

Types.v

```
1 Require Import Coq.Lists.List.
2 Require Import Coq.Lists.ListSet.
3 Require Import Coq.Arith.Peano_dec.
4 Require Import Coq.Classes.EquivDec.
5 Require Import Coq.Logic.FunctionalExtensionality.
6
7 Require Import Autosubst.Autosubst.
8
9 Require Import PrincInh.Terms.
10 Require Import PrincInh.Utills.
11
12 Import ListNotations.
13
14 Inductive type :=
15 | Atom (x: var)
16 | Arr (A B : type).
17
18 Instance Ids_type : Ids type. derive. Defined.
19 Instance Rename_type : Rename type. derive. Defined.
20 Instance Subst_type : Subst type. derive. Defined.
21 Instance SubstLemmas_type : SubstLemmas type. derive. Defined.
22
23 Notation "'?' x" := (Atom x) (at level 15).
24 Notation "a '→' b" := (Arr a b) (at level 51, right associativity).
25
26 Definition repo := list type.
27
28 Instance eq_dec_type : EqDec type eq.
29 Proof.
30   unfold EqDec.
31   intros x.
32   induction x.
33   - destruct y.
34     + destruct (x = x0); dec_eq.
35     + right. intros H. ainv.
36   - destruct y.
37     + right. intros H. ainv.
38     + destruct (IHx1 y1); dec_eq.
39       { destruct (IHx2 y2); dec_eq. }
40 Defined.
41
42 Definition mtTy {A} : var → option A := fun x ⇒ None.
43
44 Instance eq_dec_option : forall T, EqDec T eq → EqDec (option T) eq.
45 Proof.
46   unfold EqDec.
47   intros.
48   destruct x, y.
49   - destruct (X t t0); dec_eq.
50   - right. isfalse.
51   - right. isfalse.
52   - left. reflexivity.
53 Defined.
54
```

```

55 Lemma is_none_dec {T: Type} : forall (x: option T), {x = None} + { x <math>\Diamond</math> None}.
56 Proof.
57   intros. destruct x.
58   - right. discriminate.
59   - left. reflexivity.
60 Defined.
61
62
63 Definition subst_option (S : var  $\rightarrow$  type) (Gamma : var  $\rightarrow$  option type) (t : var)
   $\hookrightarrow$  : option type :=
64   match Gamma t with
65   | None  $\Rightarrow$  None
66   | Some z  $\Rightarrow$  Some (subst S z)
67   end.
68
69 Definition subst_option_monad (S : var  $\rightarrow$  type) (Gamma : var  $\rightarrow$  option type) :
   $\hookrightarrow$  var  $\rightarrow$  option type :=
70   Gamma  $\rightrightarrows$  (subst S  $\ggg$  Some).
71
72 Lemma subst_option_def : subst_option = subst_option_monad.
73 Proof.
74   unfold subst_option.
75   unfold subst_option_monad.
76   unfold kleisli_option. unfold funcomp. reflexivity.
77 Qed.
78
79 Notation "s .?[ sigma ]" := (subst_option sigma s) (at level 2,
80   sigma at level 200, left associativity, format "s .?[ sigma ]") :
   $\hookrightarrow$  subst_scope.
81
82 Lemma some_eq : forall (T : Type) (a b : T), a = b  $\rightrightarrows$  Some a = Some b.
83 Proof. intros. split.
84   - intros Heq. subst. reflexivity.
85   - intros Heq. ainv.
86 Qed.
87
88 Theorem subst_repo_some : forall (Gamma : repo) (Su : var  $\rightarrow$  type) (a : var)
   $\hookrightarrow$  (tau: type),
89   nth_error Gamma a = Some tau  $\rightarrow$ 
90   nth_error Gamma ..[Su] a = Some tau.[Su].
91 Proof.
92   intros.
93   unfold subst.
94   eapply map_nth_error in H.
95   exact H.
96 Qed.
97
98 Theorem subst_repo_none : forall (Gamma : repo) (Su : var  $\rightarrow$  type) (a : var),
99   nth_error Gamma a = None  $\rightarrow$ 
100   nth_error Gamma ..[Su] a = None.
101 Proof.
102   intros.
103   apply nth_error_None in H.
104   apply nth_error_None.
105   unfold subst.
106   erewrite  $\leftarrow$  map_length in H.

```

```

107   exact H.
108 Qed.
109
110 Theorem subst_repo : forall (Gamma : repo) (Su : var → type) (a : var),
111   nth_error Gamma .. [Su] a = (nth_error Gamma a) .. [Su].
112 Proof.
113   intros.
114   destruct (nth_error Gamma a) eqn:G.
115   - apply subst_repo_some. assumption.
116   - apply subst_repo_none. assumption.
117 Qed.
118
119 Theorem subst_repo_cons : forall (Gamma : repo) (Su : var → type)
120   (A : type),
121   (A.[Su] :: Gamma .. [Su]) = (A :: Gamma) .. [Su].
122 Proof.
123   autosubst.
124 Qed.
125
126 Inductive subformula : type → type → Prop :=
127 | subf_refl : forall rho, subformula rho rho
128 | subf_arrow_l : forall rho sigma tau, subformula rho sigma → subformula rho
129   → (sigma → tau)
130 | subf_arrow_r : forall rho sigma tau, subformula rho tau → subformula rho
131   → (sigma → tau)
132 .
133
134 Theorem subformula_dec : forall x y, {subformula x y} + {~subformula x y}.
135 Proof.
136   intros.
137   destruct (x = y); dec_eq.
138   - left. constructor.
139   - generalize dependent x. induction y; intros.
140     + right. isfalse.
141     + destruct (x = y1); dec_eq.
142       { left. constructor. constructor. }
143       { destruct (IHy1 x); dec_eq.
144         - assumption.
145         - left. constructor. assumption.
146         - destruct (x = y2); dec_eq.
147           + left. constructor 3. constructor.
148           + destruct (IHy2 x); dec_eq.
149             { assumption. }
150             { left. constructor 3. assumption. }
151             { right. isfalse. } }
152
153 Defined.
154
155 Definition single_subst (a: var) (tau: type) : var → type :=
156   fun (y: var) ⇒ if a = y then tau else ? y.
157
158 Definition rel_dom {A B} (ls : list (A * B)) : list A :=
159   map fst ls.
160
161 Definition rel_codom {A B} (ls : list (A * B)) : list B :=
162   map snd ls.

```

```

161
162 Definition not_subf (a : var) (tau : type) :=
163   ~(subformula (? a) tau).
164
165 Theorem not_subf_dec : forall a tau,
166   {~subformula (? a) tau} + {~(~subformula (? a) tau) }.
167 Proof.
168   intros.
169   destruct (subformula_dec (? a) tau).
170   - right. intros F. apply F. assumption.
171   - left. assumption.
172 Defined.
173
174 Fixpoint nth_subformula (n:nat) (rho:type) : option type :=
175   match (n, rho) with
176   | (0, ? x)  $\Rightarrow$  Some (? x)
177   | (0, sigma  $\rightarrow$  tau)  $\Rightarrow$  Some sigma
178   | (Datatypes.S n', ? x)  $\Rightarrow$  None
179   | (Datatypes.S n', sigma  $\rightarrow$  tau)  $\Rightarrow$  nth_subformula n' tau
180   end.
181
182 Definition mk_arrow_option (left : type) (right : option type) : type :=
183   match right with
184   | None  $\Rightarrow$  left
185   | Some x  $\Rightarrow$  left  $\rightarrow$  x
186   end.
187
188 Fixpoint type_init (rho: type) : option type :=
189   match rho with
190   | ? x  $\Rightarrow$  None
191   | sigma  $\rightarrow$  tau  $\Rightarrow$  Some (mk_arrow_option sigma (type_init tau))
192   end.
193
194 Fixpoint type_target (rho: type) : var :=
195   match rho with
196   | ? x  $\Rightarrow$  x
197   | sigma  $\rightarrow$  tau  $\Rightarrow$  type_target tau
198   end.
199
200 Definition split_type_target (rho: type) : (option type * var) :=
201   (type_init rho, type_target rho).
202
203 Example nth_subformula_ex : nth_subformula 2 (? 0  $\rightarrow$  (? 1  $\rightarrow$  (? 0  $\rightarrow$  ? 0)))  $\rightarrow$  (?
204    $\rightarrow$  2  $\rightarrow$  ?0)  $\rightarrow$  ? 3) = Some (? 2  $\rightarrow$  ?0).
205 Proof. reflexivity. Qed.
206
207 Fixpoint flat_length (rho : type) : nat :=
208   match rho with
209   | ? x  $\Rightarrow$  1
210   | sigma  $\rightarrow$  tau  $\Rightarrow$  Datatypes.S (flat_length tau)
211   end.
212
213 Lemma fl_1_iff_var : forall rho, flat_length rho = 1  $\rightarrow$  exists x, rho = ? x.
214 Proof.
215   intros.
216   split.

```

```

216   - intros. destruct rho.
217     + exists x. reflexivity.
218     + simpl in H. ainv. destruct rho2; simpl in H0; inversion H0.
219   - intros. destruct H. subst. reflexivity.
220 Qed.
221
222
223 Definition make_arrow_type (ts : list type) (a : type) :=
224   fold_right Arr a ts.
225
226 Lemma make_arrow_type_ts_is_nil {ts rho a}:
227   make_arrow_type ts rho = (? a) → ts = [] /\ rho = (? a).
228 Proof.
229   destruct ts.
230   - asimpl. auto.
231   - asimpl. intros. discriminate H.
232 Qed.
233
234 Lemma pump_type_target : forall sigma tau, type_target tau = type_target (sigma
235   ↪ ↗ tau).
236 Proof.
237   reflexivity.
238 Qed.
239
240 Lemma subst_var_is_var : forall Su a tau, ? a = tau.[Su] → exists b, tau = ? b.
241 Proof.
242   induction tau.
243   - simpl. intros. exists x. reflexivity.
244   - simpl. intros. inversion H.
245 Qed.
246
247 Lemma subst_make_arrow : forall Su ts x ss, ss = map (subst Su) ts →
248   ↪ make_arrow_type ss (x.[Su])
249   = (make_arrow_type ts x).[Su].
250 Proof.
251   induction ts.
252   - intros. subst. reflexivity.
253   - intros. ainv. simpl. rewrite IHts; reflexivity.
254 Qed.
255
256 Lemma make_arrow_type_last : forall ts t a,
257   make_arrow_type (ts ++ [t]) a =
258   make_arrow_type (ts) (t ↗ a).
259 Proof.
260   unfold make_arrow_type.
261   intros.
262   rewrite ← (rev_involutive ts).
263   rewrite ← (rev_head_last).
264   rewrite fold_left_rev_right.
265   simpl.
266   rewrite ← fold_left_rev_right.
267   reflexivity.
268 Qed.
269
270 Lemma make_arrow_type_head : forall ts t a,
271   make_arrow_type (t :: ts) a =

```

```

270   t ↪ make_arrow_type ts a.
271 Proof.
272   intros. reflexivity.
273 Qed.
274
275 Lemma repo_pump_subst : forall (Gamma : repo) Gamma0 A Su, Gamma = Gamma0..[Su]
  ↪ → (A :: Gamma) = A :: Gamma0..[Su].
276 Proof.
277   intros.
278   subst. try rewrite ← subst_repo_cons.
279   reflexivity.
280 Qed.
281
282 Lemma repo_subst_exists : forall (Gamma : repo) Su x A, (nth_error Gamma..[Su] x
  ↪ = Some A)
283   → exists B, B.[Su] = A /\ nth_error Gamma x = Some B.
284 Proof.
285   intros. destruct (nth_error Gamma x) eqn:Ht.
286   + exists t. rewrite subst_repo in H. rewrite Ht in H. ainv. split;
  ↪ reflexivity.
287   + rewrite subst_repo in H. rewrite Ht in H. ainv.
288 Qed.
289
290 Lemma subst_arr_is_arr_or : forall x t Su t0, x.[Su] = t ↪ t0
291   → (exists st st0,
292     x = st ↪ st0 /\ st.[Su] = t /\ st0.[Su] = t0) \/
293     (exists a, x = ? a).
294 Proof.
295   intros. destruct x.
296   - right. exists x. auto.
297   - left. exists x1. exists x2.
298     split.
299     + reflexivity.
300     + split; ainv.
301 Qed.
302
303 Lemma subst_arr : forall x y Su, x.[Su] ↪ y.[Su] = (x ↪ y).[Su].
304 Proof.
305   reflexivity.
306 Qed.
307
308 Lemma add_arr_head : forall A B B0, B = B0 → A ↪ B = A ↪ B0.
309 Proof.
310   intros. subst. reflexivity.
311 Qed.
312
313 Lemma mkarrow_subst_exists : forall ts x Su a, x.[Su] = make_arrow_type ts (? a)
  ↪ →
314   exists ts0 a0, x = (make_arrow_type ts0 (? a0)).
315 Proof.
316   induction ts.
317   - intros. simpl in H. symmetry in H. apply subst_var_is_var in H. exists [].
  ↪ ainv.
318   - intros. rewrite make_arrow_type_head in H. apply subst_arr_is_arr_or in H as
  ↪ [[st [st0 [xst [xsu stmkarr]]] | xvar].

```

```

319   + apply IHts in stmkarr. inv stmkarr. inv H. exists (st :: x0). exists x.
    ↪ rewrite make_arrow_type_head.
320   reflexivity.
321   + ainv. exists []. exists x0. reflexivity.
322 Qed.
323
324
325 Definition not_in_codom (ls : list (var * type)) (a : var) :=
326   Forall (not_subf a) (rel_codom ls).
327
328 Theorem not_in_codom_dec : forall ls x, {not_in_codom ls x} + {~not_in_codom ls
    ↪ x}.
329 Proof.
330   intros.
331   unfold not_in_codom.
332   apply Forall_dec.
333   apply not_subf_dec.
334 Defined.
335
336 Theorem not_in_codom_tail (ls : list (var * type)) (c : (var * type)) (a : var)
    ↪ :
337   not_in_codom (c :: ls) a → not_in_codom ls a.
338 Proof.
339   ainv.
340 Qed.
341
342 Definition domain_codomain_free ls :=
343   Forall (not_in_codom ls) (rel_dom ls).
344
345 Theorem domain_codomain_free_dec : forall ls, { domain_codomain_free ls } + { ~
    ↪ domain_codomain_free ls }.
346 Proof.
347   intros ls.
348   unfold domain_codomain_free.
349   apply Forall_dec.
350   apply not_in_codom_dec.
351 Defined.
352
353 Definition unique_domain {A B} (ls : list (A * B)) :=
354   NoDup (rel_dom ls).
355
356 Theorem unique_domain_dec {A B : Type} {eq_dec : EqDec A eq} : forall (ls : list (A
    ↪ * B)), {unique_domain ls} + {~unique_domain ls}.
357 Proof.
358   intros ls.
359   unfold unique_domain.
360   induction ls.
361   - left. constructor.
362   - destruct IHls.
363     + destruct (in_dec eq_dec (fst a) (rel_dom ls)).
364       { right. isfalse. }
365       { left. unfold rel_dom. rewrite map_cons. constructor; assumption. }
366     + right. isfalse.
367 Defined.
368
369 Definition correct_context ls :=

```

```

370         unique_domain ls /\
371         domain_codomain_free ls.
372
373 Theorem correct_context_dec : forall ls, {correct_context ls} +
374   ⇨ {~correct_context ls}.
375 Proof.
376   intros.
377   unfold correct_context.
378   destruct (unique_domain_dec ls).
379   - destruct (domain_codomain_free_dec ls).
380     + left. split; assumption.
381     + right. isfalse.
382   - right. isfalse.
383 Defined.
384
385 Fixpoint wrap_lam (n : nat) (m : term) : term :=
386   match n with
387   | 0 ⇒ m
388   | S n ⇒ λ _ (wrap_lam n (rename (+1) m) @ !0)
389   end.
390
391 Fixpoint fv_type (tau: type) : set var :=
392   match tau with
393   | ? a ⇒ [a]
394   | sigma → tau ⇒ set_union (nat_eq_eqdec) (fv_type sigma) (fv_type tau)
395   end.
396
397 Fixpoint subst_len_to_index (ls: list var) (v : var) : var :=
398   match ls with
399   | [] ⇒ v
400   | a :: ls' ⇒ if v = a then 0 else 1 + subst_len_to_index ls' v
401   end.
402
403 Definition canon_type_subst (tau : type) := subst_len_to_index (fv_type tau) >>>
404   ⇨ Atom.
405
406 Definition canon_type (tau: type) := tau.[canon_type_subst tau].
407
408 Example canon_type_ex : canon_type (? 8 → ? 8) = (? 0 → ? 0).
409 Proof.
410   reflexivity.
411 Qed.
412
413 Instance Ids_option {T} {ids : Ids T} : Ids (option T) := ids >>> Some.
414 Instance Rename_option {T} {rename : Rename T} : Rename (option T) := fun xi
415   ⇨ opterm ⇒
416
417   match opterm
418   ⇨ with
419   | None ⇒
420     ⇨ None
421   | Some term
422     ⇨ ⇒ Some
423     ⇨ (rename
424       ⇨ xi term)

```



```

418                                                                 end.
419
420 Fixpoint app_unify (Gamma : list type) (sigma : type) (tau : type) : option type
421   ↪ :=
422     Some tau.
423
424 Fixpoint infer_type (Gamma : repo) (depth: nat) (m : term) : option type :=
425   match m with
426   | !x ⇒ nth_error Gamma x
427   | λ s ⇒ let otau := infer_type ((? depth) :: Gamma) (depth + 1) s in
428     match otau with
429     | Some tau ⇒ Some (? depth ↪ tau)
430     | None ⇒ None
431   end
432   | p @ q ⇒ let osigma := infer_type Gamma depth q in
433     let otau_sigma := infer_type Gamma depth p in
434     match (osigma, otau_sigma) with
435     | (Some sigma, Some tau) ⇒ app_unify Gamma sigma tau
436     | _ ⇒ None
437   end
438   end.
439
440 Definition upd {A} {B} {eqdec: EqDec A _} (f : A → B) (upda: A) (updb: B) (a :
441   ↪ A) : B :=
442   if eqdec upda a then
443     updb
444   else
445     f a.
446
447 Fixpoint unify_ (types: nat) (rho1 : type) (rho2 : type) : option (var → type)
448   ↪ :=
449   match types with
450   | 0 ⇒ None
451   | S n ⇒
452     match (rho1, rho2) with
453     | (? a, _) ⇒ if subformula_dec (? a) rho2 then
454       if (? a) = rho2 then
455         Some ids
456       else
457         None
458     else
459       Some (single_subst a rho2)
460   | (_, ? a) ⇒ if subformula_dec (? a) rho1 then
461     if (? a) = rho1 then
462       Some ids
463     else
464       None
465   else
466     Some (single_subst a rho1)
467   | (sigma1 ↪ sigma2, tau1 ↪ tau2) ⇒ let oSu := unify_ n sigma2 tau2 in
468     match oSu with
469     | None ⇒ None
470     | Some Su ⇒
471       unify_ n sigma1.[Su] tau1.[Su] >=>
472         fun Sbst ⇒ Some (Su >> Sbst)

```

```

470                                     end
471     end
472 end.
473
474 Fixpoint depth_ty rho := match rho with
475 | ? n  $\Rightarrow$  1
476 | sigma  $\leadsto$  tau  $\Rightarrow$  1 + max (depth_ty sigma) (depth_ty tau)
477 end.
478
479 Definition unify rho1 rho2 := unify_
480   ((length (fv_type (rho1  $\leadsto$  rho2))) * (depth_ty (rho1  $\leadsto$  rho2))) rho1 rho2.
481
482 Definition mgu rho1 rho2 := unify rho1 rho2  $\gg$ 
483   fun Su  $\Rightarrow$  Some rho1.[Su].
484
485 Lemma nat_refl: forall x, (PeanoNat.Nat.eq_dec x x = left eq_refl).
486 Proof.
487   intros.
488   induction x.
489   - reflexivity.
490   - simpl. rewrite IHx. reflexivity.
491 Defined.
492
493 Lemma term_refl: forall x, eq_dec_term x x = left eq_refl.
494 Proof.
495   induction x.
496   - simpl. rewrite nat_refl. reflexivity.
497   - simpl. rewrite IHx1. rewrite IHx2. reflexivity.
498   - simpl. rewrite IHx. reflexivity.
499 Defined.
500
501 Lemma type_refl: forall t, eq_dec_type t t = left eq_refl.
502 Proof.
503   induction t.
504   - simpl. rewrite nat_refl. reflexivity.
505   - simpl. rewrite IHT1. rewrite IHT2. reflexivity.
506 Defined.
507
508 Lemma notU : (if subformula_dec (? 0) (? 0  $\leadsto$  ? 0) then true else false) = true.
509 Proof.
510   reflexivity.
511 Qed.
512
513 Fixpoint count_app (m: term) : nat :=
514   match m with
515 | p @ q  $\Rightarrow$  1 + count_app p
516 | _  $\Rightarrow$  0
517 end.
518
519 Fixpoint first_term t :=
520   match t with
521 | p @ q  $\Rightarrow$  first_term p
522 | s  $\Rightarrow$  s
523 end.
524
525 Fixpoint uncurry (t : term) : term * list term :=

```

```

526   match t with
527   | p @ q ⇒ let (hd, tl) := uncurry p in
528             (hd, tl ++ [q])
529   | m ⇒ (m , [])
530   end.
531
532 Lemma uncurry_var_singl t x: (x, []) = uncurry t → t = x.
533 Proof.
534   revert t x.
535   induction t.
536   + ainv.
537   + intros. asimpl in H. destruct (uncurry t1). ainv. destruct l; ainv.
538   + ainv.
539 Qed.
540
541 Hint Immediate uncurry_var_singl.
542 Hint Unfold uncurry.
543
544 Fixpoint first_fresh_type (rho: type) : var :=
545   match rho with
546   | ? x ⇒ (S x)
547   | sigma ~> tau ⇒ S (Nat.max (first_fresh_type sigma) (first_fresh_type tau))
548   end.
549
550 Definition fresh_type (rho: type) : type := ? (first_fresh_type rho).
551

```