LongTyping.v

```
Require Import Coq.Arith.PeanoNat.
Require Import Coq.Lists.List.
Require Import Autosubst.Autosubst.

Require Import PrincInh.Terms.
Require Import PrincInh.Types.
Require Import PrincInh.Typing.
Require Import PrincInh.NFTerms.
Require Import PrincInh.Utils.

Import ListNotations.
Import EqNotations.

(* Long typings for terms and nfterms *)
Inductive long_ty_T (Gamma : repo) : term -> type -> Type :=
| Long_I_T s A B : long_ty_T (A :: Gamma) s B ->
        long_ty_T Gamma (Lam s) (Arr A B)
| Long_E_T x ms ts a : nth_error Gamma x = Some (make_arrow_type ts (? a))
        -> Forall2_T (long_ty_T Gamma) ms ts ->
        long_ty_T Gamma (curry (! x) ms) (? a).

Inductive nfty_long (Gamma : repo) : nfterm -> type -> Type :=
| NFTy_lam_long s sigma tau : nfty_long (sigma :: Gamma) s tau -> nfty_long Gamma (\__ s) (sigma ~> tau)
| NFTy_var_long : forall x a ts ms (Gammaok : nth_error Gamma x = Some (make_arrow_type ts (? a)))
                        (Lenproof : length ms = length ts),
    (forall n (pms : n < length ms),
        nfty_long Gamma (nth_ok ms n pms) (nth_ok ts n (rew Lenproof in pms))) ->
    nfty_long Gamma ( !! x @@ ms) (? a).

Inductive nfty_long_subj : forall Gamma Gamma' m m' rho rho', nfty_long Gamma m rho -> nfty_long Gamma' m' rho' -> Type :=
| nfty_long_refl : forall Gamma m rho (proof: nfty_long Gamma m rho), nfty_long_subj _ _ _ _ _ _ proof proof
| nfty_long_trans : forall Gamma Gamma' Gamma'' m m' m'' rho rho' rho''
                      (proof1 : nfty_long Gamma m rho)
                      (proof2 : nfty_long Gamma' m' rho')
                      (proof3 : nfty_long Gamma'' m'' rho''),
    nfty_long_subj _ _ _ _ _ _ proof1 proof2 ->
    nfty_long_subj _ _ _ _ _ _ proof2 proof3 ->
    nfty_long_subj _ _ _ _ _ _ proof1 proof3
| nfty_long_subj_I : forall Gamma sigma tau s (proof : nfty_long (sigma :: Gamma) s tau),
    nfty_long_subj _ _ _ _ _ _ proof (NFTy_lam_long _ _ _ _ proof)
| nfty_long_subj_E : forall Gamma x ts ms a
                       (Gammaok : nth_error Gamma x = Some (make_arrow_type ts (? a)))
                       (Lenproof : length ms = length ts)
                       (proofs : (forall n (pms : n < length ms),
                                    nfty_long Gamma (nth_ok ms n pms) (nth_ok ts n (rew Lenproof in pms))))
                       n (len: n < length ms),
```

```
    nfty_long_subj _ _ _ _ _ _ (proofs n len) (NFTy_var_long _ _ _ _ _ Gammaok Lenproof proofs).

Definition nflong_princ (rho: type) (M: nfterm) : Type :=
  nfty_long [] M rho * forall rho', nfty_long [] M rho' -> { Su & rho.[Su] = rho' }.

Lemma nfty_long_subterm : forall n m, subterm_nf n m -> forall tau Gamma, nfty_long Gamma m tau -> {Gamma' & {tau' & nfty_long Gamma' n tau'}}.
Proof.
  induction 1; intros.
  - exists Gamma. exists tau. assumption.
  - inv X0. eapply IHX. exact X1.
  - inv X0. apply In_nth_error_set in i. destruct i as [n H].
    apply nth_error_nth_ok in H. destruct H as [lp H]. pose proof (X1 n lp). eapply IHX. rewrite H in X0. exact X0.
Qed.



Lemma Long_E_aux_T : forall Gamma x ms ts a curr v,
nth_error Gamma x = Some (make_arrow_type ts (? a))
        -> Forall2_T (long_ty_T Gamma) ms ts ->
        curr = curry (! x) ms -> v = (? a) ->
        long_ty_T Gamma curr v.
Proof.
  intros. subst. econstructor; try assumption.
  - apply H.
  - assumption.
Qed.

Definition long_ty_T_ind' :
      forall P : repo -> term -> type -> Type,
       (forall (Gamma : repo) (s : term) (A B : type),
        long_ty_T (A :: Gamma) s B ->
        P (A :: Gamma) s B -> P Gamma (\_ s) (A ~> B)) ->
       (forall (Gamma : repo) (x : var)
          (ms : list term) (ts : list type) (a : var),
        nth_error Gamma x = Some (make_arrow_type ts (? a)) ->
        Forall2_T (long_ty_T Gamma) ms ts ->
        Forall2_T (P Gamma) ms ts ->
        P Gamma (curry (! x) ms) (? a)) ->
       forall (Gamma : repo) (t : term) (t0 : type),
       long_ty_T Gamma t t0 -> P Gamma t t0 :=
       fun P icase ecase =>
       fix long_ty_ind'_rec (Gamma : repo) (t : term) (t0 : type)
        (proof : long_ty_T Gamma t t0) {struct proof} : P Gamma t t0 :=
            match proof with
            | Long_I_T _ s A B proof' => icase Gamma s A B proof'
                   (long_ty_ind'_rec (A :: Gamma) s B proof')
            | Long_E_T _ x ms ts a eqproof forallproof =>
              ecase Gamma x ms ts a eqproof forallproof
                   ((fix forall_rec (ms : list term) (ts : list type)
                         (proof : Forall2_T (long_ty_T Gamma) ms ts) {struct proof}
```

```
                        : Forall2_T (P Gamma) ms ts :=
                          match proof with
                          | Forall2_T_nil _ => Forall2_T_nil _
                          | Forall2_T_cons _ m t ms ts headproof tailproof =>
                            Forall2_T_cons _ m t ms ts
                                           (long_ty_ind'_rec Gamma m t headproof)
                                           (forall_rec _ _ tailproof)
                          end) ms ts forallproof
                      )
          end.
(*
Lemma Forall2_if_long_rel_T : forall Gamma ms ts, long_rel_T Gamma ms ts -> Forall2_T (long_ty_T Gamma) ms ts.
Proof.
  intros Gamma ms ts.
  induction 1; constructor; try constructor; assumption.
Qed.

Lemma long_rel_if_Forall2_T : forall Gamma ms ts,  Forall2_T (long_ty_T Gamma) ms ts -> long_rel_T Gamma ms ts.
Proof.
  intros Gamma ms ts.
  induction 1; constructor; try constructor; assumption.
Qed.*)

Lemma Forall2_inh {B C}: forall (A : B -> C -> Type) ms ts, Forall2 (fun a b => inhabited (A a b)) ms ts -> inhabited (Forall2_T (fun a b => A a b) ms ts).
  Proof.
    induction 1.
    - constructor. constructor.
    - ainv. constructor. constructor.
      + assumption.
      + assumption.
  Qed.

Lemma mkArrow_curry_ty_T : forall Gamma ms ts a ,
    Forall2_T (fun m t => ty_T Gamma m t) ms ts
    -> forall x, ty_T Gamma x (make_arrow_type ts a)
    -> ty_T Gamma (curry x ms) a.
Proof.
    induction 1.
    - intros. simpl in *. assumption.
    - intros. simpl in *. apply IHX. econstructor.
      + apply X0.
      + assumption.
Qed.

Lemma long_impl_ty_T : forall Gamma m t, long_ty_T Gamma m t -> ty_T Gamma m t.
Proof.
    intros. induction X using long_ty_T_ind'.
    - constructor. assumption.
    - eapply mkArrow_curry_ty_T.
      + apply X0.
      + constructor. assumption.
Qed.
```

```
Definition is_long_ty (t: term) (ty: type) := long_ty_T [] t ty.
Definition is_ty (t: term) (typ : type) := ty_T [] t typ.

Lemma long_ty_var_T : forall Gamma x t, nth_error Gamma x = Some (? t) →
 ↪ long_ty_T Gamma (! x) (? t).
Proof.
  intros. assert (! x = curry (! x) []). { reflexivity. } rewrite H0.
   ↪ econstructor.
  - instantiate (1:=[]). auto.
  - constructor.
Qed.
(*
Lemma long_rel_rev_T : forall ms ts Gamma, long_rel_T Gamma ms ts → long_rel_T
 ↪ Gamma (rev ms) (rev ts).
Proof.
  intros. apply long_rel_if_Forall2_T. apply Forall2_T_is_rev. repeat rewrite
 ↪ rev_involutive.
  apply Forall2_if_long_rel_T. assumption.
Qed.

Lemma rev_long_rel_T : forall ms ts Gamma, long_rel_T Gamma (rev ms) (rev ts) →
 ↪ long_rel_T Gamma ms ts.
  intros. apply long_rel_if_Forall2_T. apply Forall2_T_is_rev_r. apply
 ↪ Forall2_if_long_rel_T. assumption.
Qed.*)

Lemma long_ty_app_T : forall Gamma n m ms t ts a x,
  n = curry (! x) (ms) →
  long_ty_T Gamma m t →
  Forall2_T (long_ty_T Gamma) ms ts →
  nth_error Gamma x = Some (make_arrow_type ts (t ⇝ ? a))
  → long_ty_T Gamma (n @ m) (? a).
Proof.
  intros.
  subst. rewrite ← curry_tail.  econstructor.
  - instantiate (1:=(ts ++ [t])).
    rewrite make_arrow_type_last. assumption.
  - apply Forall2_T_is_rev_r. repeat rewrite rev_unit.
    constructor.
    + assumption.
    + apply Forall2_T_is_rev in X0. assumption.
Qed.


Lemma long_ty_lam_aux_T : forall m Gamma, { s & { t & long_ty_T (s :: Gamma) m t
 ↪ } } →
  { t0 & long_ty_T Gamma (\_ m) t0}.
Proof.
  intros.
  ainv. exists (x ⇝ x0). constructor. assumption.
Qed.

Lemma long_general_T : forall m Su tau Gamma,
  ty_T Gamma m tau → long_ty_T Gamma..[Su] m tau.[Su] → long_ty_T Gamma m tau.
Proof.
```

```
intros m.
remember (term_length m) as lengthm.
assert (term_length m ⩽ lengthm). { firstorder. }
clear Heqlengthm.
revert m H.
induction (lengthm).
- intros. exfalso. ainv.
- intros. destruct m.
  + ainv. symmetry in H4. apply curry_if_nil in H4. ainv.
  apply subst_var_is_var_T in H1. ainv. apply Long_E_T with [].
    { simpl. inv H1. reflexivity. }
    { constructor. }
  + inversion X0. apply subst_var_is_var_T in H1 as [b H1]. rewrite ← H0 in X.
    apply mp_gen_T in X as [sigmas [HForall HGamma]].
    rewrite H1 in ⋆. apply Long_E_T with sigmas.
    { assumption. }
    { assert (Forall2_T (fun t sigma ⇒ t = sigma.[Su]) ts sigmas).
      { rewrite subst_repo in H2. rewrite HGamma in H2. revert HForall H2 X1.
        clear ...
        revert ts sigmas.
        induction ms.
        - intros. inv HForall. inv X1. constructor.
        - intros. inversion HForall. inversion X1. constructor.
          { ainv. }
          { ainv. apply IHms; try assumption.
            - simpl. apply f_equal. assumption. } }
      rewrite ← H0 in H.
      generalize (curry_le (! x) ms _ H).
      clear HGamma H2.
      revert X1 HForall H3 IHn.
      clear ...
      revert sigmas ts.
      induction ms.
      - ainv. constructor.
      - intros. ainv. constructor.
        + apply IHn with Su.
          { ainv. firstorder. }
          { assumption. }
          { assumption. }
        + eapply IHms.
          { eassumption. }
          { assumption. }
          { assumption. }
          { assumption. }
          { ainv. } }
  + inversion X0. symmetry in H2. apply subst_arr_is_arr_or_T in H2 as [Harr | Hvar].
    { destruct Harr as [st [st0 [Htau [HstSu Hst0su]]]].
      rewrite Htau. constructor. apply IHn with Su.
      - simpl in H. firstorder.
      - inversion X. rewrite Htau in H0. ainv.
      - rewrite ← HstSu in X1. rewrite subst_repo_cons in X1. rewrite Hst0su.
        assumption.
    }
    { ainv. }
```

```
        { ainv. }
Qed.
```