

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

```

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

```

```

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

```

```

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

```

```

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

```

```

270 Proof.
271   intros. reflexivity.
272 Qed.
273
274 Lemma repo_pump_subst : forall (Gamma : repo) Gamma0 A Su, Gamma = Gamma0..[Su]
275   ⇨ → (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
283   ⇨ = Some A)
284   → exists B, B.[Su] = A /\ nth_error Gamma x = Some B.
285 Proof.
286   intros. destruct (nth_error Gamma x) eqn:Ht.
287   + exists t. rewrite subst_repo in H. rewrite Ht in H. ainv. split;
288     ⇨ reflexivity.
289   + rewrite subst_repo in H. rewrite Ht in H. ainv.
290 Qed.
291
292 Lemma subst_arr_is_arr_or : forall x t Su t0, x.[Su] = t ⇨ t0
293   → (exists st st0,
294     x = st ⇨ st0 /\ st.[Su] = t /\ st0.[Su] = t0) \/
295     (exists a, x = ? a).
296 Proof.
297   intros. destruct x.
298   - right. exists x. auto.
299   - left. exists x1. exists x2.
300     split.
301     + reflexivity.
302     + split; ainv.
303 Qed.
304
305 Lemma subst_arr : forall x y Su, x.[Su] ⇨ y.[Su] = (x ⇨ y).[Su].
306 Proof.
307   reflexivity.
308 Qed.
309
310 Lemma add_arr_head : forall A B B0, B = B0 → A ⇨ B = A ⇨ B0.
311 Proof.
312   intros. subst. reflexivity.
313 Qed.
314
315 Lemma mkarrow_subst_exists : forall ts x Su a, x.[Su] = make_arrow_type ts (? a)
316   ⇨ →
317   exists ts0 a0, x = (make_arrow_type ts0 (? a0)).
318 Proof.
319   induction ts.
320   - intros. simpl in H. symmetry in H. apply subst_var_is_var in H. exists [].
321     ⇨ ainv.
322   - intros. rewrite make_arrow_type_head in H. apply subst_arr_is_arr_or in H as
323     ⇨ [[st [st0 [xst [xsu stmkarr]]]] | xvar].
324     + apply IHts in stmkarr. inv stmkarr. inv H. exists (st :: x0). exists x.
325     ⇨ rewrite make_arrow_type_head.

```

```

319     reflexivity.
320 + ainv. exists []. exists x0. reflexivity.
321 Qed.
322
323
324 Definition not_in_codom (ls : list (var * type)) (a : var) :=
325   Forall (not_subf a) (rel_codom ls).
326
327 Theorem not_in_codom_dec : forall ls x, {not_in_codom ls x} + {~not_in_codom ls
328   ↪ 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   ↪ :
338   not_in_codom (c :: ls) a → not_in_codom ls a.
339 Proof.
340   ainv.
341 Qed.
342
343 Definition domain_codomain_free ls :=
344   Forall (not_in_codom ls) (rel_dom ls).
345
346 Theorem domain_codomain_free_dec : forall ls, { domain_codomain_free ls } + { ~
347   ↪ domain_codomain_free ls }.
348 Proof.
349   intros ls.
350   unfold domain_codomain_free.
351   apply Forall_dec.
352   apply not_in_codom_dec.
353 Defined.
354
355 Definition unique_domain {A B} (ls : list (A * B)) :=
356   NoDup (rel_dom ls).
357
358 Theorem unique_domain_dec {A B: Type} {eq_dec: EqDec A eq}: forall (ls : list (A
359   ↪ * B)), {unique_domain ls} + {~unique_domain ls}.
360 Proof.
361   intros ls.
362   unfold unique_domain.
363   induction ls.
364   - left. constructor.
365   - destruct IHls.
366     + destruct (in_dec eq_dec (fst a) (rel_dom ls)).
367       { right. isfalse. }
368       { left. unfold rel_dom. rewrite map_cons. constructor; assumption. }
369     + right. isfalse.
370 Defined.
371
372 Definition correct_context ls :=
373   unique_domain ls /\
374   domain_codomain_free ls.

```

```

371
372 Theorem correct_context_dec : forall ls, {correct_context ls} +
    ↪ {~correct_context ls}.
373 Proof.
374   intros.
375   unfold correct_context.
376   destruct (unique_domain_dec ls).
377   - destruct (domain_codomain_free_dec ls).
378     + left. split; assumption.
379     + right. isfalse.
380   - right. isfalse.
381 Defined.
382
383 (* Was hat das hier zu suchen?
384 Fixpoint wrap_lam (n : nat) (m : term) : term :=
385   match n with
386   | 0 ⇒ m
387   | S n ⇒ \_ (wrap_lam n (rename (+1) m) @ !0)
388   end. *)
389
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) >>>
    ↪ Atom.
404
405 Definition canon_type (tau: type) := tau.[canon_type_subst tau].
406
407 Example canon_type_ex : canon_type (? 8 ↪ ? 8) = (? 0 ↪ ? 0).
408 Proof.
409   reflexivity.
410 Qed.
411
412 Instance Ids_option {T} {ids : Ids T} : Ids (option T) := ids >>> Some.
413 Instance Rename_option {T} {rename : Rename T} : Rename (option T) := fun xi
    ↪ opterm ⇒
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```



```

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

```

```

471     end
472 end.
473
474 Fixpoint depth_ty rho := match rho with
475 | ? n ⇒ 1
476 | sigma ↪ tau ⇒ 1 + max (depth_ty sigma) (depth_ty tau)
477 end.
478
479 Definition unify rho1 rho2 := unify_
480   ((length (fv_type (rho1 ↪ rho2))) * (depth_ty (rho1 ↪ rho2))) rho1 rho2.
481
482 Definition mgu rho1 rho2 := unify rho1 rho2 >=
483   fun Su ⇒ 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 ↪ ? 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 ⇒ 1 + count_app p
516   | _ ⇒ 0
517   end.
518
519 Fixpoint first_term t :=
520   match t with
521   | p @ q ⇒ first_term p
522   | s ⇒ 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

```