

SfC.v

```

1 Require Import Coq.Arith.PeanoNat.
2 Require Import Coq.Lists.List.
3 Require Import Coq.Bool.Sumbool.
4 Require Import Coq.Classes.EquivDec.
5 Require Import Autosubst.Autosubst.
6 Require Import Datatypes.
7 Require Import Omega.
8
9 Require Import PrincInh.Types.
10 Require Import PrincInh.NFTerms.
11 Require Import PrincInh.LongTyping.
12 Require Import PrincInh.Utills.
13 Require Import PrincInh.Paths.
14
15 Import ListNotations.
16 Import EqNotations.
17
18 Open Scope bool_scope.
19
20 Inductive SfC (Delta : list path) (R: path → path → Type) : nfterm → path →
    → Type :=
21 | SfC_I s pi : SfC ((pi ++ [Src]) :: Delta) R s (pi ++ [Tgt]) →
22   SfC Delta R (□_ s) pi
23 | SfC_E ms pi pi' x : nth_error Delta x = Some pi → R (pi ++ repeat Tgt (length
    → ms)) pi' →
24   (forall n (p: n < length ms), SfC Delta R (nth_ok ms n p)
    → (make_tgt_path pi n) )
25   → SfC Delta R (!! x @@ ms) pi'.
26
27 Fixpoint proof_length_SfC {Delta R m pi} (proof : SfC Delta R m pi) : nat :=
28 match proof with
29 | SfC_I _ _ _ _ proof' ⇒ S (proof_length_SfC proof')
30 | SfC_E _ _ ms pi pi' x Deltaok Rok proofs ⇒
31   (fix dummy n (pr : n ≤ length ms) : nat :=
32     match n with
33     | 0 ⇒ fun _ ⇒ 0
34     | S n' ⇒ fun pr ⇒ S (Nat.max (proof_length_SfC (proofs n' (proj1
        → (Nat.le_succ_l n' (length ms)) pr)))
35       (dummy n' (le_Sn_le _ _ pr)))
36     end pr) (length ms) (Nat.le_refl _)
37 end.
38
39 Lemma proof_length_zero : forall Delta R m pi (proof : SfC Delta R m pi), 0 =
    → proof_length_SfC proof →
40
41 Proof.
42 intros.
43 destruct proof.
44 - discriminate H.
45 - destruct ms.
46   + exists x. reflexivity.
47   + discriminate H.

```

{x & m = !!
 → x @@
 → []}.

```

48 Qed.
49
50 Definition get_subproof_app_pi
51   {Delta R x ms pi'} (proof: Sfc Delta R (!! x @@ ms) pi') : path.
52   destruct proof; exact pi.
53 Defined.
54
55 Definition get_subproof_app_deltaok
56   {Delta R x ms pi'} (proof: Sfc Delta R (!! x @@ ms) pi') : nth_error
57   → Delta x = Some (get_subproof_app_pi proof).
58   unfold get_subproof_app_pi.
59   revert proof. remember (!! x @@ ms) as m.
60   intro proof.
61   revert Heqm.
62   destruct proof.
63   - intros. discriminate Heqm.
64   - intros. injection Heqm. intros Hx _. rewrite ← Hx. exact e.
65 Defined.
66
67 Definition get_subproof_app_Rok
68   {Delta R x ms pi'} (proof: Sfc Delta R (!! x @@ ms) pi') : R
69   → ((get_subproof_app_pi proof) ++ repeat Tgt (length ms)) pi'.
70   unfold get_subproof_app_pi.
71   revert proof. remember (!! x @@ ms) as m.
72   intro proof.
73   revert Heqm.
74   destruct proof.
75   - intros. discriminate Heqm.
76   - intros. injection Heqm. intros _ Hms. rewrite ← Hms. exact r.
77 Defined.
78
79 Definition get_subproof_app
80   {Delta R x ms pi'} (proof: Sfc Delta R (!! x @@ ms) pi') :
81   (forall n (p: n < length ms), Sfc Delta R (nth_ok ms n p)
82     (make_tgt_path (get_subproof_app_pi proof) n) ).
83   unfold get_subproof_app_pi.
84   revert proof. remember (!! x @@ ms) as m.
85   intro proof.
86   revert Heqm.
87   destruct proof.
88   - intros. discriminate Heqm.
89   - intro. injection Heqm. intros _ Hms. rewrite ← Hms. exact s.
90 Defined.
91
92 Lemma Sfc_gen_app : forall R x ms pi' Delta (proof : Sfc Delta R (!! x @@ ms)
93   → pi'),
94   Sfc_E Delta R ms
95     (get_subproof_app_pi proof) pi' x
96     (get_subproof_app_deltaok proof)
97     (get_subproof_app_Rok proof)
98     (get_subproof_app proof) = proof.
99 Proof.
100   intros.
101   unfold get_subproof_app_pi.
102   unfold get_subproof_app_deltaok.
103   unfold get_subproof_app_Rok.

```

```

101 unfold get_subproof_app.
102 generalize (eq_refl (! x @@ ms)).
103 revert proof.
104 set (P := fun m' => forall (proof : Sfc Delta R m' pi') (e : m' = !! x @@ ms),
105 Sfc_E Delta R ms match proof with
106   | Sfc_I _ _ _ pi _ | Sfc_E _ _ _ pi _ _ _ => pi
107   end pi' x
108 (match
109   proof as s in (Sfc _ _ n p)
110   return
111     (n = !! x @@ ms ->
112       nth_error Delta x = Some match s with
113         | Sfc_I _ _ _ pi _ | Sfc_E _ _ _ pi _ _ _ ->
114           pi
115         end)
116   with
117   | Sfc_I _ _ s pi _ =>
118     fun Heqm : [ ] s = !! x @@ ms =>
119     False_ind (nth_error Delta x = Some pi)
120     (eq_ind ([ ] s) (fun e0 : nfterm => match e0 with
121       | !! _ @ _ => False
122       | [ ] _ => True
123       end) I (! x @@ ms) Heqm)
124   | Sfc_E _ _ ms0 pi _ x0 e0 _ =>
125     fun Heqm : !! x0 @@ ms0 = !! x @@ ms =>
126     eq_ind x0 (fun x1 : var => nth_error Delta x1 = Some pi) e0 x
127     (f_equal (fun e1 : nfterm => match e1 with
128       | !! x1 @ _ => x1
129       | [ ] _ => x0
130       end) Heqm)
131   end e)
132 (match
133   proof as s in (Sfc _ _ n p)
134   return
135     (n = !! x @@ ms ->
136       R (match s with
137         | Sfc_I _ _ _ pi _ | Sfc_E _ _ _ pi _ _ _ => pi
138         end ++ repeat Tgt (length ms)) p)
139   with
140   | Sfc_I _ _ s pi _ =>
141     fun Heqm : [ ] s = !! x @@ ms =>
142     False_rect (R (pi ++ repeat Tgt (length ms)) pi)
143     (eq_ind ([ ] s) (fun e0 : nfterm => match e0 with
144       | !! _ @ _ => False
145       | [ ] _ => True
146       end) I (! x @@ ms) Heqm)
147   | Sfc_E _ _ ms0 pi pi'0 x0 _ r _ =>
148     fun Heqm : !! x0 @@ ms0 = !! x @@ ms =>
149     rew [fun ms1 : list nfterm => R (pi ++ repeat Tgt (length ms1)) pi'0]
150     f_equal (fun e1 : nfterm => match e1 with
151       | !! _ @ ms1 => ms1
152       | [ ] _ => ms0
153       end) Heqm in r
154   end e)
155 (match

```

```

155 proof as s in (SfC _ _ n p)
156 return
157   (n = !! x @@ ms →
158     forall (n0 : nat) (p0 : n0 < length ms),
159     SfC Delta R (nth_ok ms n0 p0)
160     (make_tgt_path match s with
161       | SfC_I _ _ _ pi _ | SfC_E _ _ _ pi _ _ _ _ => pi
162       end n0))
163 with
164 | SfC_I _ _ s pi _ =>
165   fun (Heqm : [ ] s = !! x @@ ms) (n : nat) (p : n < length ms) =>
166     False_rect (SfC Delta R (nth_ok ms n p) (make_tgt_path pi n))
167     (eq_ind ([ ] s) (fun e0 : nfterm => match e0 with
168       | !! _ @ _ => False
169       | [ ] _ => True
170       end) I (!! x @@ ms) Heqm)
171 | SfC_E _ _ ms0 pi _ x0 _ _ s =>
172   fun Heqm : !! x0 @@ ms0 = !! x @@ ms =>
173     rew [fun ms1 : list nfterm =>
174       forall (n : nat) (p : n < length ms1), SfC Delta R (nth_ok ms1 n
175         ↪ p) (make_tgt_path pi n)]
176       f_equal (fun e1 : nfterm => match e1 with
177         | !! _ @ ms1 => ms1
178         | [ ] _ => ms0
179         end) Heqm in s
180   end e) = rew [fun _ => _] e in proof).
181 assert (forall m', P m').
182 {
183   unfold P. intros m' proof' Heq.
184   destruct proof'.
185   - discriminate Heq.
186   - revert e r s.
187     injection Heq. intros Hx Hms.
188     revert Heq.
189     rewrite Hx. rewrite Hms.
190     intros Heq.
191     assert (Heq = eq_refl).
192     {
193       apply Coq.Logic.Eqdep_dec.UIP_dec.
194       apply eqdec_nfterm.
195     }
196     rewrite H. simpl.
197     reflexivity.
198   }
199 intros proof Heqm.
200 unfold P in H. generalize (H (!! x @@ ms) proof Heqm).
201 clear ...
202 match goal with
203 | [⊢ ?lhs = _ → ?lhs' = _] => assert (lhs = lhs'); [apply f_equal;
204   ↪ reflexivity]
205 end.
206 rewrite H. intros. rewrite H0.
207 rewrite (Coq.Logic.Eqdep_dec.UIP_dec eqdec_nfterm Heqm eq_refl). reflexivity.
208 Qed.

```

```

208
209 Definition get_subproof_lam {Delta R m pi}
210   (proof: SFC Delta R (λ__ m) pi) : SFC ((pi ++ [Src]) :: Delta) R m (pi
    ↪ ++ [Tgt])).
211 inversion proof. assumption.
212 Defined.
213
214 Lemma SFC_gen_lam R m pi Delta (proof : SFC Delta R (λ__ m) pi) :
215   SFC_I Delta R m pi (get_subproof_lam proof) = proof .
216 Proof.
217   unfold get_subproof_lam.
218   match goal with
219   | [ ⊢ SFC_I _ _ _ _ (_ ?eq1 ?eq2) = _ ] ⇒ generalize eq1
220   end.
221   revert proof.
222   set (P := fun m' : nfterm ⇒ (
223     forall (proof : SFC Delta R (m') pi) (e : m' = λ__ m),
224     SFC_I Delta R m pi
225     (match
226       proof in (SFC _ _ n p) return (n = λ__ m → p = pi → SFC ((pi ++ [Src])
    ↪ :: Delta) R m (pi ++ [Tgt]))
227     with
228     | SFC_I _ _ s pi0 X ⇒
229       fun (H : λ__ s = λ__ m) (H0 : pi0 = pi) ⇒
230       (rew ← [ fun n : nfterm ⇒
231         pi0 = pi →
232         SFC ((pi0 ++ [Src]) :: Delta) R n (pi0 ++ [Tgt]) →
233         SFC ((pi ++ [Src]) :: Delta) R m (pi ++ [Tgt])]
234         f_equal (fun e0 : nfterm ⇒ match e0 with
235           | !! _ @@@ _ ⇒ s
236           | λ__ s0 ⇒ s0
237         end) H in
238         (fun H1 : pi0 = pi ⇒
239           rew ← [ fun l : list dir ⇒
240             SFC ((l ++ [Src]) :: Delta) R m (l ++ [Tgt]) →
241             SFC ((pi ++ [Src]) :: Delta) R m (pi ++ [Tgt])] H1 in
242             (fun X0 : SFC ((pi ++ [Src]) :: Delta) R m (pi ++ [Tgt]) ⇒ X0))) H0
    ↪ X
243     | SFC_E _ _ ms pi0 pi' x H X X0 ⇒
244       fun (H0 : !! x @@@ ms = λ__ m) (H1 : pi' = pi) ⇒
245       False_rect
246       (pi' = pi →
247         nth_error Delta x = Some pi0 →
248         R (pi0 ++ repeat Tgt (length ms)) pi' →
249         (forall (n : nat) (p : n < length ms), SFC Delta R (nth_ok ms n p)
    ↪ (make_tgt_path pi0 n)) →
250         SFC ((pi ++ [Src]) :: Delta) R m (pi ++ [Tgt]))
251       (eq_ind (!! x @@@ ms) (fun e0 : nfterm ⇒ match e0 with
252         | !! _ @@@ _ ⇒ True
253         | λ__ _ ⇒ False
254         end) I (λ__ m) H0) H1 H X X0
255     end e eq_refl) = rew [ fun _ ⇒ _ ] e in proof
256     ).
257   assert (forall m', P m').
258   {

```

```

259 unfold P. intros m' proof' Heq.
260 destruct proof'.
261 - revert proof'.
262   remember (f_equal (fun e0 : nfterm => match e0 with
263     | !! _ @ _ => s
264     | □ _ s0 => s0
265     end) Heq) as Hfeq.
266   assert (m = s).
267   { ainv. }
268   revert Heq Hfeq HeqHfeq.
269   rewrite H.
270   intros Heq.
271   assert (Heq = eq_refl).
272   {
273     apply Coq.Logic.Eqdep_dec.UIP_dec.
274     apply eqdec_nfterm.
275   }
276   rewrite H0. simpl.
277   intros Hfeq Hfeq_eqrefl.
278   rewrite Hfeq_eqrefl.
279   unfold eq_rect_r. simpl. auto.
280 - ainv.
281 }
282 intros proof Heqm.
283 unfold P in H. generalize (H (□ _ m) proof Heqm).
284 clear ...
285 match goal with
286 | [⊢ ?lhs = _ → ?lhs' = _] => assert (lhs = lhs'); [apply f_equal;
  ↪ reflexivity]
287 end.
288 rewrite H. intros. rewrite H0.
289 rewrite (Coq.Logic.Eqdep_dec.UIP_dec eqdec_nfterm Heqm eq_refl). reflexivity.
290 Qed.
291
292
293 Inductive SfC_prev R : forall Delta Delta' m m' pi pi', SfC Delta R m pi → SfC
  ↪ Delta' R m' pi' → Type :=
294 | SfC_prev_I : forall Delta m pi (proof: SfC ((pi ++ [Src]) :: Delta) R m (pi ++
  ↪ [Tgt])),
295   SfC_prev _ _ _ _ _ proof (SfC_I _ _ _ _ proof)
296 | SfC_prev_E : forall Delta pi pi' ms x
297   (deltaok: nth_error Delta x = Some pi)
298   (Rproof: R (pi ++ repeat Tgt (length ms)) pi')
299   (proofs: (forall n (p: n < length ms), SfC Delta R (nth_ok ms n p)
  ↪ (make_tgt_path pi n) ))
300   n (ltproof: (n < length ms)),
301   SfC_prev _ _ _ _ _ (proofs n ltproof) (SfC_E _ _ _ _ _
  ↪ deltaok Rproof proofs)
302 .
303
304 Inductive SfC_subj R : forall Delta Delta' m m' pi pi', SfC Delta R m pi → SfC
  ↪ Delta' R m' pi' → Type :=
305 | SfC_subj_refl : forall Delta m pi (proof: SfC Delta R m pi), SfC_subj _ _ _ _
  ↪ _ _ _ proof proof

```

```

306 | Sfc_subj_trans : forall Delta Delta' Delta'' m m' m'' pi pi' pi'' (proof1 :
    ↪ Sfc Delta R m pi) (proof2: Sfc Delta' R m' pi') (proof3: Sfc Delta'' R m''
    ↪ pi''),
307   Sfc_subj _ _ _ _ _ _ proof1 proof2 →
308   Sfc_subj _ _ _ _ _ _ proof2 proof3 →
309   Sfc_subj _ _ _ _ _ _ proof1 proof3
310 | Sfc_subj_I : forall Delta m pi (proof: Sfc ((pi ++ [Src]) :: Delta) R m (pi ++
    ↪ [Tgt])),
311   Sfc_subj _ _ _ _ _ _ proof (Sfc_I _ _ _ _ proof)
312 | Sfc_subj_E : forall Delta pi pi' ms x
    (deltaok: nth_error Delta x = Some pi)
313   (Rproof: R (pi ++ repeat Tgt (length ms)) pi')
314   (proofs: (forall n (p: n < length ms), Sfc Delta R (nth_ok ms n p)
    ↪ (make_tgt_path pi n) ))
315     n (ltproof: (n < length ms)),
316     Sfc_subj _ _ _ _ _ _ (proofs n ltproof) (Sfc_E _ _ _ _ _ _
    ↪ deltaok Rproof proofs)
317
318 .
319
320
321 Lemma sfc_monotone_aux : forall m R R' pi Delta, Rsub R R' →
322   Sfc Delta R m pi → Sfc Delta R' m pi.
323
324 Proof.
325   induction 2.
326   - constructor. apply IHX0. assumption.
327   - econstructor.
328     + apply e.
329     + auto.
330     + ainv.
331     apply X0.
332     assumption.
333
334 Qed.
335
336 Lemma sfc_monotone : forall m R R', Rsub R R' → Sfc [] R m [] → Sfc [] R' m
    ↪ [].
337
338 Proof.
339   intros.
340   eapply sfc_monotone_aux.
341   - apply X.
342   - assumption.
343
344 Qed.
345
346 Lemma sfc_monotone_aux_list : forall m R R' pi Delta, Rsub_list R R' →
347   Sfc Delta (ts_cl_list R) m pi →
348   Sfc Delta (ts_cl_list R') m pi.
349
350 Proof.
351   intros.
352   pose proof (Rsub_list_ts R R' H).
353   eapply sfc_monotone_aux.
354   - apply X0.
355   - apply X.
356
357 Qed.
358
359 Definition evenodd_cond {Delta R m pi} (proof : Sfc Delta R m pi) :=
    (even_ones pi) /\ (odd_repo (Delta)).

```

```

356
357 Fixpoint each_judg_SfC {Delta R m pi} (P : forall Delta' R' m' pi', SfC Delta'
  ⇨ R' m' pi' → Prop) (proof : SfC Delta R m pi) : Prop :=
358   P Delta R m pi proof /\
359   match proof with
360   | SfC_I _ _ s' pi' proof' ⇒ each_judg_SfC (P) proof'
361   | SfC_E _ _ ms pi pi' x DeltaProof Rproof proof' ⇒
362     forall (n : nat) (p : n < length ms),
363       each_judg_SfC (P) (proof' n p)
364   end.
365
366 Lemma each_judg_subj_SfC : forall Delta R t pi (m : SfC Delta R t pi) Pr,
367   each_judg_SfC Pr m →
368   forall Delta' t' pi' (n : SfC Delta' R t' pi'), SfC_subj _ _ _ _ _ n m
369     ⇨ →
370     (each_judg_SfC Pr n).
371
372 Proof.
373 intros.
374 generalize dependent H.
375 generalize dependent Pr.
376 induction X; intros.
377 - assumption.
378 - apply IHX1. apply IHX2. assumption.
379 - destruct H. assumption.
380 - destruct H. apply H0.
381 Qed.
382
383 Lemma each_judg_subj_SfC_P : forall Delta R t pi (m : SfC Delta R t pi) P,
384   each_judg_SfC P m →
385   forall Delta' t' pi' (n : SfC Delta' R t' pi'), SfC_subj _ _ _ _ _ n m
386     ⇨ →
387     P _ _ _ _ n.
388
389 Proof.
390 intros.
391 pose proof (each_judg_subj_SfC _ _ _ _ H _ _ _ X).
392 destruct n; destruct H0; assumption.
393 Qed.
394
395 Fixpoint any_judg_SfC {Delta R m pi} (P : forall Delta' R' m' pi', SfC Delta' R'
  ⇨ m' pi' → Prop) (proof : SfC Delta R m pi) : Prop :=
396   P Delta R m pi proof \/
397   match proof with
398   | SfC_I _ _ s' pi' proof' ⇒ any_judg_SfC (P) proof'
399   | SfC_E _ _ ms pi pi' x DeltaProof Rproof proof' ⇒
400     exists (n : nat) (p : n < length ms),
401       any_judg_SfC (P) (proof' n p)
402   end.
403
404 Lemma evenodd_aux {Delta R m pi} (proof : SfC Delta R m pi) : evenodd_cond proof
  ⇨ → each_judg_SfC (@evenodd_cond) proof .
405
406 Proof.
407 induction proof.
408 - intros. unfold each_judg_SfC. split. { assumption. }
409 apply IHproof.
410 unfold evenodd_cond in *. destruct H as [Hev Hodd].
411 split.

```



```

407 + rewrite ← even_ones_pump. assumption.
408 + rewrite odd_repo_head_tail. unfold odd_repo.
409   constructor.
410   * simpl. apply Nat.Odd_succ. exact Hev.
411   * assumption.
412 - split. {assumption. }
413 intros. apply H.
414 destruct H0 as [Hev Hodd].
415 split.
416 + revert Hev e Hodd. clear ... intros.
417   eapply tgt_path_even_if_delta_odd.
418   * apply Hodd.
419   * eapply nth_error_In. apply e.
420 + assumption.
421 Qed.
422
423 Lemma evenodd {R m} (proof : SfC [] R m []) : each_judg_SfC (@evenodd_cond)
424   → proof .
425 Proof.
426   apply evenodd_aux.
427   unfold evenodd_cond.
428   split.
429   - unfold even_ones. simpl. exists 0. auto.
430   - unfold odd_repo. constructor.
431 Qed.
432
433 Definition r_not_refl_cond {Delta R m pi} (proof : SfC Delta R m pi) :=
434   match proof with
435   | SfC_I _ _ _ _ ⇒ True
436   | SfC_E _ _ ms pi pi' _ _ _ ⇒ ~(pi ++ repeat Tgt (length ms) = pi')
437   end.
438
439 Lemma evenodd_2_r_not_refl {Delta R m pi} (proof : SfC Delta R m pi) :
440   → evenodd_cond proof → r_not_refl_cond proof.
441 Proof.
442   destruct proof.
443   - simpl. auto.
444   - unfold r_not_refl_cond. unfold evenodd_cond.
445     intros. destruct H as [Hev Hodd].
446     unfold odd_repo in Hodd. rewrite Forall_forall in Hodd.
447     pose proof (Hodd pi). apply nth_error_In in e. apply H in e.
448     assert (~even_ones(pi ++ repeat Tgt (length ms))).
449     {
450       unfold even_ones. intros F. eapply Nat.Even_Odd_False. exact F.
451       rewrite count_occ_split. apply Odd_plus_Even_is_Odd.
452       - assumption.
453       - clear ... remember (length ms) as lms. clear ... induction lms.
454         + simpl. unfold Nat.Even. exists 0. auto.
455         + simpl. assumption.
456     }
457     unfold even_ones in H0. intros F. subst.
458     contradiction.
459 Qed.
460
461 Lemma each_judg_impl : forall (P : forall Delta R m pi, SfC Delta R m pi → Prop)
462   (Q : forall Delta R m pi, SfC Delta R m pi → Prop)

```

```

461   , (forall Delta R m pi (proof : SfC Delta R m pi), P _ _ _ proof → Q _ _ _
    ↪ _ proof)
462   → forall Delta R m pi (proof : SfC Delta R m pi), each_judg_SfC (P) proof
    ↪ → each_judg_SfC Q proof.
463 Proof.
464   induction proof.
465   - simpl. intros [Hl Hr]. split.
466     + apply H. assumption.
467     + apply IHproof. assumption.
468   - simpl. intros [Hl Hr]. split.
469     + apply H. assumption.
470     + intros. apply H0. apply Hr.
471 Qed.
472
473 Lemma r_not_refl {R m} (proof : SfC [] R m []) : each_judg_SfC
    ↪ (@r_not_refl_cond) proof.
474 Proof.
475   apply each_judg_impl with @evenodd_cond.
476   intros.
477   apply evenodd_2_r_not_refl. assumption.
478   apply evenodd.
479 Qed.
480
481 Definition is_minimal_R (m : nfterm) (R : path → path → Type) :=
482   forall R', SfC [] R' m [] → Rsub R' R.
483
484 Lemma SfC_Delta_x : forall Delta x ms pi' R, SfC Delta R (!! x @@ ms) pi' →
    ↪ nth_error Delta x < None.
485 Proof.
486   intros.
487   inv X.
488   intros F. rewrite H2 in F. inv F.
489 Qed.
490
491 Fixpoint R_m_aux (Delta: list path) (pi': path) (m : nfterm) {struct m} : option
    ↪ (list (path * path)) :=
492   match m with
493   |  s ⇒ R_m_aux ((pi' ++ [Src]) :: Delta) (pi' ++ [Tgt]) s
494   | !! x @@ ms ⇒ match nth_error Delta x with
495     | None ⇒ None
496     | Some pi ⇒ option_concat
497       (((fix combine_with ms ns :=
498         match ms with
499         | [] ⇒ []
500         | x :: xs ⇒
501           match ns with
502           | n :: ns ⇒ (R_m_aux Delta (make_tgt_path
503             ↪ pi n) x) :: combine_with xs ns
504           | [] ⇒ []
505           end
506         end) ms (range( length ms))) ++ [Some [((pi
507           ↪ ++ repeat Tgt (length ms), pi'))]]])
508   end.

```

```

509
510 Definition R_m m := R_m_aux [] [] m.
511 Hint Unfold R_m R_m_aux.
512 Hint Immediate app_nil_l app_nil_r.
513
514 Definition R_m_ts m := match R_m m with
515   | None  $\Rightarrow$  fun pi pi2  $\Rightarrow$  False
516   | Some Rmm  $\Rightarrow$  ts_cl_list (Rmm)
517   end.
518
519 Lemma combine_with_Rstuff : forall pi Delta, (fix combine_with (ms : list
 $\hookrightarrow$  nfterm) (ns : list nat) {struct ms} :
520   list (option (list (path * path))) :=
521   match ms with
522   | []  $\Rightarrow$  []
523   | x :: xs  $\Rightarrow$ 
524     match ns with
525     | []  $\Rightarrow$  []
526     | n :: ns0  $\Rightarrow$ 
527       R_m_aux Delta (make_tgt_path pi n) x :: combine_with
 $\hookrightarrow$  xs ns0
528     end
529   end)
530 = combine_with (fun x n  $\Rightarrow$  R_m_aux Delta (make_tgt_path pi n)
 $\hookrightarrow$  x).
531
532 Proof.
533   reflexivity.
534 Qed.
535
536 Lemma Rm_nth_ok : forall n ms Delta pi p1 p2, nth_ok
537   (combine_with (fun x n  $\Rightarrow$  R_m_aux Delta (make_tgt_path
 $\hookrightarrow$  pi n) x) ms (range (length ms))) n p1 = R_m_aux
 $\hookrightarrow$  Delta (make_tgt_path pi n) (nth_ok ms n p2).
538
539 Proof.
540   intros.
541   rewrite nth_ok_nth_error.
542   rewrite combine_with_map .
543   rewrite nth_error_map.
544   pose proof (combine_length ms (range (length ms))).
545   unfold range in H at 2. rewrite seq_length in H. rewrite Nat.min_id in H. pose
 $\hookrightarrow$  proof p2. rewrite  $\leftarrow$  H in H0.
546   destruct (nth_error (combine ms (range (length ms)))) eqn:Ha.
547   + destruct p.
548     apply nth_error_combine in Ha.
549     destruct Ha.
550     unfold range in H2.
551     rewrite seq_nth_error in H2; try apply p2. asimpl in *. apply some_eq in H2.
552      $\hookrightarrow$  subst.
553     rewrite  $\leftarrow$  (nth_ok_nth_error _ _ p2) in H1. subst. reflexivity.
554   + pose proof p2.
555     rewrite  $\leftarrow$  H in H0.
556     rewrite  $\leftarrow$  nth_error_Some in H0. rewrite H in H0. contradiction.
557 Qed.
558
559 Lemma Rm_in_list : forall n ms p0 (p : n < length ms) Delta,
560   In (R_m_aux Delta (p0 ++ repeat Tgt n ++ [Src]) (nth_ok ms n p))

```

```

558      (combine_with (fun x n ⇒ R_m_aux Delta (p0 ++ repeat Tgt n ++ [Src])) x)
        ↪ ms (range (length ms))).
559 Proof.
560   intros.
561   pose proof (Rm_nth_ok n ms Delta p0 (lt_comb _ p) p).
562   rewrite nth_ok_nth_error in H.
563   eapply nth_error_In.
564   apply H.
565 Qed.
566
567 Lemma R_m_ts_correct : forall m Rm Delta pi, R_m_aux Delta pi m = Some Rm →
  ↪ Sfc Delta (ts_cl_list Rm) m pi.
568 Proof.
569   induction m using nfterm_rect'.
570   - intros. simpl in H.
571     destruct (nth_error Delta x) eqn:Hx; try discriminate H.
572     rewrite (combine_with_Rstuff p0 Delta) in H.
573     unfold option_concat in H.
574     destruct (
575       all_some
576       (combine_with (fun (x : nfterm) (n : nat) ⇒ R_m_aux Delta
577         ↪ (make_tgt_path p0 n) x) ms
578         (range (length ms)) ++ [Some [(p0 ++ repeat Tgt (length ms), pi)]])
579       ) eqn:Hallsome; try discriminate H.
580     apply some_eq in H.
581     econstructor.
582     + apply Hx.
583     + constructor.
584       apply (all_some_some _ _ (Some [(p0 ++ repeat Tgt (length ms), pi)])) in
585         ↪ Hallsome.
586       ** destruct Hallsome as [y [Hsomey Hinyll]].
587       apply some_eq in Hsomey.
588       subst. apply in_concat. assumption.
589       ** apply in_or_app. right. constructor. reflexivity.
590   + intros.
591     remember (R_m_aux Delta (p0 ++ repeat Tgt n ++ [Src]) (nth_ok ms n p1)) as
592       ↪ Rmm.
593     pose proof (Rm_in_list n ms p0 p1 Delta).
594     apply (all_some_some _ _ (R_m_aux Delta (p0 ++ repeat Tgt n ++ [Src])
595       ↪ (nth_ok ms n p1))) in Hallsome.
596     * destruct Hallsome.
597     eapply (sfc_monotone_aux_list _ x0).
598     ** unfold Rsub_list. intros. destruct a.
599     rewrite ← H.
600     revert H1 H3. clear ...
601     intros. induction l.
602     {
603       inversion H3.
604     }
605     {
606       simpl.
607       apply in_or_app.
608       destruct H3.
609       - left. subst. assumption.
610       - right. apply IHl. assumption.
611     }

```

```

608      ** apply p.
609      destruct a. assumption.
610      * apply in_or_app. left. apply H0.
611      - intros. constructor.
612      apply IHm. rewrite ← H.
613      reflexivity.
614 Qed.
615
616 Lemma Sfc_var_size : forall x ms Delta R pi, Sfc Delta R (!!x @@ ms) pi → x <
  ↪ length Delta.
617 Proof.
618   intros.
619   ainv.
620   apply nth_error_Some.
621   intros F. rewrite H0 in F. discriminate F.
622 Qed.
623
624 Lemma Sfc_var_in_some x ms {Delta pi} : { R & Sfc Delta R (!!x @@ ms) pi } →
  ↪ nth_error Delta x < None.
625 Proof.
626   intros.
627   destruct X.
628   apply Sfc_var_size in s.
629   apply nth_error_Some.
630   assumption.
631 Qed.
632
633 Lemma Sfc_lam_proof s {Delta pi}: {R & Sfc Delta R (λ__ s) pi} → {R & Sfc ((pi
  ↪ ++ [Src])::Delta) R s (pi ++ [Tgt])}.
634 Proof.
635   intros.
636   destruct X.
637   inversion s0.
638   exists x. assumption.
639 Qed.
640
641 Lemma Sfc_var_proof x ms n m pi' {Delta pi} p ltp: {R & Sfc Delta R (!!x @@ ms)
  ↪ pi} → m = (nth_ok ms n p) → pi' = nth_ok Delta x ltp → {R & Sfc Delta R m
  ↪ (make_tgt_path pi' n)}.
642 Proof.
643   intros.
644   destruct X.
645   inversion s.
646   remember (nth_ok Delta x ltp).
647   symmetry in Heqp0.
648   rewrite nth_ok_nth_error in Heqp0.
649   rewrite H4 in Heqp0. apply some_eq in Heqp0. subst.
650   eexists.
651   apply X0.
652 Qed.
653
654 Lemma exists_R_m m : forall Delta, all_var_in_repo m Delta → forall pi, { R' &
  ↪ R_m_aux Delta pi m = Some R'}.
655 Proof.
656   induction m using nfterm_rect'.

```

```

657 - intros. asimpl in *. unfold all_var_in_repo in H.
658 pose proof H as Hx. apply fold_left_max_acc in Hx.
659 apply lt_S_n in Hx.
660 pose proof (in_seq (length ms) 0).
661 assert (forall n, In n (seq 0 (length ms)) → exists lp m, nth_ok ms n lp =
662   ↪ m) as Hseqlen.
663 {
664   clear ...
665   intros.
666   apply in_seq in H. destruct H. asimpl in H0.
667   apply nth_error_Some2 in H0.
668   destruct H0. apply nth_error_nth_ok in e.
669   destruct e. exists x0. exists x. assumption.
670 }
671 assert (forall (n : nat) (lp : n < length ms), max_fvar (nth_ok ms n lp) < S
672   ↪ (length Delta)) as Hmaxfvar.
673 {
674   revert H Hx.
675   clear ...
676   induction n.
677   - intros. destruct ms.
678     + inversion lp.
679     + asimpl. asimpl in H. apply fold_left_max_acc in H. destruct (max_fvar
680       ↪ n).
681       * apply Nat.lt_0_succ.
682       * apply lt_S_n in H. apply Nat.max_lub_lt_iff in H. destruct H. apply
683         ↪ lt_n_S. assumption.
684   - intros. destruct ms.
685     + inversion lp.
686     + remember (nth_ok _ _ _) as n1. symmetry in Heqn1. apply
687       ↪ nth_ok_nth_error in Heqn1.
688     asimpl in Heqn1. eapply fold_left_max_in in H.
689     * apply H.
690     * apply nth_error_In in Heqn1. constructor 2. apply in_map.
691       ↪ assumption.
692 }
693 assert (forall (n : nat) (lp : n < length ms) (pi : path),
694   { R' : list (path * path) & R_m_aux Delta pi (nth_ok ms n lp) =
695     ↪ Some R' }) as p'.
696 {
697   intros.
698   apply p.
699   apply Hmaxfvar.
700 }
701 assert (forall m pi, In m ms →
702   { R' : list (path * path) & R_m_aux Delta pi m = Some R' }) as
703   ↪ p''.
704 {
705   intros. pose proof (H1). apply In_nth_error_set in H2. destruct H2.
706   assert (x0 < length ms). {
707     apply nth_error_Some. intros F. rewrite e in F. discriminate F.
708   }
709   pose proof (p' x0 H2 pi0).
710   destruct X. exists x1. rewrite ← nth_ok_nth_error in e. rewrite
711     ↪ (nth_ok_proof_irel _ _ _ H2) in e.
712   subst. assumption.

```

```

704     Unshelve.
705     apply H2.
706 }
707 apply nth_error_Some in Hx. destruct (nth_error Delta x); try (exfalse;
    ↪ apply Hx; reflexivity).
708 rewrite (combine_with_Rstuff p0 Delta).
709 unfold option_concat.
710 destruct (all_some _) eqn:Hall.
711 + eexists. reflexivity.
712 + apply all_some_none_last in Hall. apply all_some_none_exists in Hall.
713   rewrite combine_with_map in Hall. apply in_map_set in Hall. destruct Hall
    ↪ as [[m b] [HRNone HInc]].
714   asimpl in HRNone. apply in_combine_l in HInc. pose proof (p'' m
    ↪ (make_tgt_path p0 b) HInc).
715   destruct X. rewrite HRNone in e. discriminate e.
716 - intros. simpl in *. apply IHm. unfold all_var_in_repo in *.
717   asimpl in *. omega.
718 Qed.
719
720 Definition closed m := max_fvar m = 0.
721
722 Hint Unfold closed.
723
724 Lemma closed_Rm : forall m, closed m → SfC [] (R_m_ts m) m [].
725 Proof.
726   intros.
727   unfold R_m_ts.
728   destruct (R_m m) eqn:HRm.
729   - apply R_m_ts_correct. exact HRm.
730   - unfold R_m in HRm.
731     pose proof (exists_R_m m []) as H0.
732     unfold all_var_in_repo in H0. rewrite H in H0.
733     asimpl in H0. pose proof (H0 Nat.lt_0_1). exfalse.
734     destruct (X []). rewrite HRm in e. discriminate e.
735 Qed.
736
737 Lemma SfC_repo_size : forall m R Delta pi, SfC Delta R m pi → max_fvar m < S
    ↪ (length Delta).
738 Proof.
739   induction 1.
740   - simpl.
741     asimpl in *.
742     apply lt_S_n.
743     unfold pred.
744     destruct (max_fvar s) eqn:Hmf.
745     + omega.
746     + assumption.
747   - asimpl.
748     assert (nth_error Delta x < None). { intros F. rewrite e in F. discriminate
    ↪ F. }
749     apply nth_error_Some in H0.
750     assert (forall m, In m ms → max_fvar m < S (length Delta)).
751     {
752       intros. eapply (forall_length_in ms).
753       - intros. apply In_nth_error in H1. destruct H1. apply nth_error_nth_ok
    ↪ in H1.

```

```

754         destruct H1. rewrite ← e0. apply H.
755     - apply H1.
756 }
757 apply in_fold_left_max.
758 + intros. apply in_map_iff in H2. destruct H2 as [x0 [Hmfx Hinx]]. pose
759   ↪ proof (H1 x0 Hinx).
760     rewrite Hmfx in H2. assumption.
761 + apply lt_n_S. assumption.
762 Qed.
763
764 Lemma SfC_closed : forall R m, SfC [] R m [] → closed m.
765 Proof.
766   intros.
767   unfold closed.
768   apply SfC_repo_size in X.
769   asimpl in X.
770   omega.
771 Qed.
772
773 Lemma Long_repo_size : forall m Delta tau, nfty_long Delta m tau → max_fvar m <
774   ↪ S (length Delta).
775 Proof.
776   induction 1.
777   - simpl.
778     asimpl in *.
779     apply lt_S_n.
780     unfold pred.
781     destruct (max_fvar s) eqn:Hmf.
782     + omega.
783     + assumption.
784   - asimpl in *.
785     assert (nth_error Gamma x < None). { intros F. rewrite Gammaok in F.
786       ↪ discriminate F. }
787     apply nth_error_Some in H0.
788     assert (forall m, In m ms → max_fvar m < S (length Gamma)).
789     {
790       intros. eapply (forall_length_in ms).
791       - intros. apply In_nth_error in H1. destruct H1.
792         apply nth_error_nth_ok in H1. destruct H1.
793         rewrite ← e. apply H.
794       - apply H1.
795     }
796     apply in_fold_left_max.
797 + intros. apply in_map_iff in H2. destruct H2 as [x0 [Hmfx Hinx]]. rewrite
798   ↪ ← Hmfx.
799   apply H1. assumption.
800 + apply lt_n_S. assumption.
801 Qed.
802
803 Lemma Long_closed : forall m tau, nfty_long [] m tau → closed m.
804 Proof.
805   intros. unfold closed. apply Long_repo_size in X. asimpl in X. omega.
806 Qed.

```



```

805 Lemma R_m_ts_minimal : forall m Rm Delta pi, Sfc Delta Rm m pi → forall Rm',
      ↪ R_m_aux Delta pi m = Some Rm' →
806
      Rsub (fun p p' ⇒ In (p, p') Rm')
      ↪ Rm.

807 Proof.
808   induction 1.
809   - intros. apply IHX. simpl in H. assumption.
810   - intros.
811     assert (forall skip, (forall (n : nat) (p : n < length ms), Sfc Delta R
      ↪ (nth_ok ms n p) (make_tgt_path pi (skip + n))) →
812       Forall2_T (Sfc Delta R) ms (map (make_tgt_path pi) (seq skip (length
      ↪ ms)))).
813   {
814     clear ...
815     induction ms.
816     - intros. constructor.
817     - intros. simpl. constructor.
818       + pose proof (X 0 (Nat.lt_0_succ _)).
819         asimpl in X0. apply X0.
820       + assert (forall (n : nat) (p : n < length ms), Sfc Delta R (nth_ok ms n
      ↪ p) (make_tgt_path pi (S skip + n))
821         ). {
822         intros.
823         pose proof (X (S n) (Lt.lt_n_S _ _ p)).
824         asimpl in X0.
825         erewrite (nth_ok_proof_irel) in X0.
826         apply X0.
827       }
828     apply IHms in X0. assumption.
829   }
830   apply (X0 0) in s.
831   clear X0.
832   asimpl in H. rewrite e in H. rewrite (combine_with_Rstuff pi Delta) in H.
      ↪ rewrite combine_with_map in H.
833   apply option_concat_app in H. destruct H as [oms1 [oms2 [Hom1 [Hom2 Heq]]]].
      ↪ asimpl in Hom2.
834   apply some_eq in Hom2.
835   rewrite Heq. apply Rsub_in_app.
836   + unfold option_concat in Hom1.
837     destruct (all_some _) eqn:Hall; try discriminate Hom1.
838     apply some_eq in Hom1. subst. apply Rsub_in_concat. intros.
839     apply all_some_map with (m0:=m) in Hall.
840     destruct Hall as [com [Hinc HRm]]. destruct com as [t n].
841     pose proof (in_combine_r ms _ _ Hinc) as Hinran.
842     apply in_seq in Hinran. destruct Hinran as [_ pr]. simpl in pr.
843     apply (X n pr). asimpl in HRm. apply in_combine_range with (start := 0)
      ↪ (pr0 := pr) in Hinc. ainv. assumption.
844   + subst. unfold Rsub.
845     intros. assert ((pi0, pi'0) = (pi ++ repeat Tgt (length ms), pi')).
846     { inversion H. symmetry. assumption. inversion H0. }
847     inversion H0. subst. assumption.

848 Qed.

849
850 Definition R_tau_cond (tau: type) (pipi' : path * path) : bool :=
851   let pi := fst pipi' in
852   let pi' := snd pipi' in

```

```

853 (pi <math>\triangleleft b</math> pi') && match P tau pi with
854   | None => false
855   | Some (sigma <math>\rightsquigarrow</math> tau) => false
856   | Some (? a) => match P tau pi' with
857     | None => false
858     | Some a' => (? a) = $\mathrel{=}$  b a'
859   end
860 end
861 .
862
863
864 Definition R_tau_list tau :=
865   filter (R_tau_cond tau)
866   (list_prod (dom_P tau) (dom_P tau)).
867
868 (* Another Definition of R_tau *)
869 Inductive R_tau (tau: type) : path → path → Type :=
870 | R_tau_in pi pi' :
871   In pi (dom_P tau) → In pi' (dom_P tau) → pi <math>\triangleleft</math> pi' → {a & P tau pi = Some
872     <math>\hookrightarrow</math> (? a) } →
873   P tau pi = P tau pi' → R_tau tau pi pi'.
874
875 (* Equivalence closure over R_tau *)
876 Definition R_tau_ts (tau: type) := ts_cl_list (R_tau_list tau).
877
878 (* Equivalence between R_tau_list and R_tau *)
879 Lemma R_tau_list_type : forall tau pi pi', In (pi, pi') (R_tau_list tau) <math>\rightsquigarrow</math>
880   <math>\hookrightarrow</math> inhabited (R_tau tau pi pi').
881
882 Proof.
883 split.
884 - induction tau.
885   + ainv.
886   + intros. unfold R_tau_list in H. rewrite filter_In in H. destruct H as [H1
887     <math>\hookrightarrow</math> H2]. rewrite in_prod_iff in H1.
888     destruct H1 as [Hpi Hpi']. unfold R_tau_cond in H2. simpl in H2. apply
889       <math>\hookrightarrow</math> andb_prop in H2 as [H2 H3].
890     apply nequivb_prop in H2. constructor.
891     constructor; try assumption; destruct P eqn:HP; try discriminate H3;
892     destruct t; try discriminate H3.
893     * exists x. reflexivity.
894     * destruct (P (tau1 <math>\rightsquigarrow</math> tau2) pi'); try discriminate H3.
895     rewrite equivb_prop in H3. subst. reflexivity.
896 - intros. inversion H. inversion X.
897   unfold R_tau_list. rewrite filter_In. split.
898   + rewrite in_prod_iff.
899     split; assumption.
900   + unfold R_tau_cond. simpl.
901     apply andb_true_intro. split.
902     * apply nequivb_prop. assumption.
903     * destruct P; ainv; try apply equivb_prop; ainv.
904
905 Qed.
906
907
908 Definition Delta2Gamma (tau: type) Delta : option (list type) :=
909   all_some (map (P tau) Delta).
910
911

```

```

904 Lemma subterm_long_ty : forall m tau Delta, nfty_long Delta m tau → forall n,
    ↪ subterm_nf n m → {Delta' & { tau' & nfty_long Delta' n tau'}}.
905 Proof.
906   induction 1.
907   - intros. inversion X0.
908     + subst. exists Gamma. exists (sigma ↪ tau). constructor. assumption.
909     + subst. apply IHX. assumption.
910   - intros. inversion X0.
911     + subst. exists Gamma. eexists. econstructor. eapply Gammaok. apply n.
912     + subst. apply In_nth_error_set in H1. destruct H1 as [nr H1]. apply
        ↪ nth_error_nth_ok in H1.
913       destruct H1 as [Hlen Hnth]. rewrite ← Hnth in X1.
914       pose proof (X nr Hlen n0 X1). assumption.
915 Qed.
916
917 Fixpoint wrap_lam (n: nat) m :=
918   match n with
919   | 0 ⇒ m
920   | S n ⇒ □__ (wrap_lam n m)
921   end.
922
923 Lemma Delta2Gamma_pump : forall Delta rho rho' pi, P rho pi = Some rho' →
    ↪ forall Gamma, Delta2Gamma rho Delta = Some Gamma →
924       Delta2Gamma rho (pi::Delta) = Some (rho' :: Gamma).
925 Proof.
926   intros. asimpl. rewrite H. unfold Delta2Gamma in H0. rewrite H0. reflexivity.
927 Qed.
928
929 Lemma P_lam_proof_Src {rho pi m pr Gamma} :
930   nfty_long Gamma (□__ m) (P_ok rho pi pr) → In (pi ++ [Src]) (dom_P rho).
931 Proof.
932   intros. inversion X. subst. symmetry in H. rewrite P_ok_P in H. apply P_src in
        ↪ H.
933   rewrite ← P_ok_P_ex in H. destruct H. assumption.
934 Qed.
935
936 Lemma P_lam_proof_Tgt {rho pi m pr Gamma} :
937   nfty_long Gamma (□__ m) (P_ok rho pi pr) → In (pi ++ [Tgt]) (dom_P rho).
938 Proof.
939   intros. inversion X. subst. symmetry in H. rewrite P_ok_P in H. apply P_tgt in
        ↪ H.
940   rewrite ← P_ok_P_ex in H. destruct H. assumption.
941 Qed.
942
943 Lemma P_lam_step : forall m pi Delta R,
944   SFC Delta R (□__ m) pi → forall rho Gamma,
945   Delta2Gamma rho Delta = Some Gamma →
946   forall pr (nfpr : nfty_long Gamma (□__ m) (P_ok rho pi pr)),
947   nfty_long
948     (P_ok rho (pi ++ [Src]) (P_lam_proof_Src nfpr) :: Gamma)
949     m (P_ok rho (pi ++ [Tgt]) (P_lam_proof_Tgt nfpr)).
950 Proof.
951   intros. asimpl. inv nfpr. inv X. symmetry in H2. apply P_ok_P in H2.
952   assert (P_ok rho (pi ++ [Src]) (P_lam_proof_Src nfpr) = sigma).
953   {

```

```

954   apply P_ok_P. eapply P_src. apply H2.
955 }
956 assert (P_ok rho (pi ++ [Tgt]) (P_lam_proof_Tgt nfpr) = tau).
957 {
958   apply P_ok_P. eapply P_tgt. apply H2.
959 }
960 subst. assumption.
961 Qed.
962
963 Lemma nfty_app_x_in_Gamma {x ms a Gamma} : nfty_long Gamma (!! x @@ ms) a →
964   {ts & nth_error Gamma x = Some (make_arrow_type ts
965     → a) }.
966
967 Proof.
968   intros. inversion X. exists ts. assumption.
969 Qed.
970
971 Lemma P_app_step {Delta pi' x ms R}:
972   Sfc Delta R (!! x @@ ms) pi' →
973   forall rho Gamma pi pr' ts a,
974     nth_error Delta x = Some pi →
975     nth_error Gamma x = Some (make_arrow_type ts (? a)) →
976     forall (nfpr : nfty_long Gamma (!! x @@ ms) (P_ok rho pi' pr'))
977       (prp: P rho pi = Some (make_arrow_type ts (P_ok rho pi' pr')))) (leneq :
978       → length ms = length ts)
979     n (lenpr: n < length ms),
980     nfty_long Gamma (nth_ok ms n lenpr) (P_ok rho (pi ++ repeat Tgt n ++
981       → [Src]))
982       (P_app_proof_in prp n (rew leneq
983         → in lenpr))).
984
985 Proof.
986   intros. inv nfpr. inv X. assert (ts0 = ts /\ a = a0) as [Hts Ha].
987   {
988     revert Gammaok H0. clear...
989     intros. rewrite H0 in Gammaok. clear H0. apply some_eq in Gammaok.
990     unfold make_arrow_type in Gammaok.
991     generalize dependent ts0. induction ts; intros.
992     - simpl in Gammaok.
993       destruct ts0.
994       + split. reflexivity. ainv.
995       + ainv.
996     - destruct ts0.
997       + ainv.
998       + pose proof (IHts ts0).
999         assert (t = a1).
1000         {
1001           ainv.
1002         }
1003         subst. ainv. apply H in H2. destruct H2. split. ainv. assumption.
1004   }
1005   subst.
1006   pose proof (X0 n lenpr).
1007   assert (P rho (pi ++ repeat Tgt n ++ [Src]) = Some (nth_ok ts n (rew [lt n]
1008     → Lenproof in lenpr))).
1009   {
1010     eapply P_app_proof.
1011     apply prp.

```

```

1005 }
1006 apply P_P_ok_set in H1. destruct H1 as [pr'' H1]. erewrite P_ok_proof_irl.
    ↪ rewrite H1.
1007 assumption.
1008 Qed.
1009
1010 Lemma Delta2Gamma_length {rho Delta Gamma} : Delta2Gamma rho Delta = Some Gamma
    ↪ → length Delta = length Gamma.
1011 Proof.
1012 unfold Delta2Gamma. intros. apply all_some_length in H. rewrite map_length in
    ↪ H. assumption.
1013 Qed.
1014
1015 Lemma Delta2Gamma_nth {rho Delta Gamma}: forall (HD2G: Delta2Gamma rho Delta =
    ↪ Some Gamma)
1016
1017 x (pr : x < length Delta),
    P rho (nth_ok Delta x pr) = Some (nth_ok Gamma x (rew (Delta2Gamma_length
    ↪ HD2G) in pr)).
1018 Proof.
1019 intros.
1020 intros. unfold Delta2Gamma in HD2G.
1021 assert (In (P rho (nth_ok Delta x pr)) (map (P rho) Delta)).
1022 {
1023 apply map_in. apply nth_ok_in.
1024 }
1025 pose proof (all_some_nth _ _ HD2G).
1026 erewrite (nth_ok_proof_irel _ Gamma).
1027 rewrite ← H0. rewrite nth_ok_map.
1028 erewrite nth_ok_proof_irel. reflexivity. Unshelve.
1029 rewrite map_length. assumption.
1030 Qed.
1031
1032 Lemma sfc_to_long_subj {rho} : forall Delta R m pi (base_sfc : Sfc Delta R m pi)
    ↪ Gamma,
1033
1034 Delta2Gamma rho Delta = Some Gamma →
    forall pr (base_long : nfty_long Gamma m (P_ok rho pi pr))
1035 Delta' m' pi' (subj_sfc : Sfc Delta' R m' pi'),
1036 Sfc_subj _ _ _ _ _ _ subj_sfc base_sfc →
1037 { Gamma' & prod (Delta2Gamma rho Delta' = Some Gamma')
1038 { pr' & nfty_long Gamma' m' (P_ok rho pi' pr')}}}.
1039 Proof.
1040 intros.
1041 generalize dependent Gamma.
1042 generalize dependent pr.
1043 induction X.
1044 - intros. exists Gamma. split. assumption. exists pr. assumption.
1045 - intros. pose proof (IHx2 pr Gamma H base_long) as [Gamma' [HD2G [pr'
    ↪ nfty1]]].
1046 pose proof (IHx1 pr' Gamma' HD2G nfty1) as [Gamma'' [HD2G' [pr'' nfty2]]].
1047 exists Gamma''. split. assumption. exists pr''. assumption.
1048 - intros.
1049 assert (In (pi ++ [Src]) (dom_P rho)) as prSrc.
1050 {
1051 eapply P_lam_proof_Src. exact base_long.
1052 }
1053 exists (P_ok rho (pi ++ [Src]) prSrc :: Gamma). split.

```

```

1054 { apply Delta2Gamma_pump.
1055   - remember (P_ok rho (pi ++ [Src]) prSrc). eapply P_ok_P. symmetry. exact
      ↪ Heqt.
1056   - assumption.
1057 }
1058 pose proof (P_lam_step m pi Delta R (SfC_I _ _ _ proof) rho Gamma H pr) as
      ↪ HlamStep.
1059 pose proof (dom_P_Src_to_Tgt _ _ _ Tgt prSrc) as prTgt.
1060 exists prTgt.
1061 assert (forall prSrc' pr'', nfty_long (P_ok rho (pi ++ [Src]) prSrc' ::
      ↪ Gamma) m (P_ok rho (pi ++ [Tgt]) pr''))
1062       → nfty_long (P_ok rho (pi ++ [Src]) prSrc
      ↪ :: Gamma) m (P_ok rho (pi ++ [Tgt])
      ↪ prTgt)).
1063 {
1064   intros. rewrite P_ok_proof_irl with (p2 := prSrc').
1065   rewrite P_ok_proof_irl with (p2 := pr''). assumption.
1066 }
1067 eapply X. clear X.
1068 eapply HlamStep. Unshelve. erewrite (P_ok_proof_irl). apply base_long.
1069 - intros. inversion base_long.
1070 pose proof (SfC_E Delta R ms pi pi' x deltaok Rproof proofs) as SfC_base.
1071 exists Gamma.
1072 assert (P rho pi = Some (make_arrow_type ts (P_ok rho pi' pr))).
1073 {
1074   rewrite ← H3. rewrite ← Gammaok.
1075   revert H. revert deltaok. clear ...
1076   intros. apply nth_error_nth_ok in deltaok. destruct deltaok as [pr
      ↪ deltaok].
1077   pose proof (Delta2Gamma_nth H x pr). remember (nth_ok Gamma x _). symmetry
      ↪ in Heqt.
1078   apply nth_ok_nth_error in Heqt. subst. rewrite ← Heqt in H0. exact H0.
1079 } subst.
1080 pose proof (
1081   P_app_step SfC_base rho Gamma pi pr ts a deltaok Gammaok base_long H0
      ↪ Lenproof n ltproof
1082   ). split. assumption.
1083 eexists. apply X0.

```

1084 **Qed**.

```

1085
1086 Lemma Delta2Gamma_nth_error : forall Delta Gamma rho, Delta2Gamma rho Delta =
      ↪ Some Gamma →
1087       forall x pi tau, nth_error Delta x = Some pi →
      ↪ nth_error Gamma x = Some (tau) →
1088       {pr & P_ok rho pi pr = tau}.

```

1089 **Proof**.

```

1090   intros.
1091   pose proof (nth_error_nth_ok _ _ _ H0) as [pr H0_ok].
1092   pose proof (nth_error_nth_ok _ _ _ H1) as [pr' H1_ok].
1093   pose proof (Delta2Gamma_nth H x pr). rewrite H0_ok in H2.
1094   apply P_P_ok_set in H2. destruct H2 as [pr0 H2].
1095   rewrite (nth_ok_proof_irel _ _ _ pr') in H2. rewrite H1_ok in H2.
1096   exists pr0. assumption.

```

1097 **Qed**.

1098

```

1099 Lemma sfc_app_subj_types_atomic {rho} : forall R m someDelta somepi (base_sfc :
    ↪ Sfc someDelta R m somepi)
1100     someGamma somepr (base_long : nfty_long someGamma m (P_ok rho somepi
    ↪ somepr))
1101     (someD2G: Delta2Gamma rho someDelta = Some someGamma)
1102     Delta x pi (Deltaok : nth_error Delta x = Some pi)
1103     ms pi' (subj_sfc : Sfc Delta R (!! x @@ ms) pi'),
1104     Sfc_subj _ _ _ _ _ subj_sfc base_sfc →
1105     { a & P rho (pi ++ repeat Tgt (length ms)) = Some (? a) /\ P
    ↪ rho (pi') = Some (? a) }.

1106 Proof.
1107   intros.
1108   pose proof (sfc_to_long_subj _ R m _ base_sfc _ someD2G somepr base_long Delta
    ↪ (!! x @@ ms) pi' subj_sfc X)
1109     as [Gamma [D2G [pr nfty]]].
1110   inversion nfty. rewrite Lenproof in *. subst. exists a. split.
1111   - rewrite H.
1112     pose proof (Delta2Gamma_nth_error _ _ _ D2G _ _ _ Deltaok Gammaok) as [D2Gpr
    ↪ Hpirho].
1113     rewrite ← H.
1114     pose proof (P_ok_make_arrow ts (? a)) as [mkpr HPmk].
1115     eapply P_app_split.
1116     + eapply P_ok_P. exact Hpirho.
1117     + eapply P_ok_P. exact HPmk.
1118   - eapply P_ok_P. symmetry. exact H.
1119 Qed.

1120
1121 Lemma sfc_app_subj_R_tau_cond {rho m R Delta x pi ms pi'} :
1122   forall (base_sfc : Sfc [] R m []) (base_long : nfty_long [] m rho)
1123     (subj_sfc : Sfc Delta R (!! x @@ ms) pi')
1124     (Deltaok : nth_error Delta x = Some pi),
1125     Sfc_subj _ _ _ _ _ subj_sfc base_sfc →
1126     R_tau_cond rho (pi ++ repeat Tgt (length ms), pi') = true.

1127 Proof.
1128   intros.
1129   pose proof (sfc_app_subj_types_atomic _ _ _ _ base_sfc _ (dom_P_nil _))
    ↪ base_long eq_refl _ _ _ Deltaok _ _ subj_sfc X) as [a [Hpi Hpi']].
1130   unfold R_tau_cond. simpl. rewrite Hpi. rewrite Hpi'. apply andb_true_intro.
1131   split; try (apply equivb_prop; reflexivity).
1132   pose proof (r_not_refl base_sfc).
1133   pose proof (each_judg_subj_Sfc_P _ _ _ _ _ H _ _ _ _ X). unfold
    ↪ r_not_refl_cond in H0. simpl in H0.
1134   pose proof Sfc_gen_app _ _ _ _ _ subj_sfc.
1135   rewrite ← H1 in H0.
1136   pose proof (get_subproof_app_deltaok subj_sfc).
1137   rewrite H2 in Deltaok. apply some_eq in Deltaok. subst.
1138   apply nequivb_prop. assumption.
1139 Qed.

1140
1141 Lemma sfc_app_subj_in_R_tau {rho m R} :
1142   forall (base_sfc : Sfc [] R m []) (base_long : nfty_long [] m rho) Delta x pi
    ↪ (Deltaok : nth_error Delta x = Some pi) ms pi'
1143     (subj_sfc : Sfc Delta R (!! x @@ ms) pi'),
1144     Sfc_subj _ _ _ _ _ subj_sfc base_sfc →
1145     R_tau_ts rho (pi ++ repeat Tgt (length ms)) pi'.

1146 Proof.

```

```

1147 intros.
1148 pose proof (sfc_app_subj_R_tau_cond base_sfc base_long subj_sfc Deltaok X).
1149 pose proof (sfc_app_subj_types_atomic _ _ _ base_sfc _ (dom_P_nil _)
1150   ↪ base_long eq_refl _ _ _ Deltaok _ _ subj_sfc X) as [a [Hpi Hpi']].
1151 unfold R_tau_ts.
1152 unfold R_tau_list.
1153 constructor.
1154 apply filter_In. split.
1155 - apply in_prod.
1156   + apply P_P_ok_set in Hpi as [pr _]. assumption.
1157   + apply P_P_ok_set in Hpi' as [pr _]. assumption.
1158 - assumption.
1159 Qed.
1160
1161 Lemma sfc_replace_R {R m Delta' pi''}: forall (base_sfc: Sfc Delta' R m pi'')
1162   ↪ R',
1163   (forall Delta x pi, nth_error Delta x = Some pi → forall ms pi' (subj_sfc:
1164     ↪ Sfc Delta R (!x @@ ms) pi'),
1165     Sfc_subj _ _ _ _ _ subj_sfc base_sfc → R' (pi ++ repeat Tgt
1166       ↪ (length ms)) pi'))
1167   → Sfc Delta' R' m pi''.
1168
1169 Proof.
1170 intros base_sfc.
1171 induction base_sfc.
1172 - intros.
1173   constructor. apply IHbase_sfc.
1174   intros.
1175   eapply X.
1176   + exact H.
1177   + remember (Sfc_subj_I R Delta s pi base_sfc).
1178   remember (Sfc_subj_trans _ _ _ _ _ _ _ _ _ X0 s0).
1179   apply s1.
1180 - intros. econstructor.
1181   + apply e.
1182   + eapply X0.
1183     * apply e.
1184     * constructor.
1185   + intros. apply X. intros.
1186   eapply X0.
1187     ** apply H.
1188     ** remember (Sfc_subj_E R Delta pi pi' ms x e r s n p).
1189     remember (Sfc_subj_trans _ _ _ _ _ _ _ _ _ X1 s0).
1190     apply s1.
1191 Qed.
1192
1193 Lemma long_to_sfc_tau {rho m} : nfty_long [] m rho →
1194   Sfc [] (R_tau_ts rho) m [].
1195
1196 Proof.
1197 intro base_long.
1198 pose proof (Long_closed _ _ base_long) as Hclosed.
1199 pose proof (closed_Rm _ Hclosed) as base_sfc.
1200 pose proof (sfc_app_subj_in_R_tau base_sfc base_long).
1201 pose proof sfc_replace_R base_sfc _ X.
1202 assumption.
1203 Qed.

```



```

1199 Lemma sfc_tau_to_Rsub_m_tau {m tau} : SfC [] (R_tau_ts tau) m [] → Rsub (R_m_ts
    ↪ m) (R_tau_ts tau).
1200 Proof.
1201   intros. unfold R_m_ts. destruct (R_m m) eqn:HRm.
1202   - apply R_m_ts_minimal with (Rm':=l) in X.
1203     + unfold Rsub in *. intros. unfold R_tau_ts.
1204       induction X0.
1205       * exact (X _ _ i).
1206       * econstructor 2. exact IHX0.
1207       * econstructor 3. exact IHX0_1. exact IHX0_2.
1208     + exact HRm.
1209   - unfold Rsub. intros. inversion H.
1210 Qed.
1211
1212 Lemma pi_in_R : forall m Delta pi R, SfC Delta R m pi → {pi' & {app & R pi' (pi
    ↪ ++ app)}}}.
1213 Proof.
1214   induction 1.
1215   - destruct IHX as [pi' [app IHX]]. exists pi'. exists ([Tgt] ++ app). rewrite
    ↪ app_assoc. assumption.
1216   - exists (pi ++ repeat Tgt (length ms)). exists []. rewrite app_nil_r.
    ↪ assumption.
1217 Qed.
1218
1219 Lemma R_tau_ts_dom_P {tau pi pi'} : R_tau_ts tau pi pi' → {a & P tau pi = Some
    ↪ (? a) /\ P tau pi' = Some (? a)}.
1220 Proof.
1221   intros.
1222   induction X.
1223   - unfold R_tau_list in i. apply filter_In in i as [i x]. unfold R_tau_cond in
    ↪ X.
1224     simpl in x. apply andb_prop in x. destruct x as [nab pt]. destruct (P tau a)
    ↪ eqn:HPa; try discriminate pt.
1225     destruct t eqn:Htl ; try discriminate pt. destruct (P tau b) eqn:HPb; try
    ↪ discriminate pt.
1226     exists x. rewrite equivb_prop in pt. subst. auto.
1227   - ainv. exists x. split; assumption.
1228   - destruct IHX1 as [x1 [Pa Pb]]. exists x1.
1229     destruct IHX2 as [x2 [Pa1 Pb2]]. assert (x1 = x2). rewrite Pa1 in Pb.
    ↪ injection Pb. auto. subst. auto.
1230 Qed.
1231
1232 Lemma Delta2Gamma_x {rho Delta Gamma x pi}: Delta2Gamma rho Delta = Some Gamma
    ↪ →
1233       nth_error Delta x = Some pi →
1234       {pr & nth_error Gamma x = Some (P_ok rho pi pr)}.
1235 Proof.
1236   intros.
1237   pose proof nth_error_Some3 _ _ _ H0 as lenpr.
1238   pose proof Delta2Gamma_nth H x lenpr as Prhoeq.
1239   pose proof nth_error_nth_ok _ _ _ H0 as [pr Hnthok].
1240   erewrite nth_ok_proof_irel in Hnthok.
1241   rewrite Hnthok in Prhoeq.
1242   pose proof P_P_ok_set Prhoeq as [pr0 HPok].
1243   exists pr0. rewrite HPok. remember (nth_ok Gamma x _).
1244   symmetry in Heqt.

```

```

1245 rewrite nth_ok_nth_error in Heqt. assumption.
1246 Qed.
1247
1248 Lemma Rsub_m_tau_to_Long_aux {m tau} : forall Delta pi R, Sfc Delta R m pi →
1249 forall Gamma, Delta2Gamma
1250   → tau Delta = Some Gamma
1251   →
1252   Rsub R (R_tau_ts
1253     → tau) →
1254   {pr & nfty_long
1255     → Gamma m (P_ok
1256       → tau pi pr)}.
1257
1258 Proof.
1259 induction 1.
1260 - intros. unfold Rsub in X0.
1261 pose proof pi_in_R _ _ _ X as [pi' [appl HR]].
1262 pose proof X0 _ _ HR as HRtau.
1263 assert ({rho & {pr & P_ok tau (pi ++ [Src]) pr = rho}}) as [rho [pr HPok]].
1264   → {
1265     pose proof R_tau_ts_dom_P HRtau. destruct H0 as [a [HPpi HPpi']].
1266     pose proof P_prefix HPpi' as [tau' HPpi_Tgt].
1267     pose proof P_P_ok_set HPpi_Tgt as [pr HPok].
1268     pose proof P_ok_Src_to_Tgt _ _ _ Src _ _ HPok as [pr' [rho HP]].
1269     exists rho. exists pr'. assumption.
1270   }
1271 pose proof proj1 P_ok_P HPok as HP.
1272 pose proof IHX (rho :: Gamma) (Delta2Gamma_pump _ _ _ _ HP _ H) X0 as [prTgt
1273   → nfbase].
1274 pose proof P_Src2 _ _ _ HP as [tau' [HPbase HPTgt]].
1275 pose proof P_P_ok_set HPbase as [pr' HPokbase].
1276 exists pr'. rewrite HPokbase.
1277 constructor.
1278 pose proof P_P_ok_set HPTgt as [pr'' HPoktgt].
1279 rewrite ← HPoktgt. erewrite P_ok_proof_irl. exact nfbase.
1280 - intros. unfold Rsub in X0.
1281 pose proof X0 _ _ r as HRtau.
1282 pose proof R_tau_ts_dom_P HRtau as [a [Ppi Ppi']].
1283 pose proof P_P_ok_set Ppi' as [pr Pokpi'].
1284 exists pr. rewrite Pokpi'. pose proof P_path_make_arrow_type Ppi as [ts
1285   → [HPTs HLeneq]].
1286 econstructor.
1287 + pose proof Delta2Gamma_x H e as [pr' HGamma].
1288 erewrite ← P_ok_P in HPTs. rewrite HPTs in HGamma. exact HGamma.
1289 + intros.
1290 pose proof X n pms Gamma H X0 as [pr' Hnft].
1291 erewrite (nth_ok_proof_irel n ts).
1292 assert (P_ok tau (make_tgt_path pi n) pr' = nth_ok ts n (rew ← HLeneq in
1293   → pms)). {
1294   remember (make_arrow_type ts (? a)) as rho.
1295   symmetry in Heqrho.
1296   pose proof make_arrow_type_dirs Heqrho (n:=n).
1297   destruct (nth_error ts n) eqn:Hnthts.
1298   - epose proof P_app_split HPTs H0.
1299     rewrite P_ok_P. unfold make_tgt_path. rewrite H1.
1300     rewrite ← some_eq. symmetry. apply nth_ok_nth_error. assumption.
1301   - apply nth_error_None in Hnthts.

```

```

1292         pose proof lt_not_le _ _ (rew ← HLeneg in pms). contradiction.
1293
1294     }
1295     rewrite ← H0.
1296     assumption. Unshelve. symmetry. assumption.
1297 Qed.
1298
1299 Lemma Rsub_m_tau_to_Long {m tau} : closed m → Rsub (R_m_ts m) (R_tau_ts tau) →
1300   ⇔ nfty_long [] m tau.
1301 Proof.
1302   intros Hclosed.
1303   pose proof (closed_Rm _ Hclosed) as base_sfc.
1304   intros.
1305   pose proof (Rsub_m_tau_to_Long_aux _ _ _ base_sfc [] eq_refl X) as [pr nf].
1306   simpl in nf. assumption.
1307 Qed.
1308
1309 Lemma long_to_Rsub_m_tau {m tau} : nfty_long [] m tau → Rsub (R_m_ts m)
1310   ⇔ (R_tau_ts tau).
1311 Proof.
1312   intros.
1313   apply sfc_tau_to_Rsub_m_tau.
1314   apply long_to_sfc_tau.
1315   assumption.
1316 Qed.
1317
1318 Lemma R_m_ts_dec : forall m pi pi', (R_m_ts m pi pi') + (R_m_ts m pi pi' →
1319   ⇔ False).
1320 Proof.
1321   intros.
1322   unfold R_m_ts.
1323   destruct (R_m m).
1324   + apply ts_cl_list_dec.
1325   + right. intros F. inversion F.
1326 Defined.
1327
1328 Lemma R_tau_ts_dec : forall m pi pi', (R_tau_ts m pi pi') + (R_tau_ts m pi pi'
1329   ⇔ → False).
1330 Proof.
1331   intros.
1332   unfold R_tau_ts.
1333   apply ts_cl_list_dec.
1334 Defined.
1335
1336 Definition replaceable_paths_cond m pi pi' : bool :=
1337   if (R_m_ts_dec m pi pi') then true else
1338     if (pi = pi') then true else false.
1339
1340 Definition replaceable_paths tau m pi : list path :=
1341   filter (replaceable_paths_cond m pi) (dom_P tau).
1342
1343 Definition replace_all_paths (tau: type)(pis : list path) (b : type) :=
1344   fold_left (replace_at_path b) pis tau.
1345
1346 Lemma replace_at_nil : forall b tau, replace_at_path b tau [] = b.
1347 Proof.

```

```

1344   induction tau; reflexivity.
1345 Qed.
1346
1347 Lemma replace_all_var: forall pis b,
1348   replace_all_paths (? b) pis (? b) = ? b.
1349 Proof.
1350   induction pis.
1351   - reflexivity.
1352   - intros. induction a.
1353     + simpl. apply IHpis.
1354     + destruct a.
1355       * simpl. apply IHpis.
1356       * simpl. apply IHpis.
1357 Qed.
1358
1359 Lemma replace_all_var_is_var: forall pis b c,
1360   {d & replace_all_paths (? c) pis (? b) = ? d}.
1361 Proof.
1362   induction pis.
1363   - eexists. reflexivity.
1364   - intros. induction a.
1365     + simpl. apply IHpis.
1366     + destruct a.
1367       * simpl. apply IHpis.
1368       * simpl. apply IHpis.
1369 Qed.
1370 (*
1371 Lemma nil_in_replace_paths2 : forall pis tau b, replace_all_paths tau pis (? b)
1372   ⇔ = (? b) → {In [] pis} + {tau = (? b)}.
1373 Proof.
1374   induction pis.
1375   - intros. destruct tau.
1376     + simpl in H. right. assumption.
1377     + simpl in H. discriminate H.
1378   - intros. simpl in H. destruct (tau = ? b). right. assumption.
1379     destruct a. left. constructor. reflexivity.
1380     apply IHpis in H.
1381     destruct H.
1382     + left. constructor 2. assumption.
1383     + simpl in e.
1384
1385     destruct ((replace_at_path (? b) tau a)) eqn:Hr.
1386       * rewrite e in *. left. constructor 2.
1387
1388     destruct (IHpis tau b).
1389     +
1390   + intros. destruct H.
1391     * subst. simpl. apply replace_all_var.
1392     * simpl. destruct a.
1393       -- simpl. apply replace_all_var.
1394       -- destruct d.
1395         ++ simpl. apply IHpis. assumption.
1396         ++ simpl. apply IHpis. assumption.
1397 Qed.*)
1398

```

```

1399 Lemma nil_in_replace_paths : forall pis tau b, In [] pis → replace_all_paths
    ↪ tau pis (? b) = (? b).
1400 Proof.
1401   induction pis.
1402   + ainv.
1403   + intros. destruct H.
1404     * subst. simpl. apply replace_all_var.
1405     * simpl. destruct a.
1406       -- simpl. apply replace_all_var.
1407       -- destruct d.
1408         ++ simpl. apply IHpis. assumption.
1409         ++ simpl. apply IHpis. assumption.
1410 Qed.
1411
1412 Lemma lt_S_l_max x y: x ≤ S (Init.Nat.max x y).
1413 Proof.
1414   destruct (Nat.max_dec x y).
1415   - rewrite e. omega.
1416   - rewrite e. apply Nat.max_r_iff in e. omega.
1417 Qed.
1418
1419 Lemma lt_S_r_max x y: x ≤ S (Init.Nat.max y x).
1420 Proof.
1421   destruct (Nat.max_dec y x).
1422   - rewrite e. apply Nat.max_l_iff in e. omega.
1423   - rewrite e. omega.
1424 Qed.
1425
1426 Lemma no_path_to_fresh : forall pi x tau, first_fresh_type tau ≤ x → P tau pi
    ↪ = Some (? x) → False.
1427 Proof.
1428   induction pi.
1429   - ainv. asimpl in H. omega.
1430   - ainv. destruct a; destruct tau; ainv.
1431     + assert (first_fresh_type tau1 ≤ x).
1432       { transitivity (first_fresh_type (tau1 ↪ tau2)).
1433         - apply lt_S_l_max.
1434         - assumption.
1435       }
1436     + apply (IHpi _ _ H0 H1).
1437   + assert (first_fresh_type tau2 ≤ x).
1438     { transitivity (first_fresh_type (tau1 ↪ tau2)).
1439       - apply lt_S_r_max.
1440       - assumption.
1441     }
1442     + apply (IHpi _ _ H0 H1).
1443 Qed.
1444
1445 Lemma no_path_to_first_fresh : forall pi tau, P tau pi = Some (fresh_type tau)
    ↪ → False.
1446 Proof.
1447   intros.
1448   destruct (fresh_type tau) eqn:Hft.
1449   - unfold fresh_type in Hft. ainv.
1450     + apply (no_path_to_fresh pi (first_fresh_type tau) tau (Nat.le_refl _) H1).
1451   - ainv.

```

```

1452 Qed.
1453
1454 Lemma ts_cl_list_nil {A} : forall (pi pi' :A), ts_cl_list [] pi pi' → False.
1455 Proof.
1456   intros.
1457   induction X;ainv.
1458 Qed.
1459
1460 Lemma R_tau_replace_one tau pi tau' a pr :
1461   P_ok tau pi pr = ? a →
1462   replace_at_path (fresh_type tau) tau pi = tau' →
1463   R_tau_list tau = R_tau_list tau'.
1464 Proof.
1465   intros.
1466   unfold R_tau_list. Abort.
1467
1468 Lemma P_replace_at_path : forall tau pi b pr, P_ok (replace_at_path b tau pi)
1469   ⇔ pi pr = b.
1470 Proof.
1471   induction tau.
1472   - intros. asimpl in pr. destruct pi.
1473     + reflexivity.
1474     + destruct d; ainv.
1475   - asimpl. intros. destruct pi.
1476     + reflexivity.
1477     + destruct d.
1478       * simpl. apply IHtau1.
1479       * simpl. apply IHtau2.
1480 Qed.
1481
1482 (* General *)
1483 Lemma filter_NoDup {A} : forall (l : list A) fb, NoDup l → NoDup (filter fb l).
1484 Proof.
1485   induction l.
1486   - intros. simpl. assumption.
1487   - intros. simpl. destruct (fb a).
1488     + inversion H. subst. constructor.
1489       * intros F. apply filter_In in F. destruct F. contradiction.
1490       * apply IHl. assumption.
1491     + inversion H. apply IHl. assumption.
1492 Qed.
1493
1494 Lemma NoDup_map_cons_iff {A} : forall l (x : A), NoDup (map (cons x) l) ⇔
1495   ⇔ NoDup l.
1496 Proof.
1497   split.
1498   - intros. eapply NoDup_map_inv. exact H.
1499   - intros. induction l.
1500     + constructor.
1501     + inv H. simpl. constructor.
1502       * intros F. apply in_map_iff in F. destruct F as [x' [Heq HIn]].
1503         ainv. apply H2. constructor. reflexivity.
1504       * apply IHl. assumption.
1505 Qed.
1506
1507

```

```

1505 Lemma NoDup_app_disjoint {A} : forall l l' , (forall (a :A), In a l → ~(In a
    ↪ l')) /\ (forall a, In a l' → ~(In a l)) →
1506 NoDup l → NoDup l' → NoDup (l ++ l').
1507 Proof.
1508   induction l.
1509   - ainv.
1510   - intros. destruct H. simpl. pose proof H a (in_eq _ _).
1511     constructor.
1512   + intros F. inv H0. apply in_app_or in F. destruct F.
1513     * contradiction.
1514     * contradiction.
1515   + apply IHl.
1516     * split.
1517       -- intros. apply H. constructor 2. assumption.
1518       -- intros. apply H2 in H4. intros F. apply H4. constructor 2.
1519         ↪ assumption.
1519     * inv H0. assumption.
1520     * assumption.
1521 Qed.
1522
1523 (* General Paths *)
1524 Lemma dom_P_NoDup tau : NoDup (dom_P tau).
1525 Proof.
1526   induction tau.
1527   - simpl. constructor.
1528   + intros F. inv F.
1529   + constructor.
1530   - simpl. constructor.
1531   + intros F. apply in_app_or in F. destruct F.
1532     * apply in_map_iff in H. destruct H as [x [F _]].
1533       discriminate F.
1534     * apply in_map_iff in H. destruct H as [x [F _]].
1535       discriminate F.
1536   + apply NoDup_app_disjoint.
1537     * split; intros.
1538       -- apply in_map_iff in H. destruct H as [x [Heq HIn]]. subst. intros F.
1539         ↪ apply in_map_iff in F.
1540         destruct F as [x0[Heq Hin]]. discriminate Heq.
1541       -- apply in_map_iff in H. destruct H as [x [Heq HIn]]. subst. intros F.
1542         ↪ apply in_map_iff in F.
1543         destruct F as [x0[Heq Hin]]. discriminate Heq.
1543     * apply NoDup_map_cons_iff. assumption.
1544     * apply NoDup_map_cons_iff. assumption.
1545 Qed.
1546
1547 Lemma replaceable_nodup : forall tau m pi, NoDup (replaceable_paths tau m pi).
1548 Proof.
1549   intros.
1550   apply filter_NoDup.
1551   apply dom_P_NoDup.
1552 Qed.
1553
1554 Lemma replace_all_arrow : forall pis tau sigma b, (~ In [] pis) →

```

```

1555                                     {tau' & {sigma' & replace_all_paths (tau
                                         ↪ ↪ sigma) pis (? b) = tau' ↪
                                         ↪ sigma'}}}.

1556 Proof.
1557   induction pis.
1558   - intros. simpl. eexists. eexists. reflexivity.
1559   - intros. induction a.
1560     + exfalso. apply H. constructor. reflexivity.
1561     + destruct a.
1562       * simpl. apply IHpis. intros F. apply H.
1563         constructor 2. assumption.
1564       * simpl. apply IHpis. intros F. apply H.
1565         constructor 2. assumption.

1566 Qed.
1567 (*)
1568 Lemma replace_all_arrow2 : forall pis tau sigma b, {tau' & {sigma' &
    ↪ replace_all_paths (tau ↪ sigma) pis (? b) = tau' ↪ sigma'}} +
1569                                     {replace_all_paths (tau ↪ sigma) pis (?
    ↪ b) = tau ↪ sigma} +
1570                                     {replace_all_paths (tau ↪ sigma) pis (?
    ↪ b) = (? b)}.

1571 Proof.
1572   induction pis.
1573   - intros. simpl. left. right. reflexivity.
1574   - intros. destruct a.
1575     + asimpl. right. pose proof replace_all_var. unfold replace_all_paths in H.
    ↪ apply H.
1576     + destruct (IHpis tau sigma b).
1577       * admit.
1578       * right. simpl.

1579     destruct d.
1580     * simpl. edestruct IHpis.
1581       apply IHpis. intros F. apply H.
1582       constructor 2. assumption.
1583     * simpl. apply IHpis. intros F. apply H.
1584       constructor 2. assumption.

1585 Qed. *)

1587 Inductive is_prefix {A} : list A → list A → Type :=
1588 | is_prefix_direct l a : is_prefix [] (a::l)
1589 | is_prefix_cons l l' a : is_prefix l l' → is_prefix (a::l) (a::l').

1591 Lemma is_prefix_nil_l {A} (l : list A) : is_prefix [] l + {l = []}.
1592 Proof.
1593   destruct l.
1594   - right. reflexivity.
1595   - left. constructor.

1597 Defined.

1598 Lemma is_prefix_dec {A} {eqdec : EqDec A eq} : forall (l l' : list A), is_prefix
    ↪ l l' + (is_prefix l l' → False).

1600 Proof.
1601   induction l.
1602   - intros. destruct (is_prefix_nil_l l').
1603     + left. assumption.

```



```

1604   + subst. right. intros F. inv F.
1605   - intros. destruct l'.
1606   + right. intros. inv X.
1607   + destruct (a = a0).
1608     * rewrite e. destruct (IHL l').
1609     -- left. constructor. assumption.
1610     -- right. intros. inv X. contradiction.
1611     * right. intros. inv X. apply c. reflexivity.
1612 Defined.
1613
1614 Lemma prefix_in_list_dec {A} {eqdec : EqDec A eq} : forall (l : list (list A))
1615   ⇨ pi,
1616   {pi' & (In pi' l * is_prefix pi' pi)%type} + {forall pi', (In pi' l *
1617     ⇨ is_prefix pi' pi)%type → False}.
1618
1619 Proof.
1620   induction l.
1621   - intros. right. intros pi' [Hin Hpr]. inv Hin.
1622   - intros. destruct (IHL pi) as [[pi' [Hin Hpr]]|Hpr].
1623     + left. exists pi'. split. constructor 2. assumption. assumption.
1624     + destruct (is_prefix_dec a pi).
1625       * left. exists a. split. constructor. reflexivity. assumption.
1626       * right. intros pi' [[Heq|Hin] Hpr'].
1627         -- subst. contradiction.
1628         -- apply Hpr with (pi' := pi'). split; assumption.
1629
1630 Qed.
1631
1632 Lemma replace_prefix : forall pi' pi tau b, is_prefix pi' pi → P
1633   ⇨ (replace_at_path (? b) tau pi') pi = None.
1634
1635 Proof.
1636   induction pi'.
1637   - intros. inv H. simpl. destruct a; reflexivity.
1638   - intros. simpl. destruct a.
1639     + destruct tau.
1640       * inv H. reflexivity.
1641       * inv H. simpl. apply IHpi'. assumption.
1642     + destruct tau.
1643       * inv H. reflexivity.
1644       * inv H. simpl. apply IHpi'. assumption.
1645
1646 Qed.
1647
1648 Lemma replace_not_in_dom : forall pi tau,
1649   ~ In pi (dom_P tau) → forall pi' b, ~ In pi (dom_P (replace_at_path (? b)
1650     ⇨ tau pi')).
1651
1652 Proof.
1653   intros.
1654   intros F.
1655   apply H.
1656   clear H.
1657   generalize dependent tau.
1658   revert pi.
1659   induction pi'.
1660   - intros. simpl in F. destruct F. subst. apply dom_P_nil. ainv.
1661   - intros. asimpl in F. destruct a.
1662     + destruct tau.
1663       * assumption.
1664       * simpl. simpl in F. destruct F.

```

```

1656 -- left. assumption.
1657 -- right. apply in_app_or in H. apply in_or_app.
1658 destruct H.
1659 ++ left. destruct pi.
1660     ** exfalso. induction (dom_P (_)).
1661         --- ainv.
1662         --- simpl in H. destruct H.
1663             +++ discriminate H.
1664             +++ contradiction.
1665     ** destruct d.
1666         --- apply in_map_cons_iff. apply in_map_cons_iff in H. apply
1667             ↪ IHpi'. assumption.
1668         --- induction (dom_P (_)).
1669             +++ ainv.
1670             +++ simpl in H. destruct H.
1671                 *** discriminate H.
1672                 *** apply IHl. assumption.
1673 + destruct tau.
1674 * assumption.
1675 * simpl. simpl in F. destruct F.
1676 -- left. assumption.
1677 -- right. apply in_app_or in H. apply in_or_app.
1678 destruct H.
1679 ++ left. assumption.
1680 ++ right. destruct pi.
1681     ** exfalso. induction (dom_P (_)).
1682         --- ainv.
1683         --- simpl in H. destruct H.
1684             +++ discriminate H.
1685             +++ contradiction.
1686     ** destruct d.
1687         --- induction (dom_P (_)).
1688             +++ ainv.
1689             +++ simpl in H. destruct H.
1690                 *** discriminate H.
1691                 *** apply IHl. assumption.
1692         --- apply in_map_cons_iff. apply in_map_cons_iff in H. apply
1693             ↪ IHpi'. assumption.
1693 Qed.
1694
1695 Lemma replace_at_path_var : forall pi a b, {c & replace_at_path (? a) (? b) pi =
1696     ↪ ? c}.
1697 Proof.
1698     destruct pi.
1699     - intros. eexists. simpl. reflexivity.
1700     - intros. eexists. simpl. destruct d; reflexivity.
1701 Qed.
1702
1703 Definition replace_all_paths2 tau pis b := fold_right (fun pi tau ⇒
1704     ↪ replace_at_path b tau pi) tau pis.
1705 Lemma sz_subst_is_fresh : forall pi' tau m pi pr,
1706     R_m_ts m pi pi' →
1707     P_ok (replace_all_paths tau (replaceable_paths tau m pi) (fresh_type tau))
1708     ↪ pi' pr = fresh_type tau.
1709 Proof.

```

```

1707 (*induction pi'.
1708 - intros. asimpl. unfold replace_all_paths.
1709   assert (In [] (replaceable_paths tau m pi)). {
1710     unfold replaceable_paths. apply filter_In. split.
1711     - apply dom_P_nil.
1712     - unfold replaceable_paths_cond. destruct (R_m_ts_dec m pi []).
1713       + reflexivity.
1714       + contradiction.
1715   }
1716   eapply nil_in_replace_paths in H. exact H.
1717 - intros. unfold fresh_type in *. destruct a.
1718   + destruct tau.
1719     * pose proof replace_all_var_is_var (replaceable_paths (? x) m pi)
1720   ↪ (first_fresh_type (? x)) x as [d HR].
1721     pose proof pr as pr2. rewrite HR in pr2. simpl in pr2. destruct pr2;
1722   ↪ ainv.
1723     * destruct (in_dec (list_eqdec dir_eqdec) [] (replaceable_paths (tau1 ↪
1724   ↪ tau2) m pi)).
1725     -- exfalso. eapply nil_in_replace_paths in i. rewrite i in pr. simpl in
1726   ↪ pr. destruct pr; ainv.
1727     --
1728     epose proof replace_all_arrow _ tau1 tau2 _ n as [tau' [sigma' Heq]].
1729   ↪ rewrite P_ok_P.
1730     rewrite Heq. simpl.
1731     destruct (replaceable_paths (tau1 ↪ tau2) m pi = []).
1732     ++ rewrite e in *. simpl in Heq. injection Heq. intros. subst.
1733     simpl.
1734     ainv.
1735     unfold replace_all_paths.
1736     unfold replaceable_paths.
1737     unfold replaceable_paths_cond.
1738     asimpl. ainv. destruct a.
1739     + simpl. iapply IHpi'.
1740     asimpl.*)
1741   Admitted.
1742
1743
1744 Lemma sechsundzwanzig_aux : forall tau pi m,
1745   Rsub (R_m_ts m) (R_tau_ts tau) →
1746   Rsub (R_m_ts m) (R_tau_ts (replace_all_paths tau (replaceable_paths tau m pi)
1747   ↪ (fresh_type tau))).
1748 Proof.
1749 (*
1750   unfold Rsub.
1751   intros.
1752   unfold R_m_ts in *.
1753   destruct (R_m m) eqn:HRmm.
1754   - pose proof (ts_cl_list_dec l pi).
1755     destruct (X1 pi0).
1756     + assert (ts_cl_list l pi pi').
1757     {

```

```

1757     econstructor 3. apply t. assumption.
1758   }
1759   assert (P (replace_all_paths tau (replaceable_paths tau m pi) (fresh_type
↪ tau)) pi0 =
1760     P (replace_all_paths tau (replaceable_paths tau m pi) (fresh_type
↪ tau)) pi').
1761   {
1762   }
1763   }
1764   - ainv.
1765
1766   pose proof (rdec pi).
1767   destr*)
1768   Admitted.
1769 Lemma sechsundzwanzig : forall m tau pi pr a, nfty_long [] m tau → P_ok tau pi
↪ pr = ? a →
1770
1771                                     nfty_long [] m (replace_all_paths tau
↪ (replaceable_paths tau m pi)
↪ (fresh_type tau)).
1772
1773 Proof.
1774   (*
1775   intros.
1776   pose proof long_to_sfc_taui X.
1777   assert (Rsub (R_m_ts m) (R_tau_ts (replace_all_paths tau (replaceable_paths
↪ tau m pi) (fresh_type tau))))).
1778   {
1779     ainv.
1780     revert pi pr a H X0.
1781     induction X.
1782     - ainv. asimpl. unfold Rsub. intros. unfold replaceable_paths_cond.
1783       unfold Rsub in X0.
1784
1785     apply Long_closed in X.
1786     revert m X0 X.
1787     induction tau.
1788     - asimpl. ainv.
1789       destruct m.
1790       + asimpl in H1. ainv. prooflater.
1791       + prooflater.
1792     -
1793
1794     unfold max_fvar in H1. inversion H1. unfold R_tau_ts in X0. asimpl in
↪ X0.
1795     unfold Rsub in X0. unfold fresh_type. asimpl.
1796     unfold replaceable_paths_cond. simpl. unfold R_m_ts_dec.
1797     unfold R_m_ts in X0.
1798     destruct (R_m m).
1799     destruct m.
1800     simpl dom_P in pr.
1801     pose proof pr.
1802     apply In_head_set in H.
1803     destruct H.
1804     + subst. ainv. asimpl. unfold R_m_ts in X0.
1805     epose proof (fun pi pi' X ⇒ ts_cl_list_nil pi pi' (X0 pi pi' X)).

```

```

1806
1807 + admit.
1808 + asimpl in *.
1809
1810 assert ({x1 & m = !! x1 @@ []}).
1811 {
1812     destruct m.
1813     - unfold R_m_ts in X0. asimpl in X0. eexists.
1814 }
1815
1816 rewrite nth_error_nil in H1. discriminate H1.
1817 - intros.
1818
1819 ainv.
1820 revert H. admit. (*
1821 clear...
1822 revert pi pr a.
1823 induction tau.
1824 - intros. asimpl. ainv.
1825 - intros. unfold R_tau_list. *)}
1826 pose proof (Long_closed _ _ X) as Hclosed.
1827 pose proof (Rsub_m_tau_to_Long Hclosed X1).
1828 assumption.*)
1829 Admitted.

```