

Utils.v

```
1 Require Import Coq.Classes.EquivDec.
2 Require Import Coq.Relations.Relation_Operators.
3 Require Import Coq.Lists.List.
4 Require Import Coq.omega.Omega.
5 Require Import Datatypes.
6 Require Import Omega.
7
8 Import ListNotations.
9 Import EqNotations.
10
11 Lemma in_tuple_snd {A} {eq_dec : EqDec A eq} : forall (l : list (A*A)) a, {b &
  ↪ In (a, b) l} + {forall b, ~In (a,b) l}.
12 Proof.
13   intros.
14   pose proof prod_eqdec eq_dec eq_dec as tuple_eq_dec.
15   induction l.
16   - right. auto.
17   - destruct IHl.
18     + destruct s. left. intros. exists x. constructor 2. assumption.
19     + destruct a0. destruct (a = a0).
20       * rewrite e. left. exists a1. constructor. auto.
21       * right. intros. intros F. inversion F.
22         ** apply c. injection H. intros. subst. reflexivity.
23         ** pose proof n b. contradiction.
24 Qed.
25
26
27 Lemma ge_0_eq : forall m, 0 ≥ m → 0 = m. Proof. intros. omega. Qed.
28
29 Lemma Odd_plus_Even_is_Odd : forall n m, Nat.Odd n → Nat.Even m → Nat.Odd (n +
  ↪ m).
30 Proof.
31   intros n m [n' Hodd] [m' Heven].
32   unfold Nat.Odd.
33   exists (n' + m').
34   omega.
35 Qed.
36
37
38 Lemma count_occ_split {A} : forall eq_dec (x : A) l1 l2, count_occ eq_dec (l1 ++
  ↪ l2) x = count_occ eq_dec l1 x + count_occ eq_dec l2 x.
39 Proof.
40   induction l1.
41   - reflexivity.
42   - intros. rewrite ← app_comm_cons. simpl. destruct (eq_dec a x) eqn:Heq.
43     + rewrite IHl1. firstorder.
44     + apply IHl1.
45 Qed.
46
47 Lemma count_occ_head_last {A} : forall eq_dec (x: A) y l, count_occ eq_dec (y ::
  ↪ l) x = count_occ eq_dec (l ++ [y]) x.
48 Proof.
49   intros.
50   rewrite count_occ_split.
```

```

51   simpl.
52   destruct (eq_dec y x) eqn:Heq.
53   - firstorder.
54   - firstorder.
55 Qed.
56
57 Lemma count_occ_last_neq {A} : forall eq_dec (x: A) y l, y  $\diamond$  x  $\rightarrow$  count_occ
58    $\hookrightarrow$  eq_dec (l ++ [y]) x = count_occ eq_dec l x.
59 Proof.
60   intros.
61   rewrite  $\leftarrow$  count_occ_head_last.
62   apply count_occ_cons_neq.
63   assumption.
64 Qed.
65
66 Lemma In_head_set {A} {eqdec : EqDec A eq} : forall l (a x : A), In x (a :: l)
67    $\hookrightarrow \rightarrow$  {a = x} + {In x l}.
68 Proof.
69   intros.
70   destruct (a = x).
71   - left. assumption.
72   - right. destruct H.
73     + contradiction.
74     + assumption.
75 Qed.
76
77 Lemma In_app_sumbool {A} {eqdec : EqDec A eq}: forall (a : A) l1 l2, In a (l1
78    $\hookrightarrow$  ++ l2)  $\rightarrow$  {In a l1} + {In a l2}.
79 Proof.
80   intros. apply in_app_or in H. destruct (in_dec eqdec a l1).
81   left. assumption. right. destruct H. contradiction. assumption.
82 Qed.
83
84 Lemma in_map_cons {A} : forall (x : A) xs ys, In (x::xs) (map (cons x) ys)  $\rightarrow$  In
85    $\hookrightarrow$  xs ys.
86 Proof.
87   induction ys.
88   - auto.
89   - simpl in *. intros. destruct H.
90     + left. inversion H. reflexivity.
91     + right. apply IHys. assumption.
92 Qed.
93
94 Lemma in_map_cons_not {A} : forall (x y : A) xs ys, x  $\diamond$  y  $\rightarrow$   $\sim$ (In (x::xs) (map
95    $\hookrightarrow$  (cons y) ys)).
96 Proof.
97   induction ys.
98   - intros Heq F. exact F.
99   - intros Heq F. destruct F.
100     + inversion H. symmetry in H1. contradiction.
101     + apply IHys in Heq. contradiction.
102 Qed.
103
104 Lemma in_map_cons_iff {A} : forall (x : A) xs ys, In (x::xs) (map (cons x) ys)
105    $\hookrightarrow \rhd$  In xs ys.
106 Proof.

```

```

101   intros. split.
102   - apply in_map_cons.
103   - intros. induction ys.
104     + inversion H.
105     + simpl. destruct H.
106       * left. subst. reflexivity.
107       * right. apply IHys. apply H.
108 Qed.
109
110 Lemma some_eq : forall (T : Type) (a b : T), a = b → Some a = Some b.
111 Proof. intros. split.
112   - intros Heq. subst. reflexivity.
113   - intros Heq. inversion Heq. reflexivity.
114 Qed.
115
116 Inductive Forall_T {A : Type} (P : A → Type) : list A → Type :=
117 | Forall_T_nil : Forall_T P nil
118 | Forall_T_cons : forall (x : A) (l : list A), P x → Forall_T P l → Forall_T P
119   ↪ (x :: l)
120 .
121
122 Lemma Forall_T_forall {A P} {eqdec : EqDec A eq} {l: list A} : Forall_T P l →
123   ↪ (forall x : A, In x l → P x).
124 Proof.
125   induction 1; intros.
126   - inversion H.
127   - destruct (x = x0).
128     + rewrite e in p. assumption.
129     + apply IHX.
130     inversion H.
131     * pose proof (Equivalence.equiv_reflexive_obligation_1 _ x0).
132       ↪ contradiction.
133     * assumption.
134 Qed.
135
136 Inductive Forall2_T {A B : Type} (R : A → B → Type) : list A → list B → Type
137   ↪ :=
138 | Forall2_T_nil : Forall2_T R [] []
139 | Forall2_T_cons : forall (x : A) (y : B) (l : list A) (l' : list B),
140   R x y → Forall2_T R l l' → Forall2_T R (x :: l) (y :: l').
141
142 Inductive Forall2_T_idx {A B : Type} (R : nat → A → B → Type) : nat → list A
143   ↪ → list B → Type :=
144 | Forall2_T_idx_nil : Forall2_T_idx R 0 [] []
145 | Forall2_T_idx_cons : forall (x : A) (y : B) (l : list A) (l' : list B) (n :
146   ↪ nat),
147   R n x y → Forall2_T_idx R n l l' → Forall2_T_idx R (S n) (x
148   ↪ :: l) (y :: l').
149
150 Ltac dec_eq_one :=
151   match goal with
152   | H : ?x ≠ ?x ⊢ _ ⇒ elimtype False; apply H; reflexivity
153   | H : ?x ≡ ?y ⊢ _ ⇒ red in H; subst
154   | ⊢ { ?x ≡ ?x } + { _ } ⇒ left; reflexivity
155   | ⊢ { _ } + { ?x ≠ ?x } ⇒ left; f_equal
156   | ⊢ { ?x ≡ ?y } + { _ } ⇒ right; let H := fresh in intro H; red in H;

```

```

150         (injection H || discriminate); intros; subst
151     end.
152
153 Ltac dec_eq := try solve [ repeat dec_eq_one ];
154     repeat match goal with H : _  $\equiv$  _  $\vdash$  _  $\Rightarrow$  red in H; subst end.
155
156 Ltac isfalse := intros F; inversion F; try contradiction.
157
158
159 Definition kleisli_option {A B C : Type} (f : A  $\rightarrow$  (option B)) (g : B  $\rightarrow$  option
     $\hookrightarrow$  C) x :=
160     match f x with
161     | None  $\Rightarrow$  None
162     | Some y  $\Rightarrow$  g(y)
163     end.
164
165 Definition fmap_option {A B : Type} (f : A  $\rightarrow$  B) (a : option A) : option B :=
166     match a with
167     | None  $\Rightarrow$  None
168     | Some x  $\Rightarrow$  Some (f x)
169     end.
170
171 Definition bind_option {A B : Type} (m : option A) (f : A  $\rightarrow$  option B) : option
     $\hookrightarrow$  B :=
172     match m with
173     | None  $\Rightarrow$  None
174     | Some x  $\Rightarrow$  f x
175     end.
176
177 Notation "A  $\Rightarrow$  B" := (kleisli_option A B) (at level 50, left associativity).
178 Notation "m  $\gg$  f" := (bind_option m f) (at level 50, left associativity).
179
180 Lemma kleisli_to_bind_option {A B C : Type} :
181     forall (m : A  $\rightarrow$  option B) (n : B  $\rightarrow$  option C) x,
182         (m  $\Rightarrow$  n) x = m x  $\gg$  (fun y  $\Rightarrow$  n y).
183 Proof.
184     intros. unfold kleisli_option. destruct (m x); reflexivity.
185 Qed.
186
187 Lemma monad_law_option_1 {A B : Type} : forall (f : A  $\rightarrow$  option B) a , Some a  $\gg$ 
     $\hookrightarrow$  f = f a.
188 Proof. reflexivity. Qed.
189
190 Lemma monad_law_option_2 {A : Type} : forall (m : option A) , m  $\gg$  Some = m.
191 Proof.
192     destruct m; reflexivity.
193 Qed.
194
195 Lemma monad_law_option_3 {A B C : Type} :
196     forall m (f : A  $\rightarrow$  option B) (g : B  $\rightarrow$  option C),
197         (m  $\gg$  f)  $\gg$  g = m  $\gg$  (fun x  $\Rightarrow$  f x  $\gg$  g).
198 Proof.
199     destruct m; reflexivity.
200 Qed.
201

```

```

202 Lemma split_rev : forall A (ts1:list A) ts2, rev (ts1 ++ ts2) = rev ts2 ++ rev
    ↪ ts1.
203 Proof.
204   induction ts1.
205   - intros. simpl. rewrite app_nil_r. reflexivity.
206   - intros. simpl. rewrite IHts1. rewrite app_assoc. reflexivity.
207 Qed.
208
209 Lemma rev_head_last : forall A (ts : list A) a, rev (a :: ts) = rev ts ++ [a].
210 Proof.
211   intros.
212   assert (a :: ts = [a] ++ ts). reflexivity.
213   rewrite H. apply split_rev.
214 Qed.
215
216
217 Inductive Forall2_rev {A B} R : list A → list B → Prop :=
218 | Forall2_rev_nil : Forall2_rev R [] []
219 | Forall2_rev_cons : forall x y l l',
220     R x y → Forall2_rev R l l' → Forall2_rev R (rev (x :: l)) (rev
    ↪ (y :: l')).
221
222 Hint Constructors Forall2_rev.
223
224 Lemma Forall2_head : forall A B R (l1 : list A) (l2 : list B) a b ,
225   Forall2 R (a :: l1) (b :: l2) → R a b.
226 Proof.
227   intros. inversion H. assumption.
228 Qed.
229
230 Lemma Forall2_head_T : forall A B R (l1 : list A) (l2 : list B) a b ,
231   Forall2_T R (a :: l1) (b :: l2) → R a b.
232 Proof.
233   intros. inversion X. assumption.
234 Qed.
235
236 Lemma app_eq_length_eq : forall A (l1 l2 : list A),
237   l1 = l2 → length l1 = length l2.
238 Proof.
239   intros. subst. reflexivity.
240 Qed.
241
242 Lemma app_singl_eq_singl_nil : forall A (l: list A) a b,
243   [a] = l ++ [b] → l = [] /\ a = b.
244 Proof.
245   intros.
246   induction l.
247   - simpl in H. inversion H. auto.
248   - apply app_eq_length_eq in H. simpl in H. inversion H.
249     rewrite app_length in H1. simpl in H1. rewrite ← plus_n_Sm in H1. inversion
    ↪ H1.
250 Qed.
251
252 Lemma l1_le_length_split : forall A (l: list A),
253   1 ≤ length l → exists a l', l = a::l'.
254 Proof.

```

```

255   destruct l.
256   - isfalse.
257   - intros. exists a. exists l. reflexivity.
258 Qed.
259
260 Lemma Sn_impl_1_lt_m : forall m n, S n = m  $\rightarrow$  1  $\leq$  m.
261 Proof.
262   intros. omega.
263 Qed.
264
265 Lemma Forall2_idx {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2 R ns ms
266    $\hookrightarrow \rightarrow$  forall i, nth_error ns i = Some n  $\rightarrow$ 
267   nth_error ms i = Some m  $\rightarrow$  R n m.
268 Proof.
269   induction 1.
270   - intros. destruct i; inversion H.
271   - intros. destruct i.
272     + simpl in *. inversion H1. inversion H2. subst. assumption.
273     + simpl in *. eapply IHForall2.
274       { apply H1. }
275       { apply H2. }
276 Qed.
277
278 Lemma Forall2_idx_T {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2_T R
279    $\hookrightarrow$  ns ms  $\rightarrow$  forall i, nth_error ns i = Some n  $\rightarrow$ 
280   nth_error ms i = Some m  $\rightarrow$  R n m.
281 Proof.
282   induction 1.
283   - intros. destruct i; inversion H.
284   - intros. destruct i.
285     + simpl in *. inversion H. inversion H0. subst. assumption.
286     + simpl in *. eapply IHX.
287       { apply H. }
288       { apply H0. }
289 Qed.
290
291 Lemma Forall2_length : forall A B R (l : list A) (l' : list B), Forall2 R l l'
292    $\hookrightarrow \rightarrow$  length l = length l'.
293 Proof.
294   induction 1.
295   - reflexivity.
296   - simpl. rewrite IHForall2. reflexivity.
297 Qed.
298
299 Lemma Forall2_length_T : forall A B R (l : list A) (l' : list B), Forall2_T R l
300    $\hookrightarrow$  l'  $\rightarrow$  length l = length l'.
301 Proof.
302   induction 1.
303   - reflexivity.
304   - simpl. rewrite IHX. reflexivity.
305 Qed.
306
307 Lemma nth_error_last_length {A: Type} : forall ls (x : A), nth_error (ls ++ [x])
308    $\hookrightarrow$  (length ls) = Some x.
309 Proof.
310   intros.

```

```

306   induction ls.
307   - reflexivity.
308   - simpl. assumption.
309 Qed.
310
311 Lemma Forall2_last_length {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2
  ⇨ R ns ms →
312   nth_error ns ((length ns) - 1) = Some n →
313   nth_error ms ((length ms) - 1) = Some m → R n m.
314 Proof.
315   intros.
316   apply Forall2_idx with ns ms (length ns - 1); try assumption.
317   - apply Forall2_length in H. rewrite H. assumption.
318 Qed.
319
320 Lemma Forall2_last_length_T {A B: Type} : forall (n : A) ns (m : B) ms R ,
  ⇨ Forall2_T R ns ms →
321   nth_error ns ((length ns) - 1) = Some n →
322   nth_error ms ((length ms) - 1) = Some m → R n m.
323 Proof.
324   intros.
325   apply Forall2_idx_T with ns ms (length ns - 1); try assumption.
326   - apply Forall2_length_T in X. rewrite X. assumption.
327 Qed.
328
329 Lemma Forall2_last {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2 R (ns
  ⇨ ++ [n]) (ms ++ [m]) →
330   R n m.
331 Proof.
332   intros.
333   apply Forall2_last_length with (ns ++ [n]) (ms ++ [m]).
334   - assumption.
335   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0.
336     apply nth_error_last_length.
337   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0.
338     apply nth_error_last_length.
339 Qed.
340
341 Lemma Forall2_last_T {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2_T R
  ⇨ (ns ++ [n]) (ms ++ [m]) →
342   R n m.
343 Proof.
344   intros.
345   apply Forall2_last_length_T with (ns ++ [n]) (ms ++ [m]).
346   - assumption.
347   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0.
348     apply nth_error_last_length.
349   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0.
350     apply nth_error_last_length.
351 Qed.
352
353 Lemma Forall2_firstn {A B: Type} : forall ns ms (R : A → B → Prop), Forall2 R
  ⇨ ns ms →
354   forall n ans ams, firstn n ns = ans → firstn n ms = ams → Forall2 R ans ams.
355 Proof.
356   induction 1.

```

```

357 - intros. destruct n; simpl in *; subst; constructor.
358 - intros. destruct n.
359 + subst. constructor.
360 + simpl in *. destruct ans, ams.
361   { constructor. }
362   { inversion H1. }
363   { inversion H2. }
364   { inversion H1. inversion H2. constructor.
365     - subst. assumption.
366     - eapply IHForall2; try reflexivity. }
367 Qed.
368
369 Lemma Forall2_firstn_T {A B: Type} : forall ns ms (R : A → B → Type),
  ⇨ Forall2_T R ns ms →
370   forall n ans ams, firstn n ns = ans → firstn n ms = ams → Forall2_T R ans
  ⇨ ams.
371 Proof.
372   induction 1.
373   - intros. destruct n; simpl in *; subst; constructor.
374   - intros. destruct n.
375   + subst. constructor.
376   + simpl in *. destruct ans, ams.
377     { constructor. }
378     { inversion H. }
379     { inversion H0. }
380     { inversion H. inversion H0. constructor.
381       - subst. assumption.
382       - eapply IHX; try reflexivity. }
383 Qed.
384
385 Lemma firstn_init_length {A : Type} : forall init (x : A), firstn (length init)
  ⇨ (init ++ [x]) = init.
386 Proof.
387   intros.
388   induction init.
389   - reflexivity.
390   - simpl. rewrite IHinit. reflexivity.
391 Qed.
392
393 Lemma Forall2_init_length {A B: Type} : forall (ans : list A) ns (ams : list B)
  ⇨ ms R , Forall2 R ns ms →
394   firstn ((length ms) - 1) ms = ams →
395   firstn ((length ns) - 1) ns = ans → Forall2 R ans ams.
396 Proof.
397   intros.
398   eapply Forall2_firstn.
399   - apply H.
400   - apply H1.
401   - assert (length ns = length ms).
402     { eapply Forall2_length. apply H. } rewrite H2. apply H0.
403 Qed.
404
405 Lemma Forall2_init_length_T {A B: Type} : forall (ans : list A) ns (ams : list
  ⇨ B) ms R , Forall2_T R ns ms →
406   firstn ((length ms) - 1) ms = ams →
407   firstn ((length ns) - 1) ns = ans → Forall2_T R ans ams.

```



```

408 Proof.
409   intros.
410   eapply Forall2_firstn_T.
411   - apply X.
412   - apply H0.
413   - assert (length ns = length ms).
414     { eapply Forall2_length_T. apply X. } rewrite H1. apply H.
415 Qed.
416
417 Lemma Forall2_init {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2 R (ns
  ↪ ++ [n]) (ms ++ [m]) →
418   Forall2 R ns ms.
419 Proof.
420   intros.
421   eapply Forall2_init_length.
422   - apply H.
423   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0. apply
     ↪ firstn_init_length.
424   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0. apply
     ↪ firstn_init_length.
425 Qed.
426
427 Lemma Forall2_init_T {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2_T R
  ↪ (ns ++ [n]) (ms ++ [m]) →
428   Forall2_T R ns ms.
429 Proof.
430   intros.
431   eapply Forall2_init_length_T.
432   - apply X.
433   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0. apply
     ↪ firstn_init_length.
434   - rewrite app_length. rewrite plus_comm. simpl. rewrite ← minus_n_0. apply
     ↪ firstn_init_length.
435 Qed.
436
437 Lemma Forall2_split_last {A B: Type} : forall (n : A) ns (m : B) ms R , Forall2
  ↪ R (ns ++ [n]) (ms ++ [m]) >→
438   Forall2 R ns ms /\ R n m.
439 Proof.
440   intros.
441   split.
442   - split.
443     + eapply Forall2_init. apply H.
444     + eapply Forall2_last. apply H.
445   - intros. destruct H.
446     induction H.
447     + simpl. constructor.
448       * assumption.
449       * constructor.
450     + simpl. constructor; assumption.
451 Qed.
452
453 Lemma Forall2_split_last_T {A B: Type} : forall (n : A) ns (m : B) ms R ,
  ↪ Forall2_T R (ns ++ [n]) (ms ++ [m]) →
454   prod (Forall2_T R ns ms) (R n m).
455 Proof.

```

```

456   intros.
457   split.
458     + eapply Forall2_init_T. apply X.
459     + eapply Forall2_last_T. apply X.
460 Qed.
461
462 Lemma Forall2_split_last_T_r {A B: Type} : forall (n : A) ns (m : B) ms R , prod
  ⇨ (Forall2_T R ns ms) (R n m) → Forall2_T R (ns ++ [n]) (ms ++ [m])).
463 Proof.
464   intros. destruct X.
465   induction f.
466     + simpl. constructor.
467       * assumption.
468       * constructor.
469     + simpl. constructor; assumption.
470 Qed.
471
472 Lemma Forall2_head_to_last {A B : Type} : forall n m ns ms (R : A → B → Prop),
  ⇨ Forall2 R (n :: ns) (m :: ms) ⤵ Forall2 R (ns ++ [n]) (ms ++ [m])).
473 Proof.
474   intros.
475   split.
476   - intros. apply Forall2_split_last. inversion H. split; assumption.
477   - intros. apply Forall2_split_last in H. destruct H. constructor; assumption.
478 Qed.
479
480 Lemma Forall2_head_to_last_T {A B : Type} : forall n m ns ms (R : A → B →
  ⇨ Type), Forall2_T R (n :: ns) (m :: ms) → Forall2_T R (ns ++ [n]) (ms ++
  ⇨ [m])).
481 Proof.
482   intros. apply Forall2_split_last_T_r. inversion X. split; assumption.
483 Qed.
484
485 Lemma Forall2_head_to_last_T_r {A B : Type} : forall n m ns ms (R : A → B →
  ⇨ Type), Forall2_T R (ns ++ [n]) (ms ++ [m]) → Forall2_T R (n :: ns) (m ::
  ⇨ ms).
486 Proof.
487   intros. apply Forall2_split_last_T in X. destruct X. constructor; assumption.
488 Qed.
489
490 Lemma rev_nil_iff_nil {A : Type} : forall (ms : list A), [] = rev ms ⤵ ms =
  ⇨ [].
491 Proof.
492   intros.
493   split.
494   - intros. destruct ms.
495     + reflexivity.
496     + inversion H. apply app_eq_length_eq in H1. rewrite app_length in H1.
      ⇨ simpl in H1.
      rewrite plus_comm in H1. inversion H1.
497   - intros. subst. reflexivity.
498 Qed.
499
500 Lemma app_last_eq {A : Type} : forall (ms ns: list A) a b, ms ++ [a] = ns ++ [b]
  ⇨ →
501   ms = ns /\ a = b.

```

```

503 Proof.
504   induction ms.
505   - induction ns.
506     + simpl. intros. split. reflexivity. inversion H. reflexivity.
507     + simpl. intros. inversion H. symmetry in H2. apply app_eq_nil in H2.
508       inversion H2. inversion H3.
509   - induction ns.
510     + simpl. intros. inversion H. apply app_eq_nil in H2.
511       inversion H2. inversion H3.
512     + simpl in *. intros. split.
513       { inversion H. apply IHms in H2. inversion H2. subst. reflexivity. }
514       { inversion H. apply IHms in H2. inversion H2. assumption. }
515 Qed.
516
517 Lemma rev_eq {A : Type} : forall (ms ns: list A), rev ms = rev ns  $\rightarrow$  ms = ns.
518 Proof.
519   intros.
520   split.
521   - intros. remember (length ms) as lms. generalize dependent ms. generalize
522      $\hookrightarrow$  dependent ns. induction (lms).
523     + intros. apply app_eq_length_eq in H. rewrite rev_length in H. rewrite
524        $\hookrightarrow$  rev_length in H.
525       symmetry in Heqlms. apply length_zero_iff_nil in Heqlms. subst. simpl in
526          $\hookrightarrow$  H.
527       symmetry in H. apply length_zero_iff_nil in H. subst. reflexivity.
528     + intros. assert (length ms = length ns).
529       { apply app_eq_length_eq in H. rewrite rev_length in H. rewrite rev_length
530          $\hookrightarrow$  in H. assumption. }
531       assert (1  $\leq$  length ms).
532       { rewrite  $\leftarrow$  Heqlms. firstorder. }
533       assert (1  $\leq$  length ns). { rewrite  $\leftarrow$  H0. assumption. }
534       apply l1_le_length_split in H1.
535       apply l1_le_length_split in H2.
536       inversion H1. inversion H2. inversion H3. inversion H4.
537       subst. simpl in H. apply app_last_eq in H.
538       destruct H. apply IHn in H.
539       { subst. reflexivity. }
540       { inversion Heqlms. reflexivity. }
541   - intros. subst. reflexivity.
542 Qed.
543
544 Lemma rev_cons_iff_last {A : Type} : forall (ms : list A) x l , x :: l = rev ms
545    $\hookrightarrow$   $\rightarrow$ 
546   ms = rev l ++ [x].
547 Proof.
548   intros.
549   split.
550   - intros.
551     rewrite  $\leftarrow$  (rev_involutive ms). rewrite  $\leftarrow$  rev_head_last.
552     apply rev_eq. symmetry. assumption.
553   - intros. subst. rewrite  $\leftarrow$  rev_head_last. rewrite rev_involutive.
554     reflexivity.
555 Qed.
556
557 Lemma Forall2_is_rev {A B: Type} : forall ns ms {R : A  $\rightarrow$  B  $\rightarrow$  Prop}, Forall2 R
558    $\hookrightarrow$  ns ms  $\rightarrow$  Forall2 R (rev ns) (rev ms).

```

```

553 Proof.
554   intros.
555   split.
556   - intros. induction H.
557     + simpl. constructor.
558     + simpl. apply Forall2_head_to_last. constructor; assumption.
559   - intros. remember (rev ns) as rns. remember (rev ms) as rms.
560     generalize dependent ms. generalize dependent ns.
561     induction H.
562     + intros. apply rev_nil_iff_nil in Heqrns.
563       apply rev_nil_iff_nil in Heqrms. subst. constructor.
564     + intros.
565       apply rev_cons_iff_last in Heqrns.
566       apply rev_cons_iff_last in Heqrms.
567       subst. eapply Forall2_split_last. split.
568       { apply IHForall2.
569         - symmetry. apply rev_involutive.
570         - symmetry. apply rev_involutive. }
571       assumption.
572 Qed.
573
574 Lemma Forall2_T_is_rev {A B: Type} : forall ns ms {R : A → B → Type},
575   ↪ Forall2_T R ns ms → Forall2_T R (rev ns) (rev ms).
576 Proof.
577   induction 1.
578   + simpl. constructor.
579   + simpl. apply Forall2_head_to_last_T. constructor; assumption.
580 Qed.
581
582 Lemma Forall2_T_is_rev_r {A B: Type} : forall ns ms {R : A → B → Type},
583   ↪ Forall2_T R (rev ns) (rev ms) → Forall2_T R ns ms .
584 Proof.
585   intros. remember (rev ns) as rns. remember (rev ms) as rms.
586   generalize dependent ms. generalize dependent ns.
587   induction X.
588   + intros. apply rev_nil_iff_nil in Heqrns.
589     apply rev_nil_iff_nil in Heqrms. subst. constructor.
590   + intros.
591     apply rev_cons_iff_last in Heqrns.
592     apply rev_cons_iff_last in Heqrms.
593     subst. eapply Forall2_split_last_T_r. split.
594     { apply IHX.
595       - symmetry. apply rev_involutive.
596       - symmetry. apply rev_involutive. }
597     assumption.
598 Qed.
599
600 Lemma hd_none_impl_nil {T: Type} : forall (ms : list T), hd_error ms = None →
601   ↪ ms = nil.
602 Proof.
603   intros.
604   induction ms.
605   - reflexivity.
606   - simpl in H. discriminate H.
607 Qed.

```

```

606
607 Lemma list_nonmt_split {T: Type} : forall (ms : list T), ms  $\diamond$  nil  $\rightarrow$  exists
   $\hookrightarrow$  head tail, ms = head :: tail.
608 Proof.
609   intros. split.
610   - intros. destruct (hd_error ms) eqn:he.
611     + exists t. exists (tl ms). remember (tl ms) as tail.
612     assert (hd_error ms = Some t /\ tl ms = tail). auto.
613     apply hd_error_tl_repr in H0. assumption.
614     + eapply hd_none_impl_nil in he. contradiction.
615   - intros.
616     destruct H as [head [tail Heq]].
617     intros F. subst. discriminate F.
618 Qed.
619
620 Fixpoint enumerate_aux {T : Type} (ls : list T) (start : nat) : list (nat * T)
   $\hookrightarrow$  :=
621   match ls with
622   | []  $\Rightarrow$  []
623   | x :: xs  $\Rightarrow$  (start, x) :: enumerate_aux xs (Datatypes.S start)
624   end.
625
626 Definition enumerate {T: Type} (ls: list T) : list (nat * T) :=
627   enumerate_aux ls 0.
628
629
630 Lemma prooflater : False.
631 Proof.
632   Admitted.
633
634 Ltac prooflater := exfalse; apply prooflater.
635
636 Lemma list_split_rev {A}: forall (ms : list A), (exists mshead mstail, ms =
   $\hookrightarrow$  mshead :: mstail)  $\rightarrow$  exists msinit mslast, ms = msinit ++ [mslast].
637 Proof.
638   intros. split.
639   + intros. destruct H as [hd [tl mssplit]]. assert (ms  $\diamond$  nil).
640     { intros F. subst. discriminate F. }
641     apply exists_last in H. destruct H as [x [x0 Heq]]. exists x. exists x0.
642      $\hookrightarrow$  assumption.
643   + intros. destruct H as [init [tail mssplit]]. destruct ms.
644     - symmetry in mssplit. apply app_eq_nil in mssplit. destruct mssplit. subst.
645        $\hookrightarrow$  discriminate H0.
646     - exists a. exists ms. reflexivity.
647 Qed.
648
649 Definition eq_ind_T {A : Type } :=
650 fun (x : A) (P : A  $\rightarrow$  Type) (f : P x) (y : A) (e : x = y)  $\Rightarrow$ 
651 match e in ( _ = y0 ) return (P y0) with
652 | eq_refl  $\Rightarrow$  f
653 end.
654
655 Definition list_ind_T {A : Type} :=
656 fun (P : list A  $\rightarrow$  Type) (f : P []) (f0 : forall (a : A) (l : list A), P l  $\rightarrow$  P
   $\hookrightarrow$  (a :: l))  $\Rightarrow$ 
657 fix F (l : list A) : P l := match l as l0 return (P l0) with

```

```

656 | [] ⇒ f
657 | y :: l0 ⇒ f0 y l0 (F l0)
658 end.
659
660 Definition rev_list_ind_T {A : Type} :=
661 fun (P : list A → Type) (H : P [])
662   (H0 : forall (a : A) (l : list A), P (rev l) → P (rev (a :: l))) (l : list A)
663   ↪ ⇒
664   list_ind_T (fun l0 : list A ⇒ P (rev l0)) H (fun (a : A) (l0 : list A) (IHl :
665     ↪ P (rev l0)) ⇒ H0 a l0 IHl) l.
666
667 Definition rev_ind_T {A : Type} :=
668 fun (P : list A → Type) (H : P []) (H0 : forall (x : A) (l : list A), P l → P
669   ↪ (l ++ [x]))
670   (l : list A) ⇒
671   (fun E : rev (rev l) = l ⇒
672     eq_ind_T (rev (rev l)) (fun l0 : list A ⇒ P l0)
673       (rev_list_ind_T P H (fun (a : A) (l0 : list A) (H1 : P (rev l0)) ⇒ H0 a (rev
674         ↪ l0) H1) (rev l)) l E)
675   (rev_involutive l).
676
677 Lemma nth_error_nil {A : Type} : forall x, @nth_error A [] x = None.
678 Proof.
679   induction x; reflexivity.
680 Qed.
681
682 Lemma nth_error_map {A B} : forall x (l : list A) (f : A → B),
683   nth_error ((map f) l) x = match (nth_error l x) with
684     | None
685     ↪ ⇒
686     ↪ None
687     | Some a
688     ↪ ⇒
689     ↪ Some (f
690       ↪ a)
691   end.
692
693 Proof.
694   induction x.
695   - intros. destruct l; reflexivity.
696   - intros. destruct l.
697     + reflexivity.
698     + simpl. apply IHx.
699 Qed.
700 Fixpoint nth_ok {A} (l : list A) (n : nat) : (n < length l) → A :=
701   match n with
702   | 0 ⇒ match l with
703     | [] ⇒ fun proof ⇒ False_rect A (Nat.nlt_0_r _ proof)
704     | (a :: _) ⇒ fun _ ⇒ a
705   end
706   | S m ⇒ match l with
707     | [] ⇒ fun proof ⇒ False_rect A (Nat.nlt_0_r _ proof)
708     | (a :: aa) ⇒ fun proof ⇒ nth_ok aa m (Lt.lt_S_n _ _ proof)
709   end

```

```

702   end.
703
704   Lemma nth_ok_nth_error {A} : forall n (l : list A) p a,
705     nth_ok l n p = a  $\rightarrow$  nth_error l n = Some a.
706 Proof.
707   intros.
708   split.
709   - revert n l p a.
710     induction n.
711     + intros [[h l]] p a; try solve [inversion p].
712       simpl. intros. subst. reflexivity.
713     + simpl. intros [[h l]] p a; try solve [inversion p].
714       simpl in *. apply IHn.
715   - revert n l p a. induction n.
716     + simpl. intros [[h l]] p a; intros H; try solve [inversion p]; try
717        $\hookrightarrow$  (inversion H; subst); try reflexivity.
718     + simpl. intros [[h l]] p a; intros H; try solve [inversion p]. simpl in *.
719        $\hookrightarrow$  apply IHn. assumption.
720 Qed.
721
722   Lemma nth_ok_map {A B} : forall n l (f: A  $\rightarrow$  B) p, nth_ok (map f l) n p = f
723      $\hookrightarrow$  (nth_ok l n (rew (map_length _ _) in p)).
724 Proof.
725   intros.
726   remember (nth_ok (map f l) n p) as Ha.
727   remember (nth_ok l n (rew [lt n] map_length f l in p)) as Hb.
728   symmetry in HeqHa.
729   symmetry in HeqHb.
730   rewrite nth_ok_nth_error in HeqHa.
731   rewrite nth_ok_nth_error in HeqHb.
732   rewrite nth_error_map in HeqHa. destruct (nth_error l n).
733   - inversion HeqHb. inversion HeqHa. subst. reflexivity.
734   - discriminate HeqHa.
735 Qed.
736
737   Definition tuple_dec {A} {eqdec : EqDec A eq} := prod_eqdec eqdec eqdec.
738
739   Fixpoint Inb {A} {eqdec : EqDec A eq} (x : A) l {struct l} :=
740     match l with
741     | []  $\Rightarrow$  false
742     | a :: ms  $\Rightarrow$  orb (x =b a) (Inb x ms)
743   end.
744
745   Fixpoint get_adj {A} {eqdec : EqDec A eq} (a : A) (l : list (A * A)) : list A :=
746     match l with
747     | []  $\Rightarrow$  []
748     | (a', b') :: l'  $\Rightarrow$  if (a = a') then b' :: get_adj a l' else get_adj a l'
749   end.
750
751   Inductive in_range_dir {A} (R : list (A * A)) : nat  $\rightarrow$  A  $\rightarrow$  A  $\rightarrow$  Type :=
752   | in_range_base a b : In (a, b) R  $\rightarrow$  in_range_dir R 1 a b
753   | in_range_refl a : in_range_dir R 0 a a
754   | in_range_follow a b n : in_range_dir R n a b  $\rightarrow$  in_range_dir R (S n) a b
755   | in_range_trans a b c n : In (b, c) R  $\rightarrow$  in_range_dir R n a b  $\rightarrow$  in_range_dir R
756      $\hookrightarrow$  (S n) a c.

```

```

754 Lemma in_range_dir_le {A R} {a b : A} : forall m n, n ≤ m → in_range_dir R n a
    ↪ b → in_range_dir R m a b.
755 Proof.
756   intros.
757   remember (m - n) as diff.
758   revert n m H X Heqdiff.
759   induction diff; intros.
760   - assert (n = m). { omega. } subst. assumption.
761   - assert (diff = m - S n).
762     { omega. } apply (IHdiff (S n) m). omega. apply in_range_follow. assumption.
    ↪ assumption.
763 Qed.
764
765 Lemma in_ex_dec {A} {eqdec : EqDec A eq} (a : A) R : {c & In (a, c) R} + {forall
    ↪ c : A, In (a, c) R → False}.
766 Proof.
767   induction R.
768   - right. intros. inversion H.
769   - destruct IHR.
770     + destruct s. left. exists x. constructor 2. assumption.
771     + destruct (a = fst a0).
772       * left. exists (snd a0). constructor. rewrite e. apply surjective_pairing.
773       * right. intros. inversion H.
774         -- assert (a = fst a0). subst. reflexivity. contradiction.
775         -- pose proof f c0. contradiction.
776 Defined.
777
778 Inductive ts_cl_list {A} (R: list (A * A)) : A → A → Type :=
779   | ts_R_list : forall a b, In (a, b) R → ts_cl_list R a b
780   | ts_symm_list : forall a b, ts_cl_list R a b → ts_cl_list R b a
781   | ts_trans_list : forall a b c, ts_cl_list R a b → ts_cl_list R b c →
    ↪ ts_cl_list R a c
782 .
783
784 Inductive eq_cl_list {A} (R: list (A * A)) : A → A → Type :=
785   | eq_R_list : forall a b, In (a, b) R → eq_cl_list R a b
786   | eq_refl_list_l : forall a b, In (a, b) R → eq_cl_list R a a
787   | eq_refl_list_r : forall a b, In (b, a) R → eq_cl_list R a a
788   | eq_symm_list : forall a b, eq_cl_list R a b → eq_cl_list R b a
789   | eq_trans_list : forall a b c, eq_cl_list R a b → eq_cl_list R b c →
    ↪ eq_cl_list R a c
790 .
791
792 Lemma eq_cl_list_pump {A} {R} {a b : A} : eq_cl_list R a b → forall r,
    ↪ eq_cl_list (r::R) a b.
793 Proof.
794   intros.
795   induction X.
796   - constructor. constructor 2. assumption.
797   - econstructor 2. constructor 2. exact i.
798   - econstructor 3. constructor 2. exact i.
799   - constructor 4. assumption.
800   - econstructor 5. apply IHX1. apply IHX2.
801 Qed.
802
803

```



```

804 Inductive sym_hull {A} (R: A → A → Type) : A → A → Type :=
805 | sym_R : forall a b, R a b → sym_hull R a b
806 | sym_sym : forall a b, R a b → sym_hull R b a.
807
808 Definition flip_tuple {A B} : (A * B) → (B * A) := fun tpl ⇒ (snd tpl, fst
  ↪  tpl).
809
810 Lemma flip_tuple_invol {A B} (t : A * B) : flip_tuple (flip_tuple t) = t.
811 Proof.
812   unfold flip_tuple. simpl.
813   destruct t.
814   simpl. reflexivity.
815 Qed.
816
817 Definition flipped {A B} (R: list (A * B)) : list (B * A) := map flip_tuple R.
818
819 Definition sym_hull_list {A} (R: list (A * A)) : list (A * A) :=
820   R ++ flipped R.
821
822 Lemma In_flipped {A B}: forall (R : list (A * B)) a b, In (a, b) R → In (b, a)
  ↪  (flipped R).
823 Proof.
824   induction R.
825   - intros. inversion H.
826   - intros. destruct H.
827     + simpl. left. subst. reflexivity.
828     + constructor 2. apply IHR. assumption.
829 Qed.
830
831 Lemma flipped_invol {A B} : forall (R : list (A * B)), flipped (flipped R) = R.
832 Proof.
833   induction R.
834   - reflexivity.
835   - simpl. rewrite flip_tuple_invol. rewrite IHR. reflexivity.
836 Qed.
837
838 Lemma sym_hull_dec {A} : forall (R : A → A → Type), (forall a b , R a b + (R a
  ↪  b → False)) → forall a b, sym_hull R a b + (sym_hull R a b → False).
839 Proof.
840   intros.
841   destruct (X a b).
842   - left. constructor. assumption.
843   - destruct (X b a).
844     + left. constructor 2. assumption.
845     + right. intros F. inversion F; contradiction.
846 Defined.
847
848 Definition diag_dom {A} (R: list (A * A)) : list (A * A) :=
849   map (fun a ⇒ (fst a, fst a)) R.
850
851 Definition diag_codom {A} (R: list (A * A)) : list (A * A) :=
852   map (fun a ⇒ (snd a, snd a)) R.
853
854 Definition refl_hull {A} (R: list (A * A)) : list (A * A) :=
855   R ++ diag_dom R ++ diag_codom R.
856

```

```

857 Lemma diag_codom_eq {A} (R : list (A * A)) a b : In (a, b) (diag_codom R) →
858                                     a = b.
859 Proof.
860   intros. unfold diag_codom in H.
861   induction R.
862   - inversion H.
863   - simpl in H. destruct H.
864     + inversion H. reflexivity.
865     + apply IHR. apply H.
866 Qed.
867
868 Lemma diag_dom_in_dec {A} {eqdec: EqDec A eq} (R: list (A * A))
869   : forall a b, In (a, b) (diag_dom R) → {c & In (a, c) R}.
870 Proof.
871   induction R.
872   - intros. inversion H.
873   - intros. unfold diag_dom in H. simpl map in H. apply In_head_set in H.
874     ↪ destruct H.
875     + inversion e. subst. exists (snd a). constructor. apply surjective_pairing.
876     + apply IHR in i. destruct i. eexists. constructor 2. exact i.
877 Qed.
878
879 Lemma diag_codom_in_dec {A} {eqdec: EqDec A eq} (R: list (A * A))
880   : forall a b, In (a, b) (diag_codom R) → {c & In (c, a) R}.
881 Proof.
882   induction R.
883   - intros. inversion H.
884   - intros. unfold diag_codom in H. simpl map in H. apply In_head_set in H.
885     ↪ destruct H.
886     + inversion e. exists (fst a). constructor. apply surjective_pairing.
887     + apply IHR in i. destruct i. eexists. constructor 2. exact i.
888 Qed.
889
890 Lemma diag_dom_eq {A} (R : list (A * A)) a b : In (a, b) (diag_dom R) →
891                                     a = b.
892 Proof.
893   intros. unfold diag_dom in H.
894   induction R.
895   - inversion H.
896   - simpl in H. destruct H.
897     + inversion H. reflexivity.
898     + apply IHR. apply H.
899 Qed.
900
901 Lemma diag_dom_codom_flipped {A} (R : list (A * A)) a:
902   In a (diag_dom (R)) ⟹ In a (diag_codom (flipped R)).
903 Proof.
904   revert a.
905   induction R.
906   - reflexivity.
907   - intros. split.
908     + simpl. intros. destruct H.
909       * left. assumption.
910       * right. apply IHR. apply H.
911     + simpl. intros. destruct H.
912       * left. assumption.

```

```

911     * right. apply IHR. apply H.
912 Qed.
913
914 Lemma In_refl {A} {eqdec : EqDec A eq} R : forall (a b : A), In (a, b)
    ⇨ (refl_hull R) →
915                                     {c& prod (a = b) (In (a,
916                                     ⇨ c) R)} +
                                     {c& prod (a = b) (In (c,
                                     ⇨ a) R)} + {In (a, b)
                                     ⇨ R}.
917 Proof.
918   intros. unfold refl_hull in H. apply In_app_sumbool in H. destruct H.
919   - right. exact i.
920   - apply In_app_sumbool in i. destruct i.
921     + left. left. pose proof diag_dom_eq _ _ _ i.
922       apply diag_dom_in_dec in i. destruct i. eexists. split. assumption. exact
923       ⇨ i.
924     + left. right. pose proof diag_codom_eq _ _ _ i.
925       apply diag_codom_in_dec in i. destruct i. eexists. split. assumption.
926       ⇨ exact i.
927 Qed.
928
929 Inductive trans_hull {A} (R : list (A * A)) : A → A → Type :=
930 | trans_R : forall a b, In (a, b) R → trans_hull R a b
931 | trans_trans : forall a b c, trans_hull R a b → trans_hull R b c → trans_hull
932 ⇨ R a c.
933
934 Inductive trans_refl_hull {A} (R : list (A * A)) : A → A → Type :=
935 | trans_refl_R : forall a b, In (a, b) R → trans_refl_hull R a b
936 | trans_refl_refl_l : forall a b, In (a, b) R → trans_refl_hull R a a
937 | trans_refl_refl_r : forall a b, In (b, a) R → trans_refl_hull R a a
938 | trans_refl_trans : forall a b c, trans_refl_hull R a b → trans_refl_hull R b
939 ⇨ c →
940                                     trans_refl_hull R a c.
941
942 Inductive list_rel {A} : list (A * A) → A → A → Type :=
943 | list_rel_head : forall a b x R, x = (a, b) → list_rel (x::R) a b
944 | list_rel_tail : forall a b x R, list_rel R a b → list_rel (x::R) a b.
945
946 Lemma In_sym_list_dec {A} {eqdec : EqDec A eq} (a b : A) R : In (a, b)
    ⇨ (sym_hull_list R) → {In (a, b) R} + {In (b, a) R}.
947 Proof.
948   intros. unfold sym_hull_list in H. apply In_app_sumbool in H. destruct H.
949   - left. auto.
950   - right. apply In_flipped in i. rewrite flipped_invol in i. auto.
951 Defined.
952
953 Lemma In_sym_list_sym {A} (a b : A) R : In (a, b) (sym_hull_list R) → In (b, a)
    ⇨ (sym_hull_list R).
954 Proof.
955   intros.
956   unfold sym_hull_list in *. apply in_app_or in H. apply in_or_app.
957   destruct H. right. apply In_flipped. assumption.
958   left. apply In_flipped in H. rewrite flipped_invol in H.
959   assumption.

```

```

957 Qed.
958
959 Lemma trans_refl_sym_eq_cl_list {A} {eqdec : EqDec A eq}: forall R (a b : A),
960   trans_hull (refl_hull (sym_hull_list R)) a b → eq_cl_list R a b.
961 Proof.
962   intros.
963   induction X.
964   - apply In_refl in i. destruct i as [[[c [Heq HIn]] | [c [Heq HIn]]] | HIn].
965     + subst. unfold sym_hull_list in HIn. apply In_app_sumbool in HIn. destruct
966       ↪ HIn.
967       * econstructor 2. exact i.
968       * econstructor 3. apply In_flipped in i. rewrite flipped_invol in i. exact
969         ↪ i.
970     + subst. unfold sym_hull_list in HIn. apply In_app_sumbool in HIn. destruct
971       ↪ HIn.
972       * econstructor 3. exact i.
973       * econstructor 2. apply In_flipped in i. rewrite flipped_invol in i. exact
974         ↪ i.
975     + unfold sym_hull_list in HIn. apply In_app_sumbool in HIn. destruct HIn.
976       * constructor. assumption.
977       * constructor 4. constructor. apply In_flipped in i. rewrite flipped_invol
978         ↪ in i. assumption.
979   - econstructor 5. exact IHX1. assumption.
980 Qed.
981
982 Lemma trans_refl_sym_eq_cl_list {A} {eqdec : EqDec A eq}: forall R (a b : A),
983   ↪ trans_refl_hull (sym_hull_list R) a b → eq_cl_list R a b.
984 Proof.
985   intros.
986   induction X.
987   - induction R.
988     + inversion i.
989     + unfold sym_hull_list in i. apply In_app_sumbool in i. destruct i.
990       * apply In_head_set in i. destruct i.
991         -- subst. constructor. constructor. reflexivity.
992         -- constructor. constructor 2. assumption.
993       * unfold flipped in i. simpl map in i. apply In_head_set in i. destruct i.
994         -- constructor 4. constructor. inversion e. subst. constructor. destruct
995         ↪ a0. reflexivity.
996         -- apply eq_cl_list_pump. apply IHR. unfold sym_hull_list. apply
997         ↪ in_or_app. right. assumption.
998   - apply In_sym_list_dec in i. destruct i.
999     + econstructor 2. apply i.
1000     + econstructor 3. apply i.
1001   - apply In_sym_list_dec in i. destruct i.
1002     + econstructor 3. apply i.
1003     + econstructor 2. apply i.
1004   - econstructor 5. exact IHX1. assumption.
1005 Qed.
1006
1007 Lemma trans_refl_sym_is_sym {A} : forall R (a b : A),
1008   trans_refl_hull (sym_hull_list R) b a →
1009   trans_refl_hull (sym_hull_list R) a b.
1010 Proof.
1011   intros.
1012   induction X.

```

```

1005 - constructor. unfold sym_hull_list in *. apply in_app_or in i. apply
    ↪ in_or_app.
1006   destruct i.
1007   + right. apply In_flipped. assumption.
1008   + left. apply In_flipped in H. rewrite flipped_invol in H. assumption.
1009 - econstructor 2. exact i.
1010 - econstructor 3. exact i.
1011 - econstructor 4. apply IHX2. assumption.
1012 Qed.
1013
1014 Lemma refl_skip {A} {eqdec : EqDec A eq} R : forall (a b : A), a  $\diamond$  b  $\rightarrow$  In (a,
    ↪ b) (refl_hull R)  $\rightarrow$  In (a, b) R.
1015 Proof.
1016   intros. apply In_refl in H0. destruct H0 as [[[_ [F _]] | [_ [F _]]]].
1017   - contradiction.
1018   - contradiction.
1019   - assumption.
1020 Qed.
1021
1022 Lemma refl_sym_is_sym {A} : forall R (a b : A),
1023   In (a, b) (refl_hull (sym_hull_list R))  $\rightarrow$  In (b, a) (refl_hull
    ↪ (sym_hull_list R)).
1024 Proof.
1025   intros.
1026   unfold refl_hull in *.
1027   apply in_app_or in H. destruct H.
1028   - apply in_or_app. left. unfold sym_hull_list in *. apply in_app_or in H.
    ↪ destruct H.
1029     + apply in_or_app. right. apply In_flipped. assumption.
1030     + apply in_or_app. left. apply In_flipped in H. rewrite flipped_invol in H.
    ↪ assumption.
1031   - apply in_app_or in H. destruct H.
1032     + apply in_or_app. right. apply in_or_app. epose proof diag_dom_eq _ _ _ H.
    ↪ subst. left. assumption.
1033     + apply in_or_app. right. apply in_or_app. epose proof diag_codom_eq _ _ _
    ↪ H. subst. right. assumption.
1034 Qed.
1035
1036 Lemma trans_sym_is_sym {A} : forall R (a b : A),
1037   trans_hull (sym_hull_list R) a b  $\rightarrow$ 
1038   trans_hull (sym_hull_list R) b a.
1039 Proof.
1040   intros.
1041   induction X.
1042   - constructor. apply In_sym_list_sym. assumption.
1043   - econstructor 2. exact IHX2. assumption.
1044 Qed.
1045
1046 Lemma trans_refl_sym_is_sym2 {A} : forall R (a b : A),
1047   trans_hull (refl_hull (sym_hull_list R)) a b  $\rightarrow$ 
1048   trans_hull (refl_hull (sym_hull_list R)) b a.
1049 Proof.
1050   intros.
1051   induction X.
1052   - constructor. apply refl_sym_is_sym. assumption.
1053   - econstructor 2. exact IHX2. assumption.

```

```

1054 Qed.
1055
1056 Lemma ts_cl_list_trans_sym {A} {eqdec: EqDec A eq} : forall R (a b : A),
1057   ts_cl_list R a b →
1058   trans_hull (sym_hull_list R) a b.
1059 Proof.
1060   intros.
1061   induction X.
1062   - constructor. unfold sym_hull_list. apply in_or_app. left. assumption.
1063   - apply trans_sym_is_sym. assumption.
1064   - econstructor 2. exact IHX1. assumption.
1065 Qed.
1066
1067 Lemma trans_hull_nil {A} : forall (a b : A), trans_hull [] a b → False.
1068 Proof.
1069   intros.
1070   induction X.
1071   - inversion i.
1072   - auto.
1073 Qed.
1074
1075 Lemma trans_sym_ts_cl_list {A} {eqdec: EqDec A eq} : forall R (a b : A),
1076   trans_hull (sym_hull_list R) a b →
1077   ts_cl_list R a b.
1078 Proof.
1079   intros.
1080   induction X.
1081   - unfold sym_hull_list in i. apply In_app_sumbool in i. destruct i.
1082     + constructor. assumption.
1083     + apply In_flipped in i. rewrite flipped_invol in i. constructor 2.
1084       ↪ constructor. assumption.
1084   - econstructor 3. exact IHX1. exact IHX2.
1085 Qed.
1086
1087
1088 Lemma eq_cl_list_trans_refl_sym2 {A} {eqdec: EqDec A eq} : forall R (a b : A),
1089   eq_cl_list R a b →
1090   trans_hull (refl_hull (sym_hull_list R)) a b.
1091 Proof.
1092   intros.
1093   induction X.
1094   - constructor. unfold refl_hull. apply in_or_app. left. unfold sym_hull_list.
1095     ↪ apply in_or_app. left.
1096     assumption.
1097   - constructor. unfold refl_hull. apply in_or_app. right. apply in_or_app.
1098     left. unfold diag_dom. induction R.
1099     + inversion i.
1100     + destruct i.
1101       * simpl. left. subst. reflexivity.
1102       * right. rewrite map_app. apply in_or_app. left. clear IHR. induction R.
1103         -- inversion H.
1104         -- simpl. destruct H.
1105           ++ left. destruct a1. injection H. intros. subst. reflexivity.
1106           ++ right. apply IHR. assumption.
1107   - constructor. unfold refl_hull. apply in_or_app. right. apply in_or_app.
1108     right. unfold diag_codom. induction R.

```

```

1108 + inversion i.
1109 + destruct i.
1110 * simpl. left. subst. reflexivity.
1111 * right. rewrite map_app. apply in_or_app. left. clear IHR. induction R.
1112 -- inversion H.
1113 -- simpl. destruct H.
1114 ++ left. destruct a1. inversion H. reflexivity.
1115 ++ right. apply IHR. assumption.
1116 - apply trans_refl_sym_is_sym2. assumption.
1117 - econstructor 2. apply IHX1. assumption.
1118 Qed.
1119
1120
1121 Lemma eq_cl_list_trans_refl_sym {A} {eqdec: EqDec A eq} : forall R (a b : A),
1122   eq_cl_list R a b →
1123   trans_refl_hull (sym_hull_list R) a b.
1124 Proof.
1125   intros.
1126   induction X.
1127   - repeat constructor. induction R.
1128   + inversion i.
1129   + apply In_head_set in i. destruct i. constructor. assumption. constructor
1130     ↪ 2.
1131     simpl. apply in_or_app. left. assumption.
1132   - econstructor 2. unfold sym_hull_list. apply in_or_app. left. exact i.
1133   - econstructor 3. unfold sym_hull_list. apply in_or_app. left. exact i.
1134   - apply trans_refl_sym_is_sym. assumption.
1135   - econstructor 4. exact IHX1. assumption.
1136 Qed.
1137
1138 Lemma trs_eq_cl_dec {A} {eqdec : EqDec A eq} (R : list (A * A)) : (forall a b,
1139   ↪ trans_refl_hull (sym_hull_list R) a b +
1140   (trans_refl_hull (sym_hull_list R) a b → False)) →
1141   (forall a b, eq_cl_list R a b + (eq_cl_list R a b →
1142     ↪ False)).
1143 Proof.
1144   intros.
1145   destruct (X a b).
1146   - left. apply trans_refl_sym_eq_cl_list. assumption.
1147   - right. intros. apply f. apply eq_cl_list_trans_refl_sym. assumption.
1148 Defined.
1149
1150
1151 Inductive t_path {A} (R: list (A * A)) : list (A * A) → A → A → Type :=
1152 | t_path_R : forall a b, In (a, b) R → t_path R [(a, b)] a b
1153 | t_path_trans : forall a b c p, In (a, b) R → t_path R p b c → t_path R
1154   ↪ ((a,b) :: p) a c.
1155
1156 Inductive tr_path {A} (R: list (A * A)) : list (A * A) → A → A → Type :=
1157 | tr_path_R : forall a b, In (a, b) R → tr_path R [(a, b)] a b
1158 | tr_path_refl_l : forall a b, In (a, b) R → tr_path R [] a a
1159 | tr_path_refl_r : forall a b, In (b, a) R → tr_path R [] a a
1160 | tr_path_trans : forall a b c p, In (a, b) R → tr_path R p b c → tr_path R
1161   ↪ ((a,b) :: p) a c.
1162
1163 Lemma t_path_nil {A} : forall (a b : A) p, t_path [] p a b → False.
1164 Proof.

```

```

1159   intros.
1160   induction X; inversion i.
1161 Qed.
1162
1163 Lemma t_path_0 {A} : forall (a b : A) R, t_path R [] a b → False.
1164 Proof.
1165   intros.
1166   inversion X.
1167 Qed.
1168
1169 Lemma t_path_start {A} R : forall P (a b a' b' : A), t_path R ((a, b)::P) a' b'
1170   → → a = a'.
1171 Proof.
1172   intros.
1173   inversion X; reflexivity.
1174 Qed.
1175
1176 Lemma t_path_Sn {A} (R : list (A * A)) P : forall (a b x : A),
1177   t_path R ((a, x) :: P) a b → prod (In (a, x) R) (t_path R P x b) + {prod
1178   → (P = []) (prod (b = x) (In (a, b) R))}.
1179 Proof.
1180   intros.
1181   inversion X.
1182   - subst. right. split. reflexivity. split. reflexivity. assumption.
1183   - subst. left. split; assumption.
1184 Qed.
1185
1186 Lemma t_path_P_in_R {A} (R : list (A * A)) P (a b : A): t_path R P a b → forall
1187   → a' b', In (a', b') P → In (a', b') R.
1188 Proof.
1189   induction 1.
1190   - intros. inversion H. injection H0. intros. subst. assumption. inversion H0.
1191   - intros. destruct H. injection H. intros. subst. assumption. apply IHX.
1192   → assumption.
1193 Qed.
1194
1195 Lemma t_path_trans2 {A} : forall P P' (a b c : A) R , t_path R P a b → t_path R
1196   → P' b c → t_path R (P ++ P') a c.
1197 Proof.
1198   induction P; intros.
1199   - inversion X.
1200   - destruct a. pose proof t_path_start _ _ _ _ _ X. subst.
1201     pose proof t_path_P_in_R _ _ _ _ X a0 a1 (in_eq _ _).
1202     simpl. pose proof t_path_Sn _ _ _ _ _ X as [[Hin Htr]][Heq [Hnil Hin]]].
1203   + constructor 2.
1204     * assumption.
1205     * eapply IHP. apply Htr. assumption.
1206   + subst. simpl. constructor 2. assumption. assumption.
1207 Qed.
1208
1209 Lemma t_path_ex {A} R (a b : A) : trans_hull R a b →
1210   {P & t_path R P a b}.
1211 Proof.
1212   induction 1.
1213   - eexists. constructor. assumption.

```



```

1210 - destruct IHX1. destruct IHX2. exists (x ++ x0). eapply t_path_trans2.
1211 + apply t.
1212 + assumption.
1213 Qed.
1214
1215 Lemma t_path_trh {A} R (a b : A) P : t_path R P a b → trans_hull R a b.
1216 Proof.
1217   induction 1.
1218   - constructor. assumption.
1219   - econstructor 2. constructor. apply i. exact IHX.
1220 Qed.
1221
1222 Lemma t_path_pump {A} R (a b : A) P : t_path R P a b → forall ab, t_path
1223   ↪ (ab :: R) P a b.
1224 Proof.
1225   induction 1.
1226   - intros. constructor. constructor 2. assumption.
1227   - intros. constructor 2. constructor 2. assumption. apply IHX.
1228 Qed.
1229
1230 Lemma t_path_trans_dec {A} {eqdec : EqDec A eq} R (a b a' b' : A)
1231   (IHR : forall a b : A,
1232     {P : list (A * A) & t_path R P a b} + {forall P : list (A * A), t_path
1233       ↪ R P a b → False}) :
1234   {P1 & {P2 & ((t_path R P1 a a') * (t_path R P2 b' b))%type }} +
1235   {forall P1 P2, ((t_path R P1 a a') * (t_path R P2 b' b))%type →
1236     ↪ False}.
1237 Proof.
1238   destruct (IHR a a') as [[P Htr] | Htr].
1239   - destruct (IHR b' b) as [[P' Htr'] | Htr'].
1240   + left. eexists. eexists. split. apply Htr. apply Htr'.
1241   + right. intros P1 P2 [t1 t2]. eapply Htr'. apply t2.
1242   - right. intros P1 P2 [t1 t2]. eapply Htr. apply t1.
1243 Defined.
1244
1245 Definition ex_in_rel {A} (a : A) R : Type := ({c & In (a,c) R} + {c & In (c, a)
1246   ↪ R})%type.
1247
1248 Lemma ex_in_rel_dec {A} {eqdec : EqDec A eq} : forall R (a : A), ex_in_rel a R +
1249   ↪ (ex_in_rel a R → False).
1250 Proof.
1251   induction R.
1252   - intros. right. intros. unfold ex_in_rel in X. destruct X as [[c Hin] | [c
1253     ↪ Hin]]; inversion Hin.
1254   - intros. destruct (IHR a0).
1255   + left. unfold ex_in_rel. destruct e as [[c H][c H]].
1256   * left. exists c. constructor 2. assumption.
1257   * right. exists c. constructor 2. assumption.
1258   + destruct a. destruct (a0 = a1).
1259   * left. unfold ex_in_rel. rewrite e. right. exists a. constructor.
1260     ↪ reflexivity.
1261   * destruct (a0 = a).
1262     -- rewrite e. left. left. exists a1. constructor. reflexivity.
1263     -- right. intros. destruct X as [[c' H][c' H]].
1264     ++ apply In_head_set in H. destruct H.
1265     ** inversion e. subst. apply c0. reflexivity.

```

```

1259      ** apply f. left. exists c'. assumption.
1260 ++ apply In_head_set in H. destruct H.
1261      ** inversion e. subst. apply c. reflexivity.
1262      ** apply f. right. exists c'. assumption.

```

Defined.

```

1263
1264
1265 Lemma t_path_trans_R_and {A} {eqdec: EqDec A eq} : forall R (a b a' b' : A) P',
1266   t_path ((a', b')::R) P' a b → {P & t_path R P a b} + {(a', b') = (a, b)} +
1267   {P & prod (t_path R P a a') (b = b')} +
1268   {P & prod (t_path R P b' b) (a = a')} +
1269   {P1 & {P2 & prod (t_path R P1 a a') (t_path R
    ↪ P2 b' b)}}}.

```

Proof.

```

1270   intros.
1271   induction X.
1272 - apply In_head_set in i. destruct i.
1273   + inversion e. subst. left. left. left. right. auto.
1274   + left. left. left. left. eexists. constructor. assumption.
1275 - apply In_head_set in i. destruct i.
1276   + destruct IHX as [[[[[P Htr]| Heq]| [P [Htr Heq]]]| [P [Htr Heq]]]| [P1 [P2
    ↪ [Htr1 Htr2]]]]; inversion e; subst.
1277     * left. right. eexists. split. apply Htr. reflexivity.
1278     * left. left. left. right. inversion Heq. subst. assumption.
1279     * left. left. left. right. reflexivity.
1280     * left. right. eexists. split. apply Htr. reflexivity.
1281     * left. right. eexists. split. apply Htr2. reflexivity.
1282   + destruct IHX as [[[[[P Htr]| Heq]| [P [Htr Heq]]]| [P [Htr Heq]]]| [P1 [P2
    ↪ [Htr1 Htr2]]]].
1283     * left. left. left. left. eexists. econstructor 2. apply i. apply Htr.
1284     * inversion Heq. subst. left. left. right. eexists. split. constructor.
1285       ↪ assumption. reflexivity.
1286     * left. left. right. eexists. split. econstructor 2. apply i. apply Htr.
1287       ↪ assumption.
1288     * right. eexists. eexists. split. constructor. subst. assumption. apply
1289       ↪ Htr.
1290     * right. eexists. eexists. split. constructor 2. apply i. apply Htr1.
1291       ↪ apply Htr2.

```

Qed.

```

1292
1293 Lemma prod_dec_ex {A} P Q {pdec: {p : A & P p} + {forall p, P p → False}}
1294   ↪ {qdec : Q + (Q → False)} :
1295   {p & (P p * Q)%type} + {forall p, (P p * Q)%type → False}.

```

Proof.

```

1296   destruct pdec as [[p HP]| HP].
1297 - destruct qdec.
1298   + left. exists p. split; assumption.
1299   + right. intros. apply f. destruct X. assumption.
1300 - right. intros. eapply HP. destruct X. apply p0.

```

Defined.

```

1301
1302 Lemma t_path_dec {A} {eqdec : EqDec A eq} : forall R (a b : A), {P & t_path R P
    ↪ a b} + {forall P, (t_path R P a b → False)}.

```

Proof.

```

1303   induction R; intros.
1304 - right. apply t_path_nil.
1305 - destruct (IHR a0 b) as [[P IH] | H1].

```

```

1306 {left. eexists. apply t_path_pump. apply IH. }
1307 destruct a. destruct ((a, a1) = (a0, b)).
1308 {left. inversion e. subst. eexists. constructor. constructor. reflexivity. }
1309 destruct (@prod_dec_ex _ (fun P => t_path R P a0 a) (a1 = b) (IHR a0 a)).
1310 { destruct (a1 = b). rewrite e. left. reflexivity. right. intros.
  - contradiction. }
1311 { destruct s as [p [Htr Heq]]. subst. left. eexists. eapply t_path_trans2.
  - apply t_path_pump. apply Htr.
  - constructor. constructor. reflexivity. }
1312
1313 destruct (@prod_dec_ex _ (fun P => t_path R P a1 b) (a0 = a) (IHR a1 b)).
1314 { destruct (a0 = a). rewrite e. left. reflexivity. right. intros.
  - contradiction. }
1315
1316 { destruct s as [p [Htr Heq]]. subst. left. eexists. eapply t_path_trans2.
  - constructor. constructor. reflexivity.
  - apply t_path_pump. apply Htr. }
1317
1318 destruct (t_path_trans_dec _ a0 b a a1 IHR) as [[P1 [P2 [Ht1 Ht2]]]]. left.
1319 eexists. eapply t_path_trans2. eapply t_path_trans2. apply t_path_pump.
1320 -> exact Ht1. constructor.
1321 constructor. reflexivity. apply t_path_pump. apply Ht2.
1322 right. intros.
1323 pose proof t_path_trans_R_and _ _ _ _ _ X as [[[[[P' Htr]] Heq]] [P' [Htr
  -> Heq]]] [P' [Htr Heq]]] [P1 [P2 [Htr1 Htr2]]]].
1324 + eapply H1. apply Htr.
1325 + contradiction.
1326 + eapply f. split. apply Htr. subst. reflexivity.
1327 + eapply f0. split. apply Htr. assumption.
1328 + eapply f1. split. apply Htr1. apply Htr2.
1329 Defined.
1330
1331 Lemma trans_hull_dec {A} {eqdec: EqDec A eq}: forall R (a b :A), trans_hull R a
  -> b + (trans_hull R a b -> False).
1332 Proof.
1333 intros.
1334 destruct (t_path_dec R a b) as [[P Htr]|Htr].
1335 - left. eapply t_path_trh. apply Htr.
1336 - right. intros. apply t_path_ex in X. destruct X. eapply Htr. apply t.
1337 Defined.
1338
1339 Lemma ts_cl_list_dec {A} {eqdec: EqDec A eq}: forall R (a b : A), ts_cl_list R a
  -> b + (ts_cl_list R a b -> False).
1340 Proof.
1341 intros.
1342 destruct (trans_hull_dec (sym_hull_list R) a b).
1343 - left. apply trans_sym_ts_cl_list. assumption.
1344 - right. intros. apply f. apply ts_cl_list_trans_sym. assumption.
1345 Defined.
1346
1347 Lemma eq_cl_nil_impl_refl {A} : forall (a b :A), eq_cl_list [] a b -> a = b.
1348 Proof.
1349 induction 1; try inversion i; subst; reflexivity.
1350 Qed.
1351
1352 Inductive eq_cl_bool {A} (R: A -> A -> bool) : A -> A -> Type :=
1353 | eq_R_bool : forall a b, R a b = true -> eq_cl_bool R a b
1354 | eq_refl_bool : forall a b, eq_cl_bool R a b -> eq_cl_bool R a a
1355 | eq_symm_bool : forall a b, eq_cl_bool R a b -> eq_cl_bool R b a

```

```

1356 | eq_trans_bool : forall a b c, eq_cl_bool R a b → eq_cl_bool R b c →
    ↪ eq_cl_bool R a c
1357 .
1358
1359 Inductive eq_cl_prop {A} (R: A → A → Prop) : A → A → Type :=
1360 | eq_R_prop : forall a b, R a b → eq_cl_prop R a b
1361 | eq_refl_prop : forall a b, eq_cl_prop R a b → eq_cl_prop R a a
1362 | eq_symm_prop : forall a b, eq_cl_prop R a b → eq_cl_prop R b a
1363 | eq_trans_prop : forall a b c, eq_cl_prop R a b → eq_cl_prop R b c →
    ↪ eq_cl_prop R a c
1364 .
1365 Inductive eq_cl_type {A} (R: A → A → Type) : A → A → Type :=
1366 | eq_R_type : forall a b, R a b → eq_cl_type R a b
1367 | eq_refl_type : forall a b, eq_cl_type R a b → eq_cl_type R a a
1368 | eq_symm_type : forall a b, eq_cl_type R a b → eq_cl_type R b a
1369 | eq_trans_type : forall a b c, eq_cl_type R a b → eq_cl_type R b c →
    ↪ eq_cl_type R a c
1370 .
1371
1372 Definition range (n: nat) : list nat := seq 0 n.
1373
1374 Lemma app_head {A} : forall (a : A) aa, a :: aa = [a] ++ aa.
1375 Proof.
1376   auto.
1377 Qed.
1378
1379
1380 Definition combine_with {A B C} (f: A → B → C) := fix dummy (l:list A) (l'
    ↪ :list B) : list C :=
1381   match l with
1382   | [] ⇒ []
1383   | a :: t ⇒ match l' with
1384   | [] ⇒ []
1385   | b :: t' ⇒ (f a b) :: (dummy t t')
1386   end
1387 end.
1388
1389
1390 Lemma nth_error_combine {A B} : forall n a b (x : A) (y : B), nth_error (combine
    ↪ a b) n = Some (x, y) →
1391   nth_error a n = Some x /\ nth_error b n = Some y.
1392 induction n; intros.
1393 - inversion H. destruct a; destruct b; try discriminate H1.
1394   simpl in *. inversion H. subst. split; reflexivity.
1395 - inversion H. destruct a; destruct b; try discriminate H1.
1396   simpl in *. apply IHn in H1. assumption.
1397 Qed.
1398
1399
1400 Fixpoint all_some {A} (l : list (option A)) : option (list A) :=
1401   match l with
1402   | [] ⇒ Some []
1403   | Some a :: xs ⇒ match all_some xs with
1404   | Some xxs ⇒ Some (a :: xxs)
1405   | None ⇒ None
1406   end

```

```

1407 | None :: xs ⇒ None
1408 end.
1409
1410 Definition option_concat {A} (l: list (option (list A))) : option (list A) :=
1411   match all_some l with
1412   | Some xs ⇒ Some (concat xs)
1413   | None ⇒ None
1414   end.
1415
1416 Instance option_eqdec {A: Type} {equiv0 : Equivalence eq} {innereqdec: EqDec A
  ⇨ eq} : EqDec (option A) eq.
1417 Proof.
1418   intros.
1419   unfold EqDec.
1420   intros.
1421   destruct x; destruct y; try destruct (a = a0) eqn:Haa.
1422   - left. inversion e. reflexivity.
1423   - right. intros F. apply some_eq in F. contradiction.
1424   - right. intros F. discriminate F.
1425   - right. intros F. discriminate F.
1426   - left. reflexivity.
1427 Defined.
1428
1429 Lemma in_not_first {A} : forall b a (x : A), In x (a :: b) → a ◇ x → In x b.
1430 Proof.
1431   intros.
1432   inversion H.
1433   - contradiction.
1434   - assumption.
1435 Qed.
1436
1437 Lemma all_some_some {A} {eq_dec : EqDec A eq} : forall ls l (x : option (A)),
  ⇨ all_some ls = Some l → In x ls →
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

```

{ y & x = Some y /\
⇨ In y l }.

```

1439 Proof.
1440   induction ls.
1441   - intros. inversion H0.
1442   - intros. simpl in H.
1443     destruct a eqn:Ha; try discriminate H.
1444     destruct (all_some ls) eqn:Hls; try discriminate H.
1445     apply some_eq in H.
1446     destruct (option_eqdec x a).
1447     + subst. exists a0. split. rewrite e. reflexivity. constructor. reflexivity.
1448     + apply in_not_first in H0; try (rewrite ← Ha; intros F; symmetry in F;
      ⇨ contradiction).
1449     pose proof (IHls l0 x eq_refl H0). subst. destruct X as [y [Hx Hin]].
1450     exists y. split. assumption. constructor 2. assumption.
1451 Qed.
1452
1453 Lemma all_some_none_head {A} (a : A) ls : all_some (Some a :: ls) = None →
  ⇨ all_some ls = None.
1454 Proof.
1455   simpl.
1456   intros.
1457   destruct (all_some ls); try discriminate H. reflexivity.

```

```

1458 Qed.
1459
1460 Lemma all_some_none_last {A} (a : A) ls : all_some (ls ++ [Some a]) = None →
  ↪ all_some ls = None.
1461 Proof.
1462   intros.
1463   induction ls.
1464   - simpl in H. discriminate H.
1465   - simpl in H. destruct a0.
1466     destruct (all_some (ls ++ [Some a])).
1467     + discriminate H.
1468     + apply IHls in H.
1469       simpl. rewrite H. reflexivity.
1470     + reflexivity.
1471 Qed.
1472
1473 Lemma all_some_some_app_l {A} (l1 l2 : list (option A)) l3 : all_some (l1 ++ l2)
  ↪ = Some l3 →
1474
  exists l4,
  ↪ all_some l1
  ↪ = Some l4.
1475 Proof.
1476   revert l3.
1477   induction l1.
1478   - simpl. exists []. reflexivity.
1479   - intros. simpl. rewrite ← app_comm_cons in H. destruct a eqn:Ha.
1480     + simpl in H.
1481       destruct (all_some (_ ++ _)); try discriminate H. pose proof (IHl1 l
  ↪ eq_refl). destruct H0.
1482       rewrite H0. eexists. reflexivity.
1483     + simpl in H. discriminate H.
1484 Qed.
1485
1486 Lemma all_some_forall_eq {A} {eqdec : EqDec A eq} (l : list (option A)) sl :
  ↪ all_some l = Some sl →
1487
  forall i, In i l →
  ↪ { s & i = Some s
  ↪ /\ In s sl}.
1488 Proof.
1489   revert sl.
1490   induction l.
1491   - intros. inversion H0.
1492   - intros. unfold all_some in H at 1. destruct a; try discriminate H. fold
  ↪ (@all_some A) in H.
1493     destruct (all_some l) eqn:Hall; try discriminate H. apply some_eq in H.
1494     rewrite app_head in H0. apply In_app_sumbool in H0. destruct H0.
1495     + exists a. subst. split. symmetry. inversion i0. assumption. exfalse.
  ↪ inversion H. constructor. reflexivity.
1496     + subst. pose proof (IHl _ eq_refl i i0) as [s [Hisome Hin]].
1497     exists s. split. assumption. constructor 2. assumption.
1498 Qed.
1499
1500 Lemma le_tail {A n ms} {n0 : A}: S n < length (n0 :: ms) → n < length ms.
1501 Proof.
1502   intros. simpl in H. apply lt_S_n in H. assumption.
1503 Qed.

```

```

1504
1505 Lemma nth_ok_skip {A}: forall (n0 :A) n ms p1, nth_ok (n0 :: ms) (S n) p1 =
    ↪ nth_ok ms n (le_tail p1).
1506 Proof.
1507   intros.
1508   simpl.
1509   remember (nth_ok ms n _) as n1.
1510   symmetry in Heqn1. symmetry. apply nth_ok_nth_error. apply nth_ok_nth_error in
    ↪ Heqn1. assumption.
1511 Qed.
1512
1513 Lemma in_concat {A} : forall l (x : A), In [x] l → In x (concat l).
1514 Proof.
1515   induction l.
1516   - intros. inversion H.
1517   - simpl. intros x [Heq | Hin].
1518     + subst. constructor. reflexivity.
1519     + apply in_or_app. right. apply IHl. assumption.
1520 Qed.
1521
1522 Lemma combine_with_map {A B C}: forall ms ns (f : A → B → C),
1523   combine_with f ms ns = map (fun mi ⇒ f (fst mi) (snd mi))
    ↪ (combine ms ns).
1524 Proof.
1525   induction ms.
1526   - reflexivity.
1527   - intros. destruct ns. { reflexivity. }
1528     simpl. rewrite IHms. reflexivity.
1529 Qed.
1530
1531 Lemma length_combine_with {A} {B}: forall (ms : list A) (f : A → nat → B),
    ↪ length (combine_with f ms (range (length ms))) = length ms.
1532 Proof.
1533   intros.
1534   rewrite combine_with_map.
1535   rewrite map_length.
1536   rewrite combine_length.
1537   unfold range.
1538   rewrite seq_length. apply Nat.min_id.
1539 Qed.
1540
1541 Lemma lt_comb {A B n} {ms : list A} : forall (f : A → nat → B), n < length ms
    ↪ → n <
1542   length ( combine_with f ms (range (length ms))).
1543 Proof.
1544   intros.
1545   rewrite (length_combine_with ms).
1546   assumption.
1547 Qed.
1548
1549
1550 Lemma seq_nth_error : forall len start n,
1551   n < len → nth_error (seq start len) n = Some (start+n).
1552 Proof.
1553   induction len; intros.
1554   inversion H.

```

```

1555     simpl seq.
1556     destruct n; simpl.
1557     auto with arith.
1558     rewrite IHlen;simpl; auto with arith.
1559 Qed.
1560
1561 Lemma pair_eq {A B} : forall (a: A) (b: B) a' b', a = a' /\ b = b'  $\rightarrow$  (a, b) =
1562    $\rightarrow$  (a', b').
1563 Proof.
1564   intros.
1565   split.
1566   - intros [HA HB]. subst. reflexivity.
1567   - intros. inversion H. subst. split; reflexivity.
1568 Qed.
1569
1570 Lemma equivb_prop {A} {eqdec : EqDec A eq} : forall (a b : A), a ==b b = true
1571    $\rightarrow$   $\rightarrow$  a = b.
1572 Proof.
1573   intros. split; intros.
1574   - unfold equiv_decb in H. destruct (a = b). inversion e. subst. reflexivity.
1575      $\rightarrow$  discriminate H.
1576   - subst. unfold equiv_decb. destruct (equiv_dec b b).
1577     + reflexivity.
1578     + unfold complement in c. exfalso. apply c. reflexivity.
1579 Qed.
1580
1581 Lemma nequivb_prop {A} {eqdec : EqDec A eq} : forall (a b : A), a  $\diamond$ b b = true
1582    $\rightarrow$   $\rightarrow$  a  $\diamond$  b.
1583 Proof.
1584   intros. split; intros; unfold nequiv_decb in *.
1585   - apply Bool.negb_true_iff in H. unfold equiv_decb in H. destruct (a = b).
1586      $\rightarrow$  inversion H. assumption.
1587   - apply Bool.negb_true_iff. unfold equiv_decb. destruct (a = b).
1588     + contradiction.
1589     + reflexivity.
1590 Qed.
1591
1592 Definition Forall2_T_map {A B} {P : A  $\rightarrow$  B  $\rightarrow$  Type} {Q aa bb} (forall2t :
1593    $\rightarrow$  Forall2_T (fun a b  $\Rightarrow$  {p : P a b & Q a b p}) aa bb) : Forall2_T P aa bb :=
1594   Forall2_T_rect _ _ (fun l l' _  $\Rightarrow$  Forall2_T P l l') (Forall2_T_nil _) (
1595     fun a b aa bb head tail result  $\Rightarrow$  Forall2_T_cons _ _ _ _ _
1596        $\rightarrow$  (projT1 head) result
1597     ) aa bb forall2t.
1598
1599 Definition Forall2_T_pair {A B} (P : A  $\rightarrow$  B  $\rightarrow$  Type) Q F ms pis f3 :=
1600   Forall2_T_rect _ _ (fun l l' _  $\Rightarrow$  Forall2_T (fun m pi  $\Rightarrow$  { p: P m pi & Q m
1601      $\rightarrow$  pi p } ) l l') (Forall2_T_nil _) (
1602     fun a b aa bb head tail result  $\Rightarrow$ 
1603       Forall2_T_cons _ a b aa bb (existT (Q a b) head (F a b
1604          $\rightarrow$  head)) result
1605     ) ms pis f3.
1606
1607 Lemma Forall2_T_map_inv {A B} (P : A  $\rightarrow$  B  $\rightarrow$  Type) Q aa bb f3 : forall F,
1608    $\rightarrow$  Forall2_T_map (Forall2_T_pair P Q F aa bb f3) = f3.
1609 Proof.
1610   intro F.

```



```

1601   induction f3.
1602   - constructor.
1603   - simpl. apply f_equal. assumption.
1604 Qed.
1605
1606 Definition Rsub {A} (R R' : A → A → Type) := forall (pi pi' : A), R pi pi' → R'
1607   ↪ pi pi'.
1608
1609 Definition Rsub_list {A} R R' := forall (pi pi' : A), In (pi, pi') R → In (pi,
1610   ↪ pi') R'.
1611
1612 Lemma Rsub_list_ts {A} R R' : @Rsub_list A R R' → @Rsub A (ts_cl_list R)
1613   ↪ (ts_cl_list R').
1614
1615 Proof.
1616   unfold Rsub_list.
1617   unfold Rsub.
1618   intros.
1619   induction X.
1620   - constructor. apply H. assumption.
1621   - constructor 2. assumption.
1622   - econstructor 3.
1623     + apply IHX1.
1624     + apply IHX2.
1625 Qed.
1626
1627 Lemma seq_head : forall b a, seq a (S b) = a :: seq (S a) b.
1628
1629 Proof.
1630   induction b.
1631   - reflexivity.
1632   - intros. rewrite IHb.
1633     + reflexivity.
1634 Qed.
1635
1636 Lemma list_cons_eq {A} : forall (x : A) l1 l2, l1 = l2 ⟹ x::l1 = x::l2.
1637
1638 Proof.
1639   intros.
1640   split; intros; subst. reflexivity. inversion H. reflexivity.
1641 Qed.
1642
1643 Lemma cons_app {A} : forall (a : A) l, a :: l = [a] ++ l.
1644
1645 Proof.
1646   intros.
1647   auto.
1648 Qed.
1649
1650 Lemma forallb_existsb {A} : forall P (ls : list A), forallb P ls = false →
1651   ↪ existsb (fun x ⇒ negb (P x)) ls = true.
1652
1653 Proof.
1654   intros.
1655   induction ls.
1656   - inversion H.
1657   - simpl. apply Bool.orb_true_intro.
1658     simpl in H. apply Bool.andb_false_iff in H as [H1 | H2].
1659     + left. rewrite H1. reflexivity.
1660     + right. apply IHls. apply H2.
1661 Qed.

```

```

1653
1654 Fixpoint ntimes {A} (n : nat) (f : forall m, A) : list A :=
1655   match n with
1656   | 0 => []
1657   | S n => f n :: ntimes n f
1658   end.
1659
1660 Definition ntimes_proof {A} (n: nat) : (forall m, (m < n) → A) → list A :=
1661   (fix ntimes_proof_inner (n': nat) (proof : n' < S n) (f : forall m, (m < n) →
1662     → A) {struct n'} : list A :=
1663     (match n' as n'' return (n' = n'') → list A with
1664     | 0 => fun _ => []
1665     | S n'' => fun Heq => let newproof := (Lt.lt_S_n n'' n (rew [fun _ => _] Heq
1666       → in proof)) in
1667       f n'' newproof :: ntimes_proof_inner n'' (Nat.lt_lt_succ_r _ _
1668         → newproof) f
1669     end) eq_refl) n (Nat.lt_succ_diag_r n).
1670
1671 Definition Forall_with_proof (n: nat) : (forall m, (m < n) → Prop) → Prop :=
1672   (fix ntimes_proof_inner (n': nat) (proof : n' < S n) (f : forall m, (m < n) →
1673     → Prop) {struct n'} : Prop :=
1674     (match n' as n'' return (n' = n'') → Prop with
1675     | 0 => fun _ => True
1676     | S n'' => fun Heq => let newproof := (Lt.lt_S_n n'' n (rew [fun _ => _] Heq
1677       → in proof)) in
1678       f n'' newproof /\ ntimes_proof_inner n'' (Nat.lt_lt_succ_r _ _
1679         → newproof) f
1680     end) eq_refl) n (Nat.lt_succ_diag_r n).
1681
1682 Ltac bsplit := apply andb_true_intro; split.
1683
1684 Lemma seq_skip : forall n len, seq n (S len) = n :: seq (S n) len.
1685 Proof.
1686   reflexivity.
1687 Qed.
1688
1689 Lemma nth_error_map_range {A} : forall n (f : nat → A) lms, n < lms →
1690   nth_error (map f (range lms)) n = Some (f n).
1691 Proof.
1692   induction n.
1693   - intros. unfold range. destruct lms. {inversion H. } reflexivity.
1694   - intros. unfold range. simpl. destruct lms. {inversion H. } simpl.
1695     rewrite ← seq_shift. rewrite map_map. rewrite IHn.
1696     + reflexivity.
1697     + auto with arith.
1698 Qed.
1699
1700 Fixpoint map_i {A B} (f : nat → A → B) (start : nat) (ls : list A) : list B :=
1701   match ls with
1702   | [] => []
1703   | (hd :: tl) => f start hd :: map_i f (S start) tl
1704   end.
1705
1706 Lemma nth_ok_proof_irel {A} : forall n (ms : list A) p1 p2, nth_ok ms n p1 =
1707   → nth_ok ms n p2.
1708 Proof.

```

```

1702   intros.
1703   remember (nth_ok _ _ _) as n1.
1704   symmetry. symmetry in Heqn1. apply nth_ok_nth_error. apply nth_ok_nth_error in
    ↪ Heqn1.
1705   rewrite ← Heqn1. reflexivity.
1706 Qed.
1707
1708 Lemma fold_left_max_acc : forall ls i j, fold_left Nat.max ls i < j → i < j.
1709 Proof.
1710   induction ls; intros.
1711   - assumption.
1712   - simpl in H. apply IHls in H. apply Nat.max_lub_lt_iff in H. destruct H.
    ↪ assumption.
1713 Qed.
1714
1715 Lemma fold_left_max_in : forall ls i j, fold_left Nat.max ls i < j → forall k,
    ↪ In k ls → k < j.
1716 Proof.
1717   induction ls; intros.
1718   - inversion H0.
1719   - simpl in H0. destruct H0.
1720     + simpl in H. apply fold_left_max_acc in H. apply Nat.max_lub_lt_iff in H.
        ↪ destruct H. subst. assumption.
1721     + eapply IHls. simpl in H. apply H. assumption.
1722 Qed.
1723
1724 Lemma in_fold_left_max : forall ls j i, (forall k, In k ls → k < j) → i < j →
    ↪ fold_left Nat.max ls i < j.
1725 Proof.
1726   induction ls; intros.
1727   - simpl. assumption.
1728   - simpl.
1729     destruct (dec_le i a).
1730     + rewrite (Nat.max_r _ _ H1).
1731       eapply IHls.
1732       * intros. apply H. constructor 2. assumption.
1733       * apply H. constructor. reflexivity.
1734     + apply not_le in H1. assert (a ≤ i). {
1735       omega.
1736     } rewrite (Nat.max_l _ _ H2).
1737     apply IHls. intros. apply H. constructor 2. assumption. assumption.
1738 Qed.
1739
1740 Lemma all_some_none_exists {A} : forall (ls : list (option A)), all_some ls =
    ↪ None → In None ls.
1741 Proof.
1742   induction ls.
1743   - intros F. inversion F.
1744   - intros. destruct a.
1745     + apply all_some_none_head in H. apply IHls in H. constructor 2. assumption.
1746     + constructor. reflexivity.
1747 Qed.
1748
1749 Lemma nth_error_Some2 {A} : forall (ms : list A) n, n < length ms → {x &
    ↪ nth_error ms n = Some x}.
1750 Proof.

```

```

1751 intros.
1752 apply nth_error_Some in H.
1753 destruct (nth_error ms n).
1754 - exists a. reflexivity.
1755 - exfalso. apply H. reflexivity.
1756 Qed.
1757
1758 Lemma nth_error_nth_ok {A} : forall ms (x : A) n, nth_error ms n = Some x → {
  ↪ lp & nth_ok ms n lp = x }.
1759 Proof.
1760 intros.
1761 assert (nth_error ms n <math>\Diamond</math> None).
1762 {
1763   intros F. rewrite H in F. discriminate F.
1764 }
1765 rewrite nth_error_Some in H0. exists H0.
1766 revert ms H H0.
1767 induction n; intros.
1768 - destruct ms. inversion H0. simpl in H0.
1769   simpl. simpl in H. apply some_eq in H. assumption.
1770 - destruct ms. inversion H0. simpl in H. pose proof (IHn ms H (lt_S_n _ _
  ↪ H0)).
1771   simpl. assumption.
1772 Qed.
1773
1774 Lemma In_nth_error_set {A} {eqdec : EqDec A eq} l (x : A) : In x l → { n &
  ↪ nth_error l n = Some x }.
1775 Proof.
1776 induction l.
1777 - intros. inversion H.
1778 - intros. rewrite app_head in H. apply In_app_sumbool in H. destruct H.
1779   + exists 0. simpl. inversion i. subst. reflexivity. inversion H.
1780   + apply IHl in i. destruct i. exists (S x0). rewrite ← e. reflexivity.
1781 Qed.
1782
1783 Lemma in_map_set {A B} {eqdec : EqDec B eq} {f : A → B} : forall (l : list A)
  ↪ y, In y (map f l) → { x & f x = y /\ In x l }.
1784 Proof.
1785 induction l.
1786 - intros. inversion H.
1787 - intros. rewrite app_head in H. rewrite map_app in H. apply In_app_sumbool in
  ↪ H. destruct H.
1788   + exists a. split. inversion i. assumption. inversion H. constructor.
  ↪ reflexivity.
1789   + apply IHl in i. destruct i as [x [Hf Hin]]. exists x. split. assumption.
  ↪ constructor 2. assumption.
1790 Qed.
1791
1792 Lemma forall_length_in {A} : forall ms (Pr : A → Prop), (forall n pr, Pr
  ↪ (nth_ok ms n pr)) → (forall m, In m ms → Pr m).
1793 Proof.
1794 induction ms; intros.
1795 - inversion H0.
1796 - destruct H0.
1797   + subst. pose proof (H 0 (Nat.lt_0_succ _)).
1798     simpl in H0. assumption.

```

```

1799 + apply IHms.
1800 * intros. pose proof (H (S n) (lt_n_S _ _ pr)).
1801   rewrite nth_ok_skip in H1.
1802   erewrite nth_ok_proof_irel in H1.
1803   apply H1.
1804 * assumption.
1805 Qed.
1806
1807 Lemma option_concat_head {A} : forall (m : list (option (list A))) a oms,
  ⇨ option_concat (a :: m) = Some oms →
1808
1809                                     exists
1810                                     ⇨ omms,
1811                                     ⇨ option_conca
1812                                     ⇨ m =
1813                                     ⇨ Some
1814                                     ⇨ omms.
1809 Proof.
1810   unfold option_concat.
1811   simpl.
1812   intros.
1813   destruct a; try discriminate H.
1814   destruct (all_some m); try discriminate H.
1815   exists (concat l0).
1816   reflexivity.
1817 Qed.
1818
1819 Lemma all_some_app_l {A} : forall (m1 : list (option A)) m2 ams, all_some (m1 ++
  ⇨ m2) = Some ams →
1820
1821                                     { amms & all_some m1 = Some
1822                                     ⇨ amms}.
1821 Proof.
1822   induction m1.
1823   - intros. exists []. reflexivity.
1824   - intros. simpl in H. destruct a eqn:Hl; try discriminate H.
1825     simpl. destruct (all_some (m1 ++ m2)) eqn:Hb; try discriminate H.
1826     apply IHm1 in Hb. destruct Hb. rewrite e. eexists. reflexivity.
1827 Qed.
1828
1829 Lemma all_some_app_l_sub {A} : forall (m1 : list (option A)) m2 ams, all_some
  ⇨ (m1 ++ m2) = Some ams →
1830
1831                                     { amms & all_some m1 = Some amms
1832                                     ⇨ /\ forall i, In i amms → In
1833                                     ⇨ i ams}.
1831 Proof.
1832   induction m1.
1833   - intros. exists []. split. reflexivity. intros. inversion H0.
1834   - intros. simpl in H. destruct a eqn:Hl; try discriminate H.
1835     simpl. destruct (all_some (m1 ++ m2)) eqn:Hb; try discriminate H.
1836     apply IHm1 in Hb. destruct Hb as [amms [Hall Hin]]. destruct (all_some m1);
1837       ⇨ try discriminate Hall. eexists. split.
1838   + reflexivity.
1839   + intros. apply some_eq in H. apply some_eq in Hall. rewrite ← H. destruct
1840     ⇨ H0.
1841     * constructor. assumption.
1842     * constructor 2. apply Hin. subst. assumption.
1841 Qed.

```

```

1842
1843 Lemma all_some_app {A} : forall (m1 : list (option A)) m2 ams,
1844   all_some (m1 ++ m2) = Some ams
1845   → { ams1 & { ams2 & all_some m1 = Some ams1 /\ all_some m2 = Some ams2 /\
      ↪ ams = ams1 ++ ams2 } }.
1846 Proof.
1847   induction m1.
1848   - intros. exists [] . exists ams. split.
1849     + reflexivity.
1850     + split.
1851       * assumption.
1852       * reflexivity.
1853   - intros. rewrite ← app_comm_cons in H. destruct a; try (simpl in H;
      ↪ discriminate H). simpl in H.
1854     destruct (all_some (m1 ++ m2)) eqn:Hamm; try (simpl in H; discriminate H).
1855     apply IHm1 in Hamm. destruct Hamm as [ams1 [ams2 [H1 [H2 H3]]]].
1856     eexists. eexists.
1857     split.
1858     + simpl. rewrite H1. reflexivity.
1859     + split.
1860       * apply H2.
1861       * apply some_eq in H. subst. reflexivity.
1862 Qed.
1863
1864
1865
1866 Lemma option_concat_app_l {A} : forall (m1 : list (option (list A))) m2 oms,
      ↪ option_concat (m1 ++ m2) = Some oms →
      { omms & option_concat m1 = Some omms }.
1867 Proof.
1868   unfold option_concat.
1869   intros.
1870   destruct (all_some (m1 ++ m2)) eqn:Hb; try discriminate H.
1871   apply all_some_app_l in Hb. destruct Hb. rewrite e. eexists. reflexivity.
1872 Qed.
1873
1874
1875 Lemma in_l_in_concat {A} : forall (x : list A) l, In x l → (forall i, In i x →
      ↪ In i (concat l)).
1876 Proof.
1877   induction l.
1878   - intros F. inversion F.
1879   - intros. simpl in *. destruct H.
1880     + apply in_or_app. left. subst. assumption.
1881     + apply in_or_app. right. apply IHl.
1882       * assumption.
1883       * assumption.
1884 Qed.
1885
1886
1887 Lemma concat_in {A} : forall l1 l2 (x :A), (forall i, In i l1 → In i l2) → In
      ↪ x (concat l1) → In x (concat l2).
1888 Proof.
1889   induction l1.
1890   - intros. inversion H0.
1891   - intros.
1892     simpl in H0.
1893     apply in_app_or in H0.

```

```

1893   destruct H0.
1894   + eapply in_l_in_concat in H0.
1895     * apply H0.
1896     * apply H. constructor. reflexivity.
1897   + apply IHl1.
1898   intros. apply H. constructor 2. assumption. assumption.
1899 Qed.
1900
1901 Lemma option_concat_app_l_sub {A} : forall (m1 : list (option (list A))) m2 oms,
1902   ⇨ option_concat (m1 ++ m2) = Some oms → { omms & option_concat m1 = Some omms
1903   ⇨ /\ forall x, In x omms → In x oms}.
1904 Proof.
1905   unfold option_concat.
1906   intros.
1907   destruct (all_some (m1 ++ m2)) eqn:Hb; try discriminate H.
1908   apply all_some_app_l_sub in Hb. destruct Hb as [amms [Hall Hin]].
1909   rewrite Hall. exists (concat amms). split. reflexivity. intros.
1910   apply some_eq in H. subst.
1911   eapply concat_in.
1912   - intros. apply Hin. apply H.
1913   - assumption.
1914 Qed.
1915
1916 Lemma option_concat_app {A} : forall (m1 : list (option (list A))) m2 oms,
1917   ⇨ option_concat (m1 ++ m2) = Some oms → { oms1 & {oms2 & option_concat m1 =
1918   ⇨ Some oms1 /\ option_concat m2 = Some oms2 /\ oms = oms1 ++ oms2}}.
1919 Proof.
1920   unfold option_concat.
1921   intros.
1922   destruct (all_some (m1 ++ m2)) eqn:Hb; try discriminate H.
1923   apply all_some_app in Hb. destruct Hb as [ams1 [ams2 [H1 [H2 Heq]]]].
1924   rewrite H1. rewrite H2. rewrite ← some_eq in H. rewrite Heq in H.
1925   rewrite concat_app in H. exists (concat ams1). exists (concat ams2).
1926   ⇨ firstorder.
1927 Qed.
1928
1929 Lemma ts_cl_list_sub {A} : forall (R: list (A * A)) R', Rsub_list R R' → Rsub
1930   ⇨ (ts_cl_list R) (ts_cl_list R').
1931 Proof.
1932   unfold Rsub.
1933   intros.
1934   induction X.
1935   - constructor. apply H. assumption.
1936   - constructor 2. assumption.
1937   - econstructor 3. apply IHX1. assumption.
1938 Qed.
1939
1940 Lemma Rsub_trans {t} : forall (A B C : t → t → Type) , Rsub A B → Rsub B C →
1941   ⇨ Rsub A C.
1942 Proof.
1943   firstorder.
1944 Qed.
1945
1946 Lemma Rsublist_app {A} : forall ls1 ls2 (ms : list (A * A)), Rsub_list ls1 ms →
1947   ⇨ Rsub_list ls2 ms → Rsub_list (ls1 ++ ls2) ms.
1948 Proof.

```

```

1941   unfold Rsub_list.
1942   intros.
1943   apply in_app_or in H1.
1944   destruct H1.
1945   + apply H. assumption.
1946   + apply H0. assumption.
1947 Qed.
1948
1949 Lemma Rsub_in_app {A:Type} {eqdec : EqDec A eq} : forall (R : A → A → Type)
  ⇨ oms1 oms2 , (Rsub (fun p p' ⇒ In (p, p') oms1) R) → Rsub (fun p p' ⇒ In
  ⇨ (p, p') oms2) R → Rsub (fun p p' ⇒ In (p, p') (oms1 ++ oms2)) R.
1950 Proof.
1951   unfold Rsub. intros.
1952   apply In_app_sumbool in H.
1953   destruct H.
1954   + apply X. assumption.
1955   + apply X0. assumption.
1956 Qed.
1957
1958 Lemma Rsub_in_concat {A:Type} {eqdec : EqDec A eq} : forall (R : A → A → Type)
  ⇨ l ,
1959   (forall m, In m l → (Rsub (fun p p' ⇒ In (p, p') m) R)) → Rsub (fun p
  ⇨ p' ⇒ In (p, p') (concat l)) R.
1960 Proof.
1961   induction l.
1962   - intros. unfold Rsub. intros. inversion H.
1963   - intros. simpl. apply Rsub_in_app.
1964     + apply X. constructor. reflexivity.
1965     + apply IHl. intros. apply X. constructor 2. assumption.
1966 Qed.
1967
1968 Lemma all_some_map2 {A B} {eqdec : EqDec A eq} : forall ms (f: A → option B) l,
  ⇨ all_some (map f ms) = Some l → forall m, In m ms → {x & f m = Some x}.
1969 Proof.
1970   induction ms.
1971   - intros. inversion H0.
1972   - intros. simpl in H. destruct (f a) eqn:Hfa; try discriminate H.
1973     destruct (all_some _) eqn:Hall; try discriminate H. apply In_head_set in H0.
1974       ⇨ destruct H0.
1975     + subst. exists b. apply Hfa.
1976     + eapply IHms.
1977       * apply Hall.
1978       * assumption.
1979 Qed.
1980
1981 Lemma all_some_map {A B} {eqdec : EqDec B eq} : forall ms (f : A → option B) l,
  all_some (map f ms) = Some l → (forall m, In m l → {n & In n ms /\ f n =
  ⇨ Some m}).
1982 Proof.
1983   induction ms.
1984   - intros. inversion H. subst. exfalso. inversion H0.
1985   - intros. simpl in *. destruct (f a) eqn:Hfa; try discriminate H.
1986     destruct (all_some _) eqn:Hall; try discriminate H. apply some_eq in H.
1987     pose proof (IHms f _ Hall m).
1988     rewrite ← H in H0. rewrite app_head in H0. apply In_app_sumbool in H0.
      ⇨ destruct H0.

```



```

1989 + assert (m = b). subst. inversion i. subst. reflexivity. inversion H.
      ↪ subst. exists a. split.
1990 * left. reflexivity.
1991 * assumption.
1992 + apply X in i. destruct i. destruct a0. exists x.
1993 split.
1994 * right. assumption.
1995 * assumption.
1996 Qed.
1997
1998 Lemma in_combine_range {A} : forall (ls : list A) a n start pr, (In (a, start +
      ↪ n) (combine ls (seq start (length ls)))) → nth_ok ls n pr = a.
1999 Proof.
2000 intros. rewrite nth_ok_nth_error. apply In_nth_error in H. destruct H as [n0
      ↪ H]. apply nth_error_combine in H. destruct H.
2001 assert (nth_error ls n0 <math>\Diamond</math> None). { intros F. rewrite H in F. discriminate F.
      ↪ }
2002 apply nth_error_Some in H1.
2003 pose proof (seq_nth_error (length ls) start n0 H1). rewrite H0 in H2. apply
      ↪ some_eq in H2. assert (n = n0). omega.
2004 subst. assumption.
2005 Qed.
2006
2007
2008 Inductive Exists_T {A : Type} (P : A → Type) : list A → Type :=
2009 | Exists_T_cons_hd : forall (x : A) (l : list A), P x → Exists_T P (x :: l)
2010 | Exists_T_cons_tl : forall (x : A) (l : list A), Exists_T P l → Exists_T P (x
      ↪ :: l).
2011
2012 Lemma nth_error_Some3 {A} : forall ms n (a :A), nth_error ms n = Some a → n <
      ↪ length ms.
2013 Proof.
2014 intros.
2015 apply nth_error_Some.
2016 intros F. rewrite H in F. discriminate F.
2017 Qed.
2018
2019 Lemma all_some_length {A} : forall ls (ls' : list A), all_some ls = Some ls' →
      ↪ length ls = length ls'.
2020 Proof.
2021 induction ls.
2022 - intros. inversion H. reflexivity.
2023 - intros. destruct a; try discriminate H. simpl in *. destruct (all_some ls)
      ↪ eqn:Hall; try discriminate H.
2024 destruct ls'; try discriminate H. apply some_eq in H. rewrite ← H. simpl.
      ↪ apply eq_S.
2025 apply IHls. reflexivity.
2026 Qed.
2027
2028 Lemma map_in {A B}: forall ls (f: A → B) x, In x ls → In (f x) (map f ls).
2029 Proof.
2030 induction ls.
2031 - intros. inversion H.
2032 - intros. destruct H.
2033 + simpl. subst. left. reflexivity.
2034 + simpl. right. apply IHls. assumption.

```

```

2035 Qed.
2036
2037 Lemma nth_ok_in {A} : forall (ls : list A) x pr, In (nth_ok ls x pr) ls.
2038 Proof.
2039   intros.
2040   remember (nth_ok ls x pr). symmetry in Heqa. apply nth_ok_nth_error in Heqa.
2041   ↪ eapply nth_error_In. apply Heqa.
2042 Qed.
2043
2044 Lemma all_some_some_head {A} : forall a (a0 : A) ls ls0,
2045   all_some (a :: ls) = Some (a0 :: ls0) → a = Some a0 /\ all_some ls = Some
2046   ↪ ls0.
2047 Proof.
2048   intros.
2049   simpl in H. destruct a eqn:Ha; try discriminate H. destruct (all_some ls) eqn:
2050   ↪ Hall; try discriminate H.
2051   apply some_eq in H. inversion H. split; reflexivity.
2052 Qed.
2053
2054 Lemma all_some_nth {A} : forall (ls : list (option A)) ls' (Hall : all_some ls =
2055   ↪ Some ls') x (pr : x < (length ls)),
2056   nth_ok ls x pr = Some (nth_ok ls' x (rew (all_some_length _ _ Hall) in pr)).
2057 Proof.
2058   induction ls.
2059   - intros. inversion pr.
2060   - intros. destruct ls'.
2061   + pose proof (all_some_length _ _ Hall). inversion H.
2062   + pose proof (all_some_some_head _ _ _ Hall) as [Ha Hall2]. destruct x.
2063   * simpl. assumption.
2064   * simpl.
2065     erewrite nth_ok_proof_irel.
2066     erewrite (nth_ok_proof_irel _ ls').
2067     apply IHls. Unshelve.
2068     ** firstorder.
2069     ** assumption.
2070 Qed.
2071
2072 Lemma repeat_rev {A} : forall (a : A) n, repeat a (S n) = repeat a n ++ [a].
2073 Proof.
2074   induction n.
2075   - reflexivity.
2076   - simpl. rewrite ← IHn. reflexivity.
2077 Qed.

```