Princ.v

```
Require Import Coq.Arith.PeanoNat.
Require Import Coq.Lists.List.
Require Import Coq.Bool.Sumbool.
Require Import Coq.Classes.EquivDec.
Require Import Autosubst.Autosubst.
Require Import Datatypes.

Require Import PrincInh.Types.
Require Import PrincInh.Utils.
Require Import PrincInh.NFTerms.
Require Import PrincInh.LongTyping.
Require Import PrincInh.SfC.
Require Import PrincInh.Paths.

Import ListNotations.
Import EqNotations.

Definition nfprinc (tau: type) (M: nfterm) : Type :=
    inhabited (nfty_long [] M tau) /\ forall sigma, nfty_long [] M sigma -> exists Su, subst Su tau = sigma.

(*
Definition princ_T (tau: type) (M: term) : Type :=
  prod (ty_T [] M tau) (forall sigma, ty_T [] M sigma -> {Su | tau.[Su] = sigma}).
*)
(* Lemma 14 in Paper *)
(*
Lemma subformula_princ : forall m tau (proof : nfty [] m tau) sigma' tau',
    subformula (sigma' ~> tau') tau -> TD_b proof tau' = false -> (princ_T tau m -> False).
Proof.
  intros.
  remember (first_fresh_type tau) as a.
  pose proof (zwölf (TD_b proof) proof TD_b_corr a).
  rewrite filt_mtTy in X0.
  remember (filtration (TD_b proof) a (sigma' ~> tau')) as t.
  assert (t = ? a).
  {
    subst.
    simpl.
    rewrite H0.
    rewrite Bool.andb_false_r.
    reflexivity.
  }
  unfold princ_T in X.
  destruct X.
  pose proof (s (filtration (TD_b proof) a tau) X0).
  destruct H2 as [Su H2].
  subst.
  apply (subst_subformula sigma' tau' tau (TD_b proof) (first_fresh_type tau) Su) in H.
  - rewrite H1 in H. symmetry in H. apply subst_var_is_var_T in H. ainv.
  - assumption.
```

```
Qed.
*)
Example id_princ : nflong_princ (? 0 ⇝ ? 0) (\__ ( !! 0 @@ [])).
Proof.
    unfold nflong_princ.
    split.
    - constructor. econstructor.
      + instantiate (1 := []). reflexivity.
      + intros. exfalso. inversion pms.
    - simpl. destruct rho'.
      + ainv.
      + intros. ainv.
        assert (ts = []).
        { clear X. asimpl in Lenproof. symmetry in Lenproof.
          apply length_zero_iff_nil in Lenproof. assumption. }
        subst. clear X. exists (? a .: ids). reflexivity. Unshelve. reflexivity.
Qed.

Definition norm_princ_inhab (M: nfterm) (tau: type) :=
    nflong_princ tau M.

Lemma subst_var_is_var_T : forall Su a tau, ? a = tau.[Su] → {b | tau = ? b}.
Proof.
  intros.
  simpl in H.
  destruct tau.
  - exists x. reflexivity.
  - simpl in H. exfalso. inv H.
Qed.

Definition star tau := forall pi, In pi (dom_P tau) → forall x, P tau (pi ++ [Tgt]) = Some (? x) →
                                                                R_tau_ts tau (pi ++ [Tgt]) (pi ++ [Tgt]).

(* General: Utils *)
Definition Req {T} (A B : (T → T → Type)) := (Rsub A B * Rsub B A)%type.

(* General: Utils *)
Lemma trans_hull_in_R {A} {eqdec : EqDec A eq} R (a b : A) : trans_hull R a b → {c & In (a, c) R} + {c & In (c, a) R}.
Proof.
  intros.
  induction X.
  - left. eexists. exact i.
  - destruct IHX1 as [[c' Hin]|[c' Hin]].
    + left. exists c'. assumption.
    + right. exists c'. assumption.
Qed.

Lemma ts_cl_in_R {A} {eqdec : EqDec A eq} R (a b : A) : ts_cl_list R a b → {c & In (a, c) R} + {c & In (c, a) R}.
Proof.
  intros. apply ts_cl_list_trans_sym in X. apply trans_hull_in_R in X. destruct X.
```

```
  - destruct s. unfold sym_hull_list in i. apply In_app_sumbool in i. destruct i.
    + left. eexists. apply i.
    + right. apply In_flipped in i. rewrite flipped_invol in i. eexists. apply i.
  - destruct s. unfold sym_hull_list in i. apply In_app_sumbool in i. destruct i.
    + right. eexists. apply i.
    + left. apply In_flipped in i. rewrite flipped_invol in i. eexists. apply i.
Qed.

(* General: Paths *)
Lemma R_tau_ts_in_dom_P : forall pi pi' tau, R_tau_ts tau pi pi' -> prod (In pi (dom_P tau))
                                                                        {a & P tau pi = Some (? a)}.
Proof.
  intros.
  apply ts_cl_in_R in X.
  destruct X as [[x Hin] | [x Hin]].
  - unfold R_tau_list in Hin. apply filter_In in Hin. destruct Hin. apply in_prod_iff in H.
    unfold R_tau_cond in H0. simpl in H0. apply andb_prop in H0. destruct H0.
    destruct (P tau pi); try discriminate H1. destruct t; try discriminate H1. split. ainv. exists x0. reflexivity.
  - unfold R_tau_list in Hin. apply filter_In in Hin. destruct Hin. apply in_prod_iff in H.
    unfold R_tau_cond in H0. simpl in H0. apply andb_prop in H0. destruct H0.
    destruct (P tau x); try discriminate H1. destruct t; try discriminate H1.
    destruct (P tau pi); try discriminate H1. rewrite equivb_prop in H1. subst.
    split. ainv. exists x0. reflexivity.
Qed.

Lemma almost_refl_l {A} R : forall (pi pi' :A), ts_cl_list R pi pi' -> ts_cl_list R pi pi.
Proof.
  intros.
  econstructor 3. apply X. constructor 2. assumption.
Qed.

Lemma almost_refl_r {A} R : forall (pi pi' :A), ts_cl_list R pi pi' -> ts_cl_list R pi' pi'.
Proof.
  intros.
  econstructor 3. constructor 2. exact X. assumption.
Qed.

Lemma replace_at_paths_split : forall pi tau1 tau2, {m1 & { m2 & replace_at_path (tau1 ~> tau2) (fresh_type (tau1 ~> tau2)) pi = m1 ~> m2}} + {pi = []} + {In pi (dom_P (tau1~>tau2))}.
Proof.
  intros.
Admitted.

(* Irgendwas mit 26
```

```
Lemma replace_all_paths_split : forall pi s tau1 tau2, {m1 & { m2 & replace_all_paths (fresh_type (tau1 ⇝ tau2)) (replaceable_paths (tau1 ⇝ tau2) (\__ s) pi) = m1 ⇝ m2}} + {pi = []}.
Proof.
  intros.
  destruct (pi == []).
  - right. assumption.
  - left. eexists. eexists.
    unfold replace_all_paths.
Admitted. *)

Lemma siebenundzwanzig {m tau} : nfty_long [] m tau → nflong_princ tau m → Req (R_m_ts m) (R_tau_ts tau).
Proof.
  intros. pose proof (Long_closed _ _ X). pose proof X as nfty_l.
  apply long_to_sfc_tau in X. apply sfc_tau_to_Rsub_m_tau in X.
  split.
  - assumption.
  - unfold Rsub. assert (forall pi pi', (R_tau_ts tau) pi pi' → ((R_m_ts m) pi pi' → False) → False).
    { intros.
      remember (fresh_type tau).
      pose proof R_tau_ts_in_dom_P _ _ _ X1 as [Hin [a Heq]].
      apply P_P_ok_set in Heq as [Hpr HPok].
      pose proof sechsundzwanzig m tau pi Hpr a nfty_l HPok as Hszw.
      ainv.
      pose proof X3 _ Hszw as [Su Heqtau].
      clear X1 H1 Hpr a nfty_l H0 X.
      admit.
    }
    (*intros.
    pose proof H1 pi pi' _ X0.
    destruct (R_m_ts_dec m pi pi').
    + ainv.
    + exfalso. eapply H1. apply f.*)
Admitted.

Lemma einunddreissig : forall tau, star tau → forall m, nfty_long [] m tau →
  R_m_ts m = R_tau_ts tau → nflong_princ tau m.
Proof.
Admitted.
```