Terms2.v

```
Require Import List.

Require Import Autosubst.Autosubst.

Require Import PrincInh.Terms.

Import ListNotations.

Inductive term2 :=
| App_var (ms : list term2) (x : var)
| App_lam (ms : list term2) (s : {bind term2})
.

Definition term2_ind' : forall P : term2 → Prop,
       (forall (ms : list term2) (x : var),
       (Forall P ms) →
       P (App_var (ms) x)) →
       (forall (ms : list term2) (s : {bind term2}),
       Forall P ms →
       P s → P (App_lam ms s)) →
       forall t : term2, P t :=
fun P app_var_case app_lam_case ⇒
  fix term2_ind'_rec (t : term2) :=
    match t with
    | App_var ms x ⇒ app_var_case ms x ((fix term2_ind'_var_rec (ms : list
     ↪ term2) : Forall P ms :=
           match ms with
           | [] ⇒ Forall_nil _
           | t::ts ⇒ @Forall_cons _ _ t ts (term2_ind'_rec t)
            ↪ (term2_ind'_var_rec ts)
           end) ms)
    | App_lam ms s ⇒ app_lam_case ms s ((fix term2_ind'_lam_rec (ms : list
     ↪ term2) : Forall P ms :=
           match ms with
           | [] ⇒ Forall_nil _
           | t::ts ⇒ @Forall_cons _ _ t ts (term2_ind'_rec t)
            ↪ (term2_ind'_lam_rec ts)
           end) ms) (term2_ind'_rec s)
    end.

Notation "'!!!' x" := (App_var [] x) (at level 15).
Notation "'!!' p '@@' q" := (App_var q p) (at level 31, left associativity).
Notation "'\___' p" := (App_lam [] p) (at level 35, right associativity).
Notation "'\__' p '@@' q" := (App_lam q p) (at level 35, right associativity).


Fixpoint term2_term (m : term2) : term :=
  match m with
  | !! x @@ q ⇒ curry (! x) (map term2_term q)
  | \__ s @@ q ⇒ curry (\_ (term2_term s)) (map term2_term q)
  end.

Fixpoint term_term2 (m : term) : term2 :=
  match m with
```

```
  | !x ⇒ !!!x
  | \_ s ⇒ \___ (term_term2 s)
  | p @ q ⇒ let p' := term_term2 p in
             match p' with
             | !! x @@ q' ⇒ !! x @@ (q' ++ [term_term2 q])
             | \__ s @@ q' ⇒ \__ s @@ (q' ++ [term_term2 q])
             end
  end.

Lemma term2_term_id : forall m, (term2_term >>> term_term2) m = m.
Proof.
  induction m using term2_ind'.
  - unfold "_ >>> _" in *.
    simpl in *. rewrite (Forall_forall _ ms)in H.
    induction ms using rev_ind.
    + reflexivity.
    + rewrite map_app. simpl. rewrite curry_tail. simpl.
      rewrite IHms.
      { rewrite (H x0).
        - reflexivity.
        - apply (in_or_app). right. constructor. reflexivity. }
      { intros.
        apply H. apply (in_or_app). left. assumption.
      }
 - unfold "_ >>> _" in *.
   simpl in *. rewrite (Forall_forall _ ms) in H.
   induction ms using rev_ind.
   + simpl. rewrite IHm. reflexivity.
   + rewrite map_app. simpl. rewrite curry_tail. simpl.
     rewrite IHms.
     { rewrite (H x).
       - reflexivity.
       - apply (in_or_app). right. constructor. reflexivity. }
     { intros.
       apply H. apply (in_or_app). left. assumption. }
Qed.

Lemma term_term2_id : forall m, (term_term2 >>> term2_term) m = m.
Proof.
  unfold "_ >>> _".
  induction m.
  - reflexivity.
  - simpl. destruct (term_term2 m1) eqn:Httm1.
    + simpl. rewrite map_app. simpl. rewrite curry_tail.
      rewrite IHm2. ainv.
    + simpl. rewrite map_app. simpl. rewrite curry_tail.
      rewrite IHm2. ainv.
  - simpl. rewrite IHm. reflexivity.
Qed.

Lemma term2_term_if : forall m n, term2_term m = term2_term n → m = n.
Proof.
  intros.
  rewrite ← (term2_term_id m).
  rewrite ← (term2_term_id n).
```

```
  unfold "_ >>> _".
  rewrite H.
  reflexivity.
Qed.

Lemma term_term2_if : forall m n, term_term2 m = term_term2 n → m = n.
Proof.
  intros.
  rewrite ← (term_term2_id m).
  rewrite ← (term_term2_id n).
  unfold "_ >>> _".
  rewrite H.
  reflexivity.
Qed.

Instance Ids_term2 : Ids term2.  unfold Ids. apply (App_var []).  Defined.
Instance Rename_term2 : Rename term2 (*:= fun xi ⇒ (term2_term >>> Rename_term
 ↪  xi >>> term_term2). *)
:= fix ren_term2 (xi: var → var) (s : term2) {struct s} : term2
              := match s as t return (annot term2 t) with
                | !! x @@ ms ⇒ !! (xi x) @@ (map (ren_term2 xi) ms)
                | \__ s @@ ms ⇒ \__ (ren_term2 (upren xi) s) @@ (map (ren_term2
                 ↪  xi) ms)
                end.
Instance Subst_term2 : Subst term2 (*:= fun sigma ⇒ (term2_term >>> subst
 ↪  (sigma >>> term2_term) >>> term_term2). *)

:=
fix dummy (sigma : var → term2) (s : term2) {struct s} : term2 :=
  match s as t return (annot term2 t) with
  | !! x @@ ms ⇒ match sigma x with
                  | !! y @@ ns ⇒ !! y @@ (ns ++ map (dummy sigma) ms)
                  | \__ s @@ ns ⇒ \__ s @@ (ns ++ map (dummy sigma) ms)
                  end
  | \__ s0 @@ ms ⇒ \__ dummy (up sigma) s0 @@ map (dummy sigma) ms
  end.

Lemma rename_subst_term2 (xi : var → var) (s : term2) :
    rename xi s = s.[ren xi].
Proof.
  intros.
  revert xi.
  induction s using term2_ind'.
  - intros. simpl. rewrite Forall_forall in H. erewrite map_ext_in.
    + reflexivity.
    + intros. apply H. assumption.
  - intros. simpl. rewrite Forall_forall in H.
    rewrite IHs. rewrite up_upren_internal.
    + erewrite map_ext_in.
      { reflexivity. }
      { intros. apply H. assumption. }
    + auto.
Defined.

Lemma subst_up_ids : up ids = ids.
```

```
Proof.
  unfold up.
  unfold ids.
  unfold "_ >>> _".
  f_ext.
  induction x.
  - reflexivity.
  - simpl. simpl in IHx. auto.
Qed.

Lemma subst_id_term2 (s : term2) :
    s.[ids] = s.
Proof.
  induction s using term2_ind'.
  - simpl. rewrite Forall_forall in H. erewrite map_ext_in.
    + instantiate (1:=id). rewrite map_id. reflexivity.
    + auto.
  - simpl. rewrite subst_up_ids. unfold subst in IHs. rewrite IHs.
    rewrite Forall_forall in H. erewrite map_ext_in.
    + instantiate (1:=id). rewrite map_id. reflexivity.
    + auto.
Qed.

Lemma id_subst_term2 (sigma : var → term2) (x : var) :
    (ids x).[sigma] = sigma x.
Proof.
  simpl.
  destruct (sigma x);
  rewrite app_nil_r;
  reflexivity.
Qed.

Lemma term2_var_app_split_eq : forall x y ms ns, x = y →  ms = ns → !! x @@ ms
 ↪  = !! y @@ ns.
Proof.
  intros. subst. reflexivity.
Qed.

Lemma term2_var_app_apps_eq : forall x ms ns, ms = ns → !! x @@ ms = !! x @@
 ↪  ns.
Proof.
  intros. subst. reflexivity.
Qed.

Lemma term2_lam_app_split_eq : forall x y ms ns, x = y → ms = ns → \__ x @@ ms
 ↪  = \__ y @@ ns.
Proof.
  intros. subst. reflexivity.
Qed.

Lemma term2_lam_app_apps_eq : forall x ms ns, ms = ns → \__ x @@ ms = \__ x @@
 ↪  ns.
Proof.
  intros. subst. reflexivity.
Qed.
```

```
Lemma list_split_eq_l {A}: forall (l : list A) l1 l2, l1 = l2 → l ++ l1 = l ++ l2.
Proof.
  intros. subst. reflexivity.
Qed.

Lemma ren_subst_comp_term2 : forall xi sigma (s : term2), (rename xi s).[sigma] = s.[xi >>> sigma].
Proof.
fix ih 3. intros xi sigma s. destruct s.
  - simpl. destruct (sigma (xi x)) eqn:H.
    + apply term2_var_app_apps_eq. apply list_split_eq_l. unfold subst in ih.
      rewrite map_map. induction ms.
      { reflexivity. }
      { simpl. rewrite ih. rewrite IHms. reflexivity. }
    + apply term2_lam_app_apps_eq. apply list_split_eq_l. unfold subst in ih.
      rewrite map_map. induction ms.
      { reflexivity. }
      { simpl. rewrite ih. rewrite IHms. reflexivity. }
  - simpl. apply term2_lam_app_split_eq.
    + rewrite up_comp_ren_subst. unfold subst in ih. apply ih.
    + rewrite map_map. induction ms.
      { reflexivity. }
      { simpl. rewrite ih. rewrite IHms. reflexivity. }
Qed.

Lemma up_comp_subst_ren_term2 :
      forall sigma xi, up (sigma >>> rename xi) = up sigma >>>rename (upren xi).
Proof.
  apply up_comp_subst_ren_internal.
    - reflexivity.
    - apply rename_subst_term2.
    - apply ren_subst_comp_term2.
Qed.

Lemma up_comp_subst_ren_n_term2 :
 forall sigma xi n, upn n (sigma >>> rename xi) = upn n sigma >>> rename (iterate upren n xi).
Proof.
   apply up_comp_subst_ren_n_internal. apply up_comp_subst_ren_term2.
Qed.

Lemma subst_ren_comp_term2 : forall sigma xi (s : term2),
      rename xi s.[sigma] = s.[sigma >>> rename xi].
Proof.
  fix ih 3. intros. destruct s.
  - simpl. destruct (sigma x).
    + simpl. apply term2_var_app_apps_eq. rewrite map_app. apply list_split_eq_l.
      unfold subst in ih. rewrite map_map. induction ms.
      { reflexivity. }
      { simpl. rewrite ih. rewrite IHms. reflexivity. }
    + simpl. apply term2_lam_app_apps_eq. rewrite map_app. apply list_split_eq_l.
      unfold subst in ih. rewrite map_map. induction ms.
```

```
      { reflexivity. }
      { simpl. rewrite ih. rewrite IHms. reflexivity. }
  - simpl. apply term2_lam_app_split_eq.
    + unfold rename in ih. unfold rename. unfold subst in ih. rewrite
      ↪ up_comp_subst_ren_term2.
      apply ih.
    + rewrite map_map. induction ms.
      { reflexivity. }
      { simpl. rewrite ih. rewrite IHms. reflexivity. }
Qed.

Lemma subst_comp_term2 : forall (sigma tau : var → term2) (s : term2),
    s.[sigma].[tau] = s.[sigma >> tau].
Proof.
intros.
  revert sigma tau.
  induction s using term2_ind'.
  - intros. simpl. destruct (sigma x); simpl.
     + destruct (tau x0);
        try apply term2_var_app_apps_eq;
        try apply term2_lam_app_apps_eq;
        try rewrite ← app_assoc;
        apply list_split_eq_l;
        rewrite map_app;
        apply list_split_eq_l;
        rewrite Forall_forall in H;
        rewrite map_map;
        apply map_ext_in;
        intros; apply H; assumption.
    + apply term2_lam_app_apps_eq.
      rewrite map_app.
      apply list_split_eq_l.
        rewrite Forall_forall in H.
        rewrite map_map.
        apply map_ext_in.
        intros. apply H. assumption.
  - intros. simpl. unfold subst in IHs.
    apply term2_lam_app_split_eq.
    + rewrite IHs. rewrite up_comp_internal.
      { reflexivity. }
      { apply id_subst_term2. }
      { apply ren_subst_comp_term2. }
      { apply subst_ren_comp_term2. }
    + rewrite map_map. rewrite Forall_forall in H. erewrite map_ext_in.
      { reflexivity. }
      { intros. apply H. assumption. }
Qed.

Instance SubstLemmas_term2 : SubstLemmas term2.
Proof.
  split.
  { apply rename_subst_term2. }
  { apply subst_id_term2. }
  { apply id_subst_term2. }
  { apply subst_comp_term2. }
Qed.
```

```
315
316   (*
317 Coercion term_term2 : term ⟶ term2.
318 Coercion term2_term : term2 ⟶ term.
319 *)
```