ~ x in dom R2) [DC]])}
1793   pick-witness w for (!chain-> [lemma ==> (x in dom R1) [left-and]
1794                                ==> (exists y . x @ y in R1) [DOMC]])
1795   (!chain-> [lemma ==> (~ x in dom R2) [right-and]
1796            ==> (~ exists y . x @ y in R2) [DOMC]
1797            ==> (forall y . ~ x @ y in R2) [qn]
1798            ==> (~ x @ w in R2) [(uspec with w)]
1799            ==> (x @ w in R1 & ~ x @ w in R2) [augment]
1800            ==> (exists y . x @ y in R1 & ~ x @ y in R2) [existence]
1801            ==> (exists y . x @ y in R1 \ R2) [DC]
1802            ==> (x in dom (R1 \ R2)) [DOMC]]))
1803
1804
1805 conclude range-theorem-3 :=
1806   (forall R1 R2 . range R1 \ range R2 subset range (R1 \ R2))
1807 pick-any R1 R2
1808 (!subset-intro
1809   pick-any y
1810   assume hyp := (y in range R1 \ range R2)
1811   let {lemma := (!chain-> [hyp ==> (y in range R1 & ~ y in range R2) [DC]])}
1812   pick-witness w for (!chain-> [lemma ==> (y in range R1) [left-and]
1813                                ==> (exists x . x @ y in R1) [RANC]])
1814   (!chain-> [lemma ==> (~ y in range R2) [right-and]
1815            ==> (~ exists x . x @ y in R2) [RANC]
1816            ==> (forall x . ~ x @ y in R2) [qn]
1817            ==> (~ w @ y in R2) [(uspec with w)]
1818            ==> (w @ y in R1 & ~ w @ y in R2) [augment]

```

```

1819 ==> (exists x . x @ y in R1 & ~ x @ y in R2) [existence]
1820 ==> (exists x . x @ y in R1 \ R2) [DC]
1821 ==> (y in range (R1 \ R2)) [RANC]])
1822
1823
1824
1825 declare conv: (S, T) [(Set (Pair S T)) -> (Set (Pair T S)) [210 [alist->set]]]
1826 define -- := conv
1827
1828 assert* conv-def :=
1829   [ ( -- null = null)
1830     ( -- x @ y ++ t = y @ x ++ -- t) ]
1831
1832
1833 define pair-lemma-1 := Pair.pair-theorem-2
1834
1835 conclude converse-characterization :=
1836   (forall R x y . x @ y in -- R <==> y @ x in R)
1837 by-induction converse-characterization {
1838   null => pick-any x y
1839     (!chain [(x @ y in -- null)
1840               <==> (x @ y in null) [conv-def]
1841               <==> false [NC]
1842               <==> (y @ x in null) [NC]])
1843
1844 | (R as (insert (pair a b) t)) =>
1845   let {
1846     IH := (forall x y . x @ y in -- t <==> y @ x in t)
1847     pick-any x y
1848       (!chain [(x @ y in -- R)
1849                 <==> (x @ y in b @ a ++ -- t) [conv-def]
1850                 <==> (x @ y = b @ a | x @ y in -- t) [in-def]
1851                 <==> (y @ x = a @ b | x @ y in -- t) [pair-lemma-1]
1852                 <==> (y @ x = a @ b | y @ x in t) [IH]
1853                 <==> (y @ x in R) [in-def]])
1854   }
1855
1856
1857 conclude converse-theorem-1 :=
1858   (forall R . -- -- R = R)
1859 by-induction converse-theorem-1 {
1860   null => (!chain [(-- -- null) = (-- null) [conv-def]
1861                   = null [conv-def]])
1862 | (R as (insert (pair x y) t)) =>
1863   let {IH := (-- -- t = t)}
1864     (!chain [(-- -- x @ y ++ t)
1865               = (-- (y @ x ++ -- t)) [conv-def]
1866               = (x @ y ++ -- t) [conv-def]
1867               = (x @ y ++ t) [IH]])
1868   }
1869
1870 conclude converse-theorem-2 :=
1871   (forall R1 R2 . -- (R1 /\ R2) = -- R1 /\ -- R2)
1872 pick-any R1 R2
1873 (!set-identity-intro-direct
1874   (!pair-converter
1875     pick-any x y
1876       (!chain [(x @ y in -- (R1 /\ R2))
1877                 <==> (y @ x in R1 /\ R2) [converse-characterization]
1878                 <==> (y @ x in R1 & y @ x in R2) [IC]
1879                 <==> (x @ y in -- R1 & x @ y in -- R2) [converse-characterization]
1880                 <==> (x @ y in -- R1 /\ -- R2) [IC]]))
1881
1882
1883 conclude converse-theorem-3 :=
1884   (forall R1 R2 . -- (R1 \ R2) = -- R1 \ -- R2)
1885 pick-any R1 R2
1886 (!set-identity-intro-direct
1887   (!pair-converter
1888     pick-any x y

```

```

1889      (!chain [(x @ y in -- (R1 \ R2))
1890        <==> (y @ x in R1 \ R2) [converse-characterization]
1891        <==> (y @ x in R1 | y @ x in R2) [UC]
1892        <==> (x @ y in -- R1 | x @ y in -- R2) [converse-characterization]
1893        <==> (x @ y in -- R1 \ -- R2) [UC]]))
1894
1895
1896 conclude converse-theorem-4 :=
1897   (forall R1 R2 . -- (R1 \ R2) = -- R1 \ -- R2)
1898 pick-any R1 R2
1899   (!set-identity-intro-direct
1900     (!pair-converter
1901       pick-any x y
1902         (!chain [(x @ y in -- (R1 \ R2))
1903           <==> (y @ x in R1 \ R2) [converse-characterization]
1904           <==> (y @ x in R1 & ~ y @ x in R2) [DC]
1905           <==> (x @ y in -- R1 & ~ x @ y in -- R2) [converse-characterization]
1906           <==> (x @ y in -- R1 \ -- R2) [DC]]))
1907
1908
1909 declare composed-with: (S1, S2, S3) [(Pair S1 S2) (Set (Pair S2 S3))] -> (Set (Pair S1 S3)) [200 [id alist->set]]
1910
1911 assert* composed-with-def :=
1912   [(_ composed-with null = null)
1913     (x @ y composed-with z @ w ++ t = x @ w ++ (x @ y composed-with t) <== y = z)
1914     (x @ y composed-with z @ w ++ t = x @ y composed-with t <== y /= z)]
1915
1916
1917
1918 conclude composed-with-characterization :=
1919   (forall R x y z w . w @ z in x @ y composed-with R <==> w = x & y @ z in R)
1920 by-induction composed-with-characterization {
1921   (R as null) => pick-any x y z w
1922     (!chain [(w @ z in x @ y composed-with null)
1923       <==> (w @ z in null) [composed-with-def]
1924       <==> false [NC]
1925       <==> (w = x & y @ z in null) [prop-taut NC]])
1926
1927 | (R as (insert (pair a b) t)) =>
1928   pick-any x y z w
1929   let {IH := (forall x y z w . w @ z in x @ y composed-with t <==> w = x & y @ z in t)}
1930   (!two-cases
1931     assume case1 := (y = a)
1932     (!chain [(w @ z in x @ y composed-with a @ b ++ t)
1933       <==> (w @ z in x @ b ++ (x @ y composed-with t)) [composed-with-def]
1934       <==> (w @ z = x @ b | w @ z in x @ y composed-with t) [in-def]
1935       <==> (w @ z = x @ b | (w = x & y @ z in t)) [IH]
1936       <==> (w = x & z = b | w = x & y @ z in t) [pair-axioms]
1937       <==> (w = x & y = a & z = b | w = x & y @ z in t) [augment]
1938       <==> (w = x & y @ z = a @ b | w = x & y @ z in t) [pair-axioms]
1939       <==> (w = x & (y @ z = a @ b | y @ z in t)) [prop-taut]
1940       <==> (w = x & y @ z in R) [in-def]])
1941     assume case2 := (y /= a)
1942     (!iff-comm
1943       (!chain [(w = x & y @ z in R)
1944         <==> (w = x & (y @ z = a @ b | y @ z in t)) [in-def]
1945         <==> (w = x & (y = a & z = b | y @ z in t)) [pair-axioms]
1946         <==> (w = x & (case2 & y = a & z = b | y @ z in t)) [augment]
1947         <==> (w = x & (false | y @ z in t)) [prop-taut]
1948         <==> (w = x & y @ z in t) [prop-taut]
1949         <==> (w @ z in x @ y composed-with t) [IH]
1950         <==> (w @ z in x @ y composed-with R) [composed-with-def]]))
1951   }
1952
1953 conclude composed-with-characterization' :=
1954   (forall R x y z . x @ z in x @ y composed-with R <==> y @ z in R)
1955 pick-any R x y z
1956   (!chain [(x @ z in x @ y composed-with R)
1957     <==> (x = x & y @ z in R) [composed-with-characterization]
1958     <==> (y @ z in R) [augment]])

```

```

1959
1960
1961 declare o: (S1, S2, S3) [(Set (Pair S1 S2)) (Set (Pair S2 S3))] -> (Set (Pair S1 S3)) [200 [alist->set alist->set]]
1962
1963 assert* o-def :=
1964   [(null o _ = null)
1965    (x @ y ++ t o R = x @ y composed-with R \ / t o R)]
1966
1967 conclude o-characterization :=
1968   (forall R1 R2 x z . x @ z in R1 o R2 <==> exists y . x @ y in R1 & y @ z in R2)
1969 by-induction o-characterization {
1970   (R1 as null) => pick-any R2 x z
1971     (!chain [(x @ z in R1 o R2)
1972              <==> (x @ z in null) [o-def]
1973              <==> false [NC]
1974              <==> (exists y . false & y @ z in R2) [(method (p q) (!force q))]
1975              <==> (exists y . x @ y in null & y @ z in R2) [NC (method (p q) (!force q))]])
1976 | (R1 as (insert (pair a b) t)) =>
1977   pick-any R2 x z
1978   let {IH := (forall R2 x z . x @ z in t o R2 <==> exists y . x @ y in t & y @ z in R2)}
1979   let {dirl := assume hyp := (x @ z in R1 o R2)
1980       (!cases (!chain-> [hyp
1981                          ==> (x @ z in a @ b composed-with R2 \ / t o R2) [o-def]
1982                          ==> (x @ z in a @ b composed-with R2 | x @ z in t o R2) [UC]
1983                          ==> (x @ z in a @ b composed-with R2 | exists y . x @ y in t & y @ z in R2) [
1984                          ==> (x @ z in a @ b composed-with R2 | exists y . x @ y in R1 & y @ z in R2) [
1985                          ==> (x = a & b @ z in R2 | exists y . x @ y in R1 & y @ z in R2) [composed-w
1986                          assume case1 := (x = a & b @ z in R2)
1987                          (!chain-> [true ==> (a @ b in R1) [in-lemma-1]
1988                                   ==> (x @ b in R1) [case1]
1989                                   ==> (x @ b in R1 & b @ z in R2) [augment]
1990                                   ==> (exists y . x @ y in R1 & y @ z in R2) [taut]]])
1991                          assume case2 := (exists y . x @ y in R1 & y @ z in R2)
1992                          (!claim case2));
1993                          dir2 := assume hyp := (exists y . x @ y in R1 & y @ z in R2)
1994                          pick-witness y for hyp
1995                          (!cases (!chain-> [(x @ y in R1)
1996                                             ==> (x @ y = a @ b | x @ y in t) [in-def]])
1997                          assume case1 := (x @ y = a @ b)
1998                          let {_ := (!chain-> [case1 ==> (x = a) [pair-axioms]]};
1999                          _ := (!chain-> [case1 ==> (y = b) [pair-axioms]])}
2000                          (!chain-> [(x = a)
2001                                   ==> (x = a & y @ z in R2) [augment]
2002                                   ==> (x = a & b @ z in R2) [(y = b)]
2003                                   ==> (x @ z in a @ b composed-with R2) [composed-with-characterization]
2004                                   ==> (x @ z in a @ b composed-with R2 \ / t o R2) [UC]
2005                                   ==> (x @ z in R1 o R2) [o-def]])
2006                          assume case2 := (x @ y in t)
2007                          (!chain-> [case2
2008                                   ==> (x @ y in t & y @ z in R2) [augment]
2009                                   ==> (exists y . x @ y in t & y @ z in R2) [existence]
2010                                   ==> (x @ z in t o R2) [IH]
2011                                   ==> (x @ z in a @ b composed-with R2 | x @ z in t o R2) [prop-taut]
2012                                   ==> (x @ z in a @ b composed-with R2 \ / t o R2) [UC]
2013                                   ==> (x @ z in R1 o R2) [o-def]])])
2014       }
2015       (!equiv dirl dir2)
2016 }
2017
2018 conclude compose-theorem-1 :=
2019   (forall R1 R2 . dom R1 o R2 subset dom R1)
2020 pick-any R1 R2
2021   (!subset-intro
2022     pick-any x
2023       (!chain [(x in dom R1 o R2)
2024                ==> (exists y . x @ y in R1 o R2) [dom-characterization]
2025                ==> (exists y . exists z . x @ z in R1 & z @ y in R2) [o-characterization]
2026                ==> (exists y . exists z . x @ z in R1) [taut]
2027                ==> (exists y . x in dom R1) [dom-characterization]
2028                ==> (x in dom R1) [taut]]))

```

```

2029
2030 conclude compose-theorem-2 :=
2031   (forall R1 R2 R3 R4 . R1 subset R2 & R3 subset R4 ==> R1 o R3 subset R2 o R4)
2032 pick-any R1:(Set (Pair 'S 'T)) R2:(Set (Pair 'S 'T))
2033   R3:(Set (Pair 'T 'U)) R4:(Set (Pair 'T 'U))
2034   assume hyp := (R1 subset R2 & R3 subset R4)
2035   (!subset-intro
2036     (!pair-converter
2037       pick-any x y
2038       (!chain [(x @ y in R1 o R3)
2039         ==> (exists z . x @ z in R1 & z @ y in R3) [o-characterization]
2040         ==> (exists z . x @ z in R2 & z @ y in R3) [SC]
2041         ==> (exists z . x @ z in R2 & z @ y in R4) [SC]
2042         ==> (x @ y in R2 o R4) [o-characterization]])))
2043
2044 conclude compose-theorem-3 :=
2045   (forall R1 R2 R3 . R1 o (R2 \ / R3) = R1 o R2 \ / R1 o R3)
2046 pick-any R1 R2 R3
2047   (!set-identity-intro-direct
2048     (!pair-converter
2049       pick-any x y
2050       (!chain [(x @ y in R1 o (R2 \ / R3))
2051         <==> (exists z . x @ z in R1 & z @ y in R2 \ / R3) [o-characterization]
2052         <==> (exists z . x @ z in R1 & (z @ y in R2 | z @ y in R3)) [UC]
2053         <==> (exists z . x @ z in R1 & z @ y in R2 | x @ z in R1 & z @ y in R3) [prop-taut]
2054         <==> ((exists z . x @ z in R1 & z @ y in R2) | (exists z . x @ z in R1 & z @ y in R3)) [taut]
2055         <==> (x @ y in R1 o R2 | x @ y in R1 o R3) [o-characterization]
2056         <==> (x @ y in R1 o R2 \ / R1 o R3) [UC]])))
2057
2058 conclude compose-theorem-4 :=
2059   (forall R1 R2 R3 . R1 o (R2 /\ R3) subset R1 o R2 /\ R1 o R3)
2060 pick-any R1 R2 R3
2061   (!subset-intro
2062     (!pair-converter
2063       pick-any x y
2064       (!chain [(x @ y in R1 o (R2 /\ R3))
2065         ==> (exists z . x @ z in R1 & z @ y in R2 /\ R3) [o-characterization]
2066         ==> (exists z . x @ z in R1 & (z @ y in R2 & z @ y in R3)) [IC]
2067         ==> (exists z . (x @ z in R1 & z @ y in R2) & (x @ z in R1 & z @ y in R3)) [prop-taut]
2068         ==> ((exists z . x @ z in R1 & z @ y in R2) & (exists z . x @ z in R1 & z @ y in R3)) [taut]
2069         ==> (x @ y in R1 o R2 & x @ y in R1 o R3) [o-characterization]
2070         ==> (x @ y in R1 o R2 /\ R1 o R3) [IC]])))
2071
2072 conclude compose-theorem-5 :=
2073   (forall R1 R2 R3 . R1 o R2 \ R1 o R3 subset R1 o (R2 \ R3))
2074 pick-any R1 R2 R3
2075   (!subset-intro
2076     (!pair-converter
2077       pick-any x y
2078       (!chain [(x @ y in R1 o R2 \ R1 o R3)
2079         ==> (x @ y in R1 o R2 & ~ x @ y in R1 o R3) [DC]
2080         ==> ((exists z . x @ z in R1 & z @ y in R2) & ~ (exists z . x @ z in R1 & z @ y in R3)) [o-characterization]
2081         ==> (exists z . x @ z in R1 & z @ y in R2 & ~ z @ y in R3) [taut]
2082         ==> (exists z . x @ z in R1 & z @ y in R2 \ R3) [DC]
2083         ==> (x @ y in R1 o (R2 \ R3)) [o-characterization]])))
2084
2085 conclude composition-assoc :=
2086   (forall R1 R2 R3 . R1 o R2 o R3 = (R1 o R2) o R3)
2087 pick-any R1 R2 R3
2088   (!set-identity-intro-direct
2089     (!pair-converter
2090       pick-any x y
2091       (!chain [(x @ y in R1 o R2 o R3)
2092         <==> (exists z . x @ z in R1 & z @ y in R2 o R3) [o-characterization]
2093         <==> (exists z . x @ z in R1 & exists w . z @ w in R2 & w @ y in R3) [o-characterization]
2094         <==> (exists w z . x @ z in R1 & z @ w in R2 & w @ y in R3) [taut]
2095         <==> (exists w . (exists z . x @ z in R1 & z @ w in R2) & w @ y in R3) [taut]
2096         <==> (exists w . x @ w in R1 o R2 & w @ y in R3) [o-characterization]
2097         <==> (x @ y in (R1 o R2) o R3) [o-characterization]])))
2098

```

```

2099 conclude compose-theorem-6 :=
2100   (forall R1 R2 . -- (R1 o R2) = -- R2 o -- R1)
2101 pick-any R1 R2
2102   (!set-identity-intro-direct
2103     (!pair-converter
2104       pick-any x y
2105         (!chain [(x @ y in -- (R1 o R2))
2106           <==> (y @ x in R1 o R2) [converse-characterization]
2107           <==> (exists z . y @ z in R1 & z @ x in R2) [o-characterization]
2108           <==> (exists z . z @ y in -- R1 & x @ z in -- R2) [converse-characterization]
2109           <==> (exists z . x @ z in -- R2 & z @ y in -- R1) [prop-taut]
2110           <==> (x @ y in -- R2 o -- R1) [o-characterization]])))
2111
2112 declare restrict1: (S, T) [(Set (Pair S T)) S] -> (Set (Pair S T)) [200 [alist->set id]]
2113
2114 assert* restrict1-def :=
2115   [(null restrict1 _ = null)
2116     (x @ y ++ t restrict1 z = x @ y ++ (t restrict1 z) <== x = z)
2117     (x @ y ++ t restrict1 z = t restrict1 z <== x /= z)]
2118
2119 define restrict1-characterization :=
2120   (forall R x y a . x @ y in R restrict1 a <==> x @ y in R & x = a)
2121
2122 (define ^1 restrict1)
2123
2124 conclude restrict1-lemma :=
2125   (forall R x y a . x @ y in R & x = a ==> x @ y in R ^1 a)
2126 by-induction restrict1-lemma {
2127   (R as null) => pick-any x y a
2128     (!chain [(x @ y in R & x = a)
2129       ==> (x @ y in R) [left-and]
2130       ==> false [NC]
2131       ==> (x @ y in R ^1 a) [prop-taut]])
2132 | (R as (insert (pair x' y') t)) =>
2133   let {IH := (forall x y a . x @ y in t & x = a ==> x @ y in t ^1 a)}
2134   pick-any x y a
2135     assume hyp := (x @ y in R & x = a)
2136     (!two-cases
2137       assume case1 := (x' = a)
2138       (!chain-> [hyp
2139         ==> ((x @ y = x' @ y' | x @ y in t) & x = a) [in-def]
2140         ==> (x @ y = x' @ y' & x = a | x @ y in t & x = a) [prop-taut]
2141         ==> (x @ y in x' @ y' ++ (t ^1 a) & x = a | x @ y in t & x = a) [in-def]
2142         ==> (x @ y in R ^1 a & x = a | x @ y in t & x = a) [restrict1-def]
2143         ==> (x @ y in R ^1 a & x = a | x @ y in t ^1 a) [IH]
2144         ==> (x @ y in R ^1 a & x = a | x @ y in x' @ y' ++ (t ^1 a)) [in-def]
2145         ==> (x @ y in R ^1 a & x = a | x @ y in R ^1 a) [restrict1-def]
2146         ==> (x @ y in R ^1 a) [prop-taut]])
2147       assume case2 := (x' /= a)
2148       (!cases (!chain-> [hyp
2149         ==> ((x @ y = x' @ y' | x @ y in t) & x = a) [in-def]
2150         ==> ((x = x' & y = y' | x @ y in t) & x = a) [pair-axioms]
2151         ==> (x = x' & y = y' & x = a | x @ y in t & x = a) [prop-taut]])
2152         assume hyp1 := (x = x' & y = y' & x = a)
2153         let {_ := (!absurd (!chain-> [hyp1 ==> (x = a)
2154           ==> (x' = a)])
2155           case2))
2156         (!from-false (x @ y in R ^1 a))
2157         assume hyp2 := (x @ y in t & x = a)
2158         (!chain-> [hyp2 ==> (x @ y in t ^1 a) [IH]
2159           ==> (x @ y in R ^1 a) [restrict1-def]]))
2160   }
2161
2162 by-induction restrict1-characterization {
2163   (R as null) => pick-any x y a
2164     (!chain [(x @ y in R ^1 a)
2165       <==> (x @ y in null) [restrict1-def]
2166       <==> false [NC]
2167       <==> (false & x = a) [prop-taut]
2168       <==> (x @ y in R & x = a) [NC]])

```

```

2169 | (R as (insert (pair x' y') t)) =>
2170   pick-any x y a
2171   let {IH := (forall x y a . x @ y in t ^1 a <==> x @ y in t & x = a);
2172       goal := (x @ y in R ^1 a <==> x @ y in R & x = a);
2173       dir1 := assume hyp := (x @ y in R ^1 a)
2174           (!two-cases
2175             assume case1 := (x' = a)
2176             (!chain-> [hyp
2177               ==> (x @ y in x' @ y' ++ (t ^1 a)) [restrict1-def]
2178               ==> (x @ y = x' @ y' | x @ y in t ^1 a) [in-def]])
2179             assume hyp1a := (x @ y = x' @ y')
2180             (!both (!chain-> [hyp1a ==> (x @ y in R) [in-def]])
2181               (!chain-> [hyp1a ==> (x = x') [pair-axioms]
2182                 ==> (x = a) [case1]]))
2183             (!chain [(x @ y in t ^1 a) ==> (x @ y in t & x = a) [IH]
2184               ==> (x @ y in R & x = a) [in-def]]))
2185             assume case2 := (x' /= a)
2186             (!chain-> [hyp ==> (x @ y in t ^1 a) [restrict1-def]
2187               ==> (x @ y in t & x = a) [IH]
2188               ==> (x @ y in R & x = a) [in-def]]));
2189       dir2 := (!chain [(x @ y in R & x = a) ==> (x @ y in R ^1 a) [restrict1-lemma]])}
2190   (!equiv dir1 dir2)
2191 }
2192
2193 declare restrict: (S, T) [(Set (Pair S T)) (Set S)] -> (Set (Pair S T)) [200 [alist->set alist->set]]
2194
2195 define ^ := restrict
2196 assert* restrict-def :=
2197 [(R restrict null = null)
2198   (R restrict h ++ t = R restrict1 h \/ R restrict t)]
2199
2200 conclude restrict-characterization :=
2201 (forall A R x y . x @ y in R restrict A <==> x @ y in R & x in A)
2202 by-induction restrict-characterization {
2203   (A as null) => pick-any R x y
2204     (!chain [(x @ y in R restrict A)
2205       <==> (x @ y in null) [restrict-def]
2206       <==> false [NC]
2207       <==> (x @ y in R & false) [prop-taut]
2208       <==> (x @ y in R & x in A) [NC]])
2209 | (A as (insert h t)) =>
2210   let {IH := (forall R x y . x @ y in R restrict t <==> x @ y in R & x in t)}
2211   pick-any R x y
2212     (!chain [(x @ y in R restrict A)
2213       <==> (x @ y in R ^1 h \/ R restrict t) [restrict-def]
2214       <==> (x @ y in R ^1 h | x @ y in R restrict t) [UC]
2215       <==> ((x @ y in R & x = h) | x @ y in R restrict t) [restrict1-characterization]
2216       <==> ((x @ y in R & x = h) | x @ y in R & x in t) [IH]
2217       <==> ((x @ y in R) & (x = h | x in t)) [prop-taut]
2218       <==> (x @ y in R & x in A) [in-def]])
2219 }
2220
2221 conclude restriction-theorem-1 :=
2222 (forall R A B . A subset B ==> R ^ A subset R ^ B)
2223 pick-any R A B
2224   assume (A subset B)
2225   (!subset-intro
2226     (!pair-converter
2227       pick-any x y
2228       (!chain [(x @ y in R ^ A)
2229         ==> (x @ y in R & x in A) [restrict-characterization]
2230         ==> (x @ y in R & x in B) [SC]
2231         ==> (x @ y in R ^ B) [restrict-characterization]]))
2232   )
2233 conclude restriction-theorem-2 :=
2234 (forall R A B . R ^ (A /\ B) = R ^ A /\ R ^ B)
2235 pick-any R A B
2236   (!set-identity-intro-direct
2237     (!pair-converter
2238       pick-any x y

```



```

2239      (!chain [(x @ y in R ^ (A /\ B))
2240        <==> (x @ y in R & x in A /\ B) [restrict-characterization]
2241        <==> (x @ y in R & x in A & x in B) [IC]
2242        <==> ((x @ y in R & x in A) & (x @ y in R & x in B)) [prop-taut]
2243        <==> (x @ y in R ^ A & x @ y in R ^ B) [restrict-characterization]
2244        <==> (x @ y in R ^ A /\ R ^ B) [IC]]))
2245
2246 conclude restriction-theorem-3 :=
2247   (forall R A B . R ^ (A /\ B) = R ^ A /\ R ^ B)
2248 pick-any R A B
2249   (!set-identity-intro-direct
2250     (!pair-converter
2251       pick-any x y
2252       (!chain [(x @ y in R ^ (A /\ B))
2253         <==> (x @ y in R & x in A /\ B) [restrict-characterization]
2254         <==> (x @ y in R & (x in A | x in B)) [UC]
2255         <==> ((x @ y in R & x in A) | (x @ y in R & x in B)) [prop-taut]
2256         <==> (x @ y in R ^ A | x @ y in R ^ B) [restrict-characterization]
2257         <==> (x @ y in R ^ A /\ R ^ B) [UC]]))
2258
2259 conclude restriction-theorem-4 :=
2260   (forall R A B . R ^ (A \ B) = R ^ A \ R ^ B)
2261 pick-any R A B
2262   (!set-identity-intro-direct
2263     (!pair-converter
2264       pick-any x y
2265       (!chain [(x @ y in R ^ (A \ B))
2266         <==> (x @ y in R & x in A \ B) [restrict-characterization]
2267         <==> (x @ y in R & (x in A & ~ x in B)) [DC]
2268         <==> ((x @ y in R & x in A) & ~ (x @ y in R & x in B)) [prop-taut]
2269         <==> (x @ y in R ^ A & ~ x @ y in R ^ B) [restrict-characterization]
2270         <==> (x @ y in R ^ A \ R ^ B) [DC]]))
2271
2272 conclude restriction-theorem-5 :=
2273   (forall R1 R2 A . (R1 o R2) ^ A = (R1 ^ A) o R2)
2274 pick-any R1 R2 A
2275   (!set-identity-intro-direct
2276     (!pair-converter
2277       pick-any x y
2278       (!chain [(x @ y in (R1 o R2) ^ A)
2279         <==> (x @ y in R1 o R2 & x in A) [restrict-characterization]
2280         <==> ((exists z . x @ z in R1 & z @ y in R2) & x in A) [o-characterization]
2281         <==> (exists z . x @ z in R1 & z @ y in R2 & x in A) [taut]
2282         <==> (exists z . (x @ z in R1 & x in A) & z @ y in R2) [prop-taut]
2283         <==> (exists z . x @ z in R1 ^ A & z @ y in R2) [restrict-characterization]
2284         <==> (x @ y in (R1 ^ A) o R2) [o-characterization]]))
2285
2286 declare image: (S, T) [(Set (Pair S T)) (Set S)] -> (Set T) [** 200 [alist->set alist->set]]
2287
2288 #define ** := image
2289
2290 assert* image-def := [(R ** A = range R ^ A)]
2291
2292 conclude image-characterization :=
2293   (forall R A y . y in R ** A <==> exists x . x @ y in R & x in A)
2294 pick-any R A y
2295   (!chain [(y in R ** A)
2296     <==> (y in range R ^ A) [image-def]
2297     <==> (exists x . x @ y in R ^ A) [range-characterization]
2298     <==> (exists x . x @ y in R & x in A) [restrict-characterization]])
2299
2300 conclude image-lemma :=
2301   (forall R A x y . x @ y in R & x in A ==> y in R ** A)
2302 pick-any R A x y
2303   (!chain [(x @ y in R & x in A)
2304     ==> (exists x . x @ y in R & x in A) [existence]
2305     ==> (y in R ** A) [image-characterization]])
2306
2307 conclude image-theorem-1 :=
2308   (forall R A B . R ** (A /\ B) = R ** A /\ R ** B)

```

```

2309 pick-any R A B
2310   (!set-identity-intro-direct
2311     pick-any y
2312     (!chain [(y in R ** (A \ B))
2313       <==> (exists x . x @ y in R & x in A \ B) [image-characterization]
2314       <==> (exists x . x @ y in R & (x in A | x in B)) [UC]
2315       <==> (exists x . (x @ y in R & x in A) | (x @ y in R & x in B)) [prop-taut]
2316       <==> ((exists x . x @ y in R & x in A) | (exists x . x @ y in R & x in B)) [taut]
2317       <==> (y in R ** A | y in R ** B) [image-characterization]
2318       <==> (y in R ** A \ R ** B) [UC]]))
2319
2320 conclude image-theorem-2 :=
2321   (forall R A B . R ** (A /\ B) subset R ** A /\ R ** B)
2322 pick-any R A B
2323   (!subset-intro
2324     pick-any y
2325     (!chain [(y in R ** (A /\ B))
2326       ==> (exists x . x @ y in R & x in A /\ B) [image-characterization]
2327       ==> (exists x . x @ y in R & x in A & x in B) [IC]
2328       ==> (exists x . (x @ y in R & x in A) & (x @ y in R & x in B)) [prop-taut]
2329       ==> ((exists x . x @ y in R & x in A) & (exists x . x @ y in R & x in B)) [taut]
2330       ==> (y in R ** A & y in R ** B) [image-characterization]
2331       ==> (y in R ** A /\ R ** B) [IC]]))
2332
2333
2334 conclude image-theorem-3 :=
2335   (forall R A B . R ** A \ R ** B subset R ** (A \ B))
2336 pick-any R A B
2337   (!subset-intro
2338     pick-any y
2339     (!chain [(y in R ** A \ R ** B)
2340       ==> (y in R ** A & ~ y in R ** B) [DC]
2341       ==> ((exists x . x @ y in R & x in A) & ~ (exists x . x @ y in R & x in B)) [image-characterization]
2342       ==> ((exists x . x @ y in R & x in A) & (forall x . x @ y in R ==> ~ x in B)) [taut]
2343       ==> (exists x . x @ y in R & x in A & ~ x in B) [taut]
2344       ==> (exists x . x @ y in R & x in A \ B) [DC]
2345       ==> (y in R ** (A \ B)) [image-characterization]]))
2346
2347 conclude image-theorem-4 :=
2348   (forall R A B . A subset B ==> R ** A subset R ** B)
2349 pick-any R A B
2350   assume hyp := (A subset B)
2351   (!subset-intro
2352     pick-any y
2353     (!chain [(y in R ** A)
2354       ==> (exists x . x @ y in R & x in A) [image-characterization]
2355       ==> (exists x . x @ y in R & x in B) [SC]
2356       ==> (y in R ** B) [image-characterization]]))
2357
2358 conclude image-theorem-5 :=
2359   (forall R A . R ** A = null <==> dom R /\ A = null)
2360 pick-any R A
2361   (!chain [(R ** A = null)
2362     <==> (forall y . ~ y in R ** A) [null-characterization-2]
2363     <==> (forall y . ~ exists x . x @ y in R & x in A) [image-characterization]
2364     <==> (forall x . ~ exists y . x @ y in R & x in A) [taut]
2365     <==> (forall x . ~ ((exists y . x @ y in R) & x in A)) [taut]
2366     <==> (forall x . ~ (x in dom R & x in A)) [dom-characterization]
2367     <==> (forall x . ~ (x in dom R /\ A)) [IC]
2368     <==> (dom R /\ A = null) [null-characterization-2]])
2369
2370 conclude image-theorem-6 :=
2371   (forall R A . dom R /\ A subset -- R ** R ** A)
2372 pick-any R A
2373   (!subset-intro
2374     pick-any x
2375     (!chain [(x in dom R /\ A)
2376       ==> (x in dom R & x in A) [IC]
2377       ==> ((exists y . x @ y in R) & x in A) [dom-characterization]
2378       ==> (exists y . x @ y in R & x @ y in R & x in A) [taut]

```

```

2379      ==> (exists y . x @ y in R & y in R ** A) [image-lemma]
2380      ==> (exists y . y @ x in -- R & y in R ** A) [converse-characterization]
2381      ==> (x in -- R ** R ** A) [image-characterization]]))
2382
2383 #(falsify (forall R A . dom R /\ A = -- R ** R ** A) 20)
2384
2385 conclude image-theorem-7 :=
2386   (forall R A B . (R ** A) /\ B subset R ** (A /\ -- R ** B))
2387 pick-any R A B
2388   (!subset-intro
2389     pick-any y
2390       (!chain [(y in (R ** A) /\ B)
2391         ==> (y in R ** A & y in B) [IC]
2392         ==> ((exists x . x @ y in R & x in A) & y in B) [image-characterization]
2393         ==> (exists x . x @ y in R & x in A & y in B) [taut]
2394         ==> (exists x . y @ x in -- R & x in A & x @ y in R & y in B) [converse-characterization augment]
2395         ==> (exists x . (y @ x in -- R & y in B) & x in A & x @ y in R) [prop-taut]
2396         ==> (exists x . x in -- R ** B & x in A & x @ y in R) [image-lemma]
2397         ==> (exists x . x @ y in R & (x in A & x in -- R ** B)) [prop-taut]
2398         ==> (exists x . x @ y in R & x in A /\ -- R ** B) [IC]
2399         ==> (y in R ** (A /\ -- R ** B)) [image-characterization]]))
2400
2401 define lemma := (close t /\ (x insert-in-all t) = null)
2402 define lemma2 := (close (forall y . y in t ==> ~ x in y) ==> t /\ (x insert-in-all t) = null)
2403
2404 declare card: (S) [(Set S)] -> N [[alist->set]]
2405
2406 define S := N.S
2407
2408 assert* card-def :=
2409   [(card null = zero)
2410     (card h ++ t = card t <== h in t)
2411     (card h ++ t = S card t <== ~ h in t)]
2412
2413 transform-output eval [nat->int]
2414
2415 define [< <=] := [N.< N.<=]
2416
2417 overload + N.+
2418
2419 define card-theorem-1 :=
2420   (card singleton _ = S zero)
2421
2422 conclude card-theorem-2 :=
2423   (forall A x . ~ x in A ==> card A < card x ++ A)
2424 pick-any A x
2425   assume hyp := (~ x in A)
2426     (!chain-> [true ==> (card A < S card A) [N.Less.<S]
2427       ==> (card A < card x ++ A) [card-def]])
2428
2429 (define vpf
2430   (method (goal premises)
2431     (!vprove-from goal premises [['poly true] ['subsorting false] ['max-time 100]])))
2432
2433 (define spf
2434   (method (goal premises)
2435     (!sprove-from goal premises [['poly true] ['subsorting false] ['max-time 100]])))
2436
2437 conclude minus-card-theorem :=
2438   (forall A x . x in A ==> card A = N.S card A - x)
2439 by-induction minus-card-theorem {
2440   (A as null: (Set 'S)) =>
2441     pick-any x
2442       (!chain [(x in A)
2443         ==> false [NC]
2444         ==> (card A = N.S card A - x) [prop-taut]])
2445 | (A as (insert h:'S t:(Set 'S))) =>
2446   let {IH := (forall x . x in t ==> card t = N.S card (t - x))}
2447     pick-any x:'S
2448       assume hyp := (x in A)

```

```

2449 (!two-cases
2450   assume case1 := (x = h)
2451   (!two-cases
2452     assume (h in t)
2453     let {_ := (!chain-> [(h in t) ==> (x in t) [case1]])}
2454     (!chain [(card A)
2455               = (card t) [card-def]
2456               = (N.S (card t - x)) [IH]
2457               = (N.S (card A - x)) [remove-def]])
2458     assume (~ h in t)
2459     let {_ := (!chain-> [(~ h in t) ==> (~ x in t) [case1]])}
2460     (!combine-equations
2461       (!chain [(card A) = (N.S card t)])
2462       (!chain [(N.S card (A - x))
2463                 = (N.S card (t - x))
2464                 = (N.S card t)])))
2465     assume case2 := (x != h)
2466     let {_ := (!chain-> [(x in A)
2467                           ==> (x = h | x in t) [in-def]
2468                           ==> (x in t) [(dsyl with case2)])]}
2469     (!two-cases
2470       assume (h in t)
2471       let {_ := (!chain-> [(h in t)
2472                             ==> (h in t & x != h) [augment]
2473                             ==> (h in t - x) [remove-corollary-3]])}
2474       (!chain [(card A)
2475                 = (card t) [card-def]
2476                 = (N.S card (t - x)) [IH]
2477                 = (N.S card h ++ (t - x)) [card-def]
2478                 = (N.S card (A - x)) [remove-def]])
2479       assume (~ h in t)
2480       let {_ := (!chain-> [(~ h in t) ==> (~ h in t - x) [remove-corollary-4]])}
2481       (!chain-> [(card t) = (N.S card t - x) [IH]
2482                 ==> (N.S card t = N.S N.S card t - x)
2483                 ==> (card A = N.S N.S card t - x) [card-def]
2484                 ==> (card A = N.S card h ++ (t - x)) [card-def]
2485                 ==> (card A = N.S card A - x) [remove-def])])
2486   }
2487
2488 define subset-card-theorem :=
2489   (forall A B . A subset B ==> card A <= card B)
2490
2491
2492 by-induction subset-card-theorem {
2493   null => pick-any B:(Set 'S)
2494     assume hyp := (null subset B)
2495     (!chain-> [true ==> (zero <= card B) [N.Less=.zero<=]
2496               ==> (card null:(Set 'S) <= card B) [card-def]])
2497   | (A as (insert h:'S t:(Set 'S))) =>
2498     let {IH := (forall B . t subset B ==> card t <= card B)}
2499     pick-any B:(Set 'S)
2500     assume hyp := (A subset B)
2501     (!two-cases
2502       assume case1 := (in h t)
2503       (!chain-> [hyp ==> (t subset B) [subset-lemma-2]
2504                 ==> (card t <= card B) [IH]
2505                 ==> (card A <= card B) [card-def]])
2506       assume case2 := (~ in h t)
2507       let {t-sub-B := (!chain-> [hyp ==> (t subset B) [subset-lemma-2]]);
2508           _ := (!chain-> [true
2509                         ==> (in h A) [in-lemma-1]
2510                         ==> (in h B) [SC]])}
2511       (!chain-> [t-sub-B ==> (t subset B & case2) [augment]
2512                 ==> (t subset B - h) [remove-corollary-5]
2513                 ==> (card t <= card B - h) [IH]
2514                 ==> (S card t <= S card B - h)
2515                 ==> (S card t <= card B) [minus-card-theorem]
2516                 ==> (card A <= card B) [card-def])])
2517   }
2518

```

```

2519 conclude proper-subset-card-theorem :=
2520   (forall A B . A proper-subset B ==> card A < card B)
2521 pick-any A B
2522   assume hyp := (A proper-subset B)
2523   pick-witness x for (!chain-> [hyp ==> (A subset B & exists x . x in B & ~ x in A) [PSC]
2524     ==> (exists x . x in B & ~ x in A) [right-and]])
2525     let {L1 := (!chain-> [hyp ==> (A subset B) [PSC]
2526       ==> (x ++ A subset B) [subset-lemma-1]
2527       ==> (card x ++ A <= card B) [subset-card-theorem]]];
2528       L2 := (!chain-> [(~ x in A) ==> (card A < card x ++ A) [card-theorem-2]])}
2529     (!chain-> [L1 ==> (L1 & L2) [augment]
2530       ==> (card A < card B) [N.Less=.transitive1]])
2531
2532 conclude intersection-card-theorem-1 :=
2533   (forall A B . card A /\ B <= card A)
2534 pick-any A B
2535   (!chain-> [true ==> (A /\ B subset A) [intersection-subset-theorem]
2536     ==> (card A /\ B <= card A) [subset-card-theorem]])
2537
2538 conclude intersection-card-theorem-2 :=
2539   (forall A B . card A /\ B <= card B)
2540 pick-any A B
2541   (!chain-> [true ==> (A /\ B subset B) [intersection-subset-theorem-2]
2542     ==> (card A /\ B <= card B) [subset-card-theorem]])
2543
2544 conclude intersection-card-theorem-3 :=
2545   (forall A B x . ~ x in A & x in B ==> card (x ++ A) /\ B = N.S card A /\ B)
2546 pick-any A B x
2547   assume hyp := (~ x in A & x in B)
2548   let {_ := (!chain-> [(~ x in A) ==> (~ x in A /\ B) [intersection-lemma-2]])}
2549   (!chain [(card (x ++ A) /\ B)
2550     = (card x ++ (A /\ B)) [intersection-def]
2551     = (S card A /\ B) [card-def]])
2552
2553 # by-induction card-lemma-1 {
2554 #   (A as (insert h t)) =>
2555 #     let {_ := (mark 'A)}
2556 #     (!vpf (forall B x . ~ x in A & x in B ==> card (x ++ A) /\ B = N.S card A /\ B) (ab))
2557 # | (A as Set.null) => let {_ := (mark 'B)} (!dhalt)
2558 # }
2559
2560 define card-lemma-2 :=
2561   (forall A B . card A \ B = ((card A) + (card B)) N.- (card A /\ B))
2562
2563 overload - N.-
2564
2565 conclude num-lemma :=
2566   (forall x y z . (x + y) - z = (S x + y) - S z)
2567 pick-any x:N y:N z:N
2568   (!chain-> [(S x + y) - S z
2569     = (S (x + y) - S z) [N.Plus.left-nonzero]
2570     = ((x + y) - z) [N.Minus.axioms]
2571     ==> ((x + y) - z = (S x + y) - S z) [sym]])
2572
2573 conclude lemma-p1 :=
2574   (forall A B x . ~ x in A and x in B ==> card (x ++ A) /\ B = S card A /\ B)
2575 pick-any A B x
2576   assume hyp := (~ x in A & x in B)
2577   let {_ := (!chain-> [(~ x in A)
2578     ==> (~ x in A /\ B) [intersection-lemma-2]])}
2579   (!chain [(card x ++ A /\ B)
2580     = (card x ++ (A /\ B)) [intersection-def]
2581     = (S card A /\ B) [card-def]])
2582
2583 conclude lemma-p2 :=
2584   (forall A B x . ~ x in A & ~ x in B ==> A /\ B = (x ++ A) /\ B)
2585 pick-any A B x
2586   assume hyp := (~ x in A & ~ x in B)
2587   (!set-identity-intro-direct
2588     pick-any y

```

```

2589      (!equiv assume hyp1 := (y in A /\ B)
2590      let {L1 := (!chain-> [hyp1 ==> (y in A) [IC]
2591      ==> (y = x | y in A) [alternate]
2592      ==> (y in x ++ A) [in-def]]);
2593      L2 := (!chain-> [hyp1 ==> (y in B) [IC]])}
2594      (!chain-> [L1
2595      ==> (L1 & L2) [augment]
2596      ==> (y in (x ++ A) /\ B) [IC]])
2597      assume hyp2 := (y in (x ++ A) /\ B)
2598      let {L1 := (!chain-> [hyp2 ==> (y in B) [IC]]);
2599      L2 := (!by-contradiction (y != x)
2600      assume (y = x)
2601      (!chain-> [(y in B) ==> (x in B) [(y = x)]
2602      ==> (x in B & ~ x in B) [augment]
2603      ==> false [prop-taut]]))}
2604      (!chain-> [hyp2 ==> (y in x ++ A) [IC]
2605      ==> (y = x | y in A) [in-def]
2606      ==> ((y = x | y in A) & y != x) [augment]
2607      ==> (((y = x) & (y != x)) | (y in A & y != x)) [prop-taut]
2608      ==> (false | y in A & y != x) [prop-taut]
2609      ==> (y in A) [prop-taut]
2610      ==> (y in A & y in B) [augment]
2611      ==> (y in A /\ B) [IC]]))
2612
2613      # by-induction card-lemma-2 {
2614      # null => (!vpf (forall B . card null \/ B = (card null) + (card B) N.- card null /\ B) (ab))
2615      # | (A as (insert h t)) =>
2616      #   let {_ := (mark 'A)}
2617      #   (!vpf (forall B . card A \/ B = (card A) + (card B) N.- card A /\ B) (ab))
2618
2619      # }
2620
2621      #(falsify card-lemma-1 10)
2622
2623      conclude union-lemma-2 :=
2624      (forall A B x . x ++ (A \/ B) = A \/ x ++ B)
2625      pick-any A B x
2626      (!chain [(x ++ (A \/ B))
2627      = (x ++ (B \/ A)) [union-commutes]
2628      = ((x ++ B) \/ A) [union-def]
2629      = (A \/ (x ++ B)) [union-commutes]])
2630
2631      conclude union-subset-lemma-1 := (forall A B . A subset A \/ B)
2632      pick-any A B
2633      (!subset-intro
2634      pick-any x
2635      (!chain [(x in A) ==> (x in A \/ B) [UC]]))
2636
2637      conclude union-subset-lemma-2 := (forall A B . B subset A \/ B)
2638      pick-any A B
2639      (!subset-intro
2640      pick-any x
2641      (!chain [(x in B) ==> (x in A \/ B) [UC]]))
2642
2643      conclude minus-lemma :=
2644      (forall x y . y <= x ==> S (x - y) = (S x) - y)
2645      pick-any x:N y:N
2646      assume (y <= x)
2647      let {_ := (!chain-> [(y <= x) ==> (y <= S x) [N.Less=.S2]])}
2648      (!chain-> [(S x) = (S x)
2649      ==> (S ((x - y) + y) = S x) [N.Minus.Plus-Cancel]
2650      ==> (S (x - y) + y = S x) [N.Plus.left-nonzero]
2651      ==> (S (x - y) + y = (S x - y) + y) [N.Minus.Plus-Cancel]
2652      ==> (S (x - y) = S x - y) [N.Plus.=-cancellation]])
2653
2654      conclude union-card :=
2655      (forall A B . card A \/ B = ((card A) + (card B)) - card A /\ B)
2656      by-induction union-card {
2657      null => pick-any B
2658      let {ns := null:(Set 'S)}

```

```

2659         (!chain [(card ns \ / B)
2660                 = (card B)                                [union-def]
2661                 = ((card B) - zero)                       [N.Minus.axioms]
2662                 = ((card B) - (card ns))                  [card-def]
2663                 = ((card B) - card ns \ / B)              [intersection-def]
2664                 = ((zero + card B) - card ns \ / B)       [N.Plus.left-zero]
2665                 = (((card ns) + (card B)) - card ns \ / B) [card-def]])
2666 | (A as (insert h t:(Set 'S))) =>
2667   let {IH := (forall B . card t \ / B = ((card t) + (card B)) - card t \ / B)}
2668   pick-any B:(Set 'S)
2669     (!two-cases
2670       assume case1 := (h in t)
2671       let {_ := (!chain-> [(h in t) ==> (h in t \ / B) [UC]]];
2672           L1 := (!chain [(card A \ / B)
2673                        = (card h ++ (t \ / B))              [union-def]
2674                        = (card t \ / B)                      [card-def]
2675                        = (((card t) + (card B)) - (card t \ / B)) [IH]
2676                        = (((card A) + (card B)) - (card t \ / B)) [card-def]])}
2677       (!two-cases
2678         assume (h in B)
2679         let {_ := (!both (h in B) (h in t))}
2680         (!chain [(card A \ / B)
2681                  = (((card A) + (card B)) - (card t \ / B)) [L1]
2682                  = (((card A) + (card B)) - (card A \ / B)) [intersection-lemma-1]])
2683         assume (~ h in B)
2684         (!chain [(card A \ / B)
2685                  = (((card A) + (card B)) - (card t \ / B)) [L1]
2686                  = (((card A) + (card B)) - (card A \ / B)) [intersection-def]])
2687         assume case2 := (~ h in t)
2688         (!two-cases
2689           assume (h in B)
2690           let {_ := (!chain-> [(h in B) ==> (h in t \ / B) [UC]]];
2691               _ := (!chain-> [(~ h in t) ==> (~ h in t \ / B) [IC]])}
2692           (!chain [(card A \ / B)
2693                    = (card h ++ (t \ / B))                  [union-def]
2694                    = (card t \ / B)                        [card-def]
2695                    = (((card t) + (card B)) - (card t \ / B)) [IH]
2696                    = (((S card t) + (card B)) - (S (card t \ / B))) [num-lemma]
2697                    = (((card A) + (card B)) - (S (card t \ / B))) [card-def]
2698                    = (((card A) + (card B)) - (S (card t \ / B))) [card-def]
2699                    = (((card A) + (card B)) - (card h ++ (t \ / B))) [card-def]
2700                    = (((card A) + (card B)) - (card A \ / B)) [intersection-def]])
2701           assume (~ h in B)
2702           let {_ := (!chain-> [(~ h in t)
2703                               ==> (~ h in t & ~ h in B) [augment]
2704                               ==> (~ (h in t | h in B)) [dm]
2705                               ==> (~ h in t \ / B) [UC]]];
2706               _ := (!chain-> [true
2707                              ==> (card t \ / B <= card t) [intersection-card-theorem-1]
2708                              ==> (card t \ / B <= (card t) + (card B)) [N.Less=.Plus-k1]])}
2709           (!chain [(card A \ / B)
2710                    = (card h ++ (t \ / B)) [union-def]
2711                    = (S card t \ / B) [card-def]
2712                    = (S (((card t) + card B) - card t \ / B)) [IH]
2713                    = ((S ((card t) + (card B))) - (card t \ / B)) [minus-lemma]
2714                    = ((S ((card t) + (card B))) - (card A \ / B)) [lemma-p2]
2715                    = (((S card t) + card B) - (card A \ / B)) [N.Plus.left-nonzero]
2716                    = (((card A) + card B) - (card A \ / B)) [card-def]])}
2717   }
2718
2719   conclude diff-card-lemma :=
2720     (forall A B . card A = (card A \ B) + (card A \ / B))
2721   pick-any A B
2722     (!chain-> [true ==> (A = (A \ B) \ / (A \ / B)) [diff-theorem-12]
2723               ==> (card A = card (A \ B) \ / (A \ / B))
2724               ==> (card A = ((card A \ B) + (card A \ / B)) - (card (A \ B) \ / (A \ / B))) [union-card]
2725               ==> (card A = ((card A \ B) + (card A \ / B)) - (card null)) [diff-theorem-13]
2726               ==> (card A = ((card A \ B) + (card A \ / B)) - zero) [card-def]
2727               ==> (card A = (card A \ B) + card A \ / B) [N.Minus.axioms]])
2728

```

```
2729 conclude diff-card-theorem :=
2730   (forall A B . card A \ B = (card A) - card A /\ B)
2731 pick-any A B
2732   (!chain-> [true ==> (card A = (card A \ B) + card A /\ B) [diff-card-lemma]
2733     ==> ((card A \ B) + card A /\ B = card A) [sym]
2734     ==> (card A \ B = (card A) - card A /\ B) [N.Minus.Plus-Minus-properties]])
2735
2736 } # close module Set
```