NFTerms.v

```
Require Import Coq.Lists.List.
Require Import Coq.Classes.EquivDec.
Require Import Autosubst.Autosubst.

Require Import PrincInh.Utils.
Require Import PrincInh.Terms.

Import ListNotations.
Import EqNotations.

Inductive nfterm :=
| NFcurr (ms: list nfterm) (x : var)
| NFLam (s: {bind nfterm})
.

Instance Ids_term : Ids nfterm := fun var => NFcurr [] var.
Instance Rename_term : Rename nfterm :=
  fun ren =>
    fix dummy m := match m as n return (annot nfterm n) with
                   | NFcurr ms x => NFcurr (mmap dummy ms) (ren x)
                   | NFLam s => NFLam (dummy s)
                   end.



Definition nfterm_ind' : forall P : nfterm -> Prop,
       (forall (ms : list nfterm) (x : var), (Forall P ms) -> P (NFcurr ms x)) ->
       (forall s : {bind nfterm}, P s -> P (NFLam s)) -> forall n : nfterm, P n :=
fun (P : nfterm -> Prop) f
  (f0 : forall s : {bind nfterm}, P s -> P (NFLam s)) =>
fix F (n : nfterm) : P n :=
  match n as n0 return (P n0) with
  | NFcurr ms x => f ms x ((fix ms_rec ms : Forall P ms
                           := match ms with
                              | [] => Forall_nil _
                              | x :: xs => @Forall_cons _ _ x xs (F x) (ms_rec xs)
                              end
                          ) ms)
  | NFLam s => f0 s (F s)
  end.

Definition nfterm_rect'
     : forall P : nfterm -> Type,
       (forall (ms : list nfterm) (x : var) (p: forall n (lp : n < length ms), P (nth_ok ms n lp)), P (NFcurr ms x)) ->
       (forall s : {bind nfterm}, P s -> P (NFLam s)) -> forall n : nfterm, P n.
Proof.
  intros.
  revert n.
  fix F 1.
  destruct n.
```

```
  - apply X. revert ms. fix ih 1. destruct ms.
    + intros. exfalso. inversion lp.
    + destruct n0.
      * intros. simpl. apply F.
      * simpl. intros. apply ih.
  - apply X0. apply F.
Qed.

Notation "'!!' x '@@' ms" := (NFcurr ms x) (at level 31, left associativity).
Notation "'\__' s" := (NFLam s) (at level 35, right associativity).

Inductive subterm_nf : nfterm → nfterm → Type :=
| sub_ref : forall m, subterm_nf m m
| sub_lam m1 m2 : subterm_nf m1 m2 → subterm_nf m1 (\__ m2)
| sub_app m1 m2 ms x : subterm_nf m1 m2 → In m2 ms → subterm_nf m1 ( !! x @@ ms).


Definition eqdec_nfterm_fix (m1 m2 : nfterm) : {m1 = m2} + {m1 <> m2}.
  revert m1 m2. fix ih 1. intros. destruct m1; destruct m2.
  - destruct (x == x0).
    + destruct (@list_eqdec _ _ ih ms ms0).
      * left. ainv.
      * right. intros F. ainv. apply c. reflexivity.
    + right. intros F. ainv. apply c. reflexivity.
  - right. intros F. discriminate F.
  - right. intros F. discriminate F.
  - destruct (ih s s0).
    + left. ainv.
    + right. intros F. ainv. apply n. reflexivity.
Defined.

Instance eqdec_nfterm : EqDec nfterm eq. unfold EqDec. apply eqdec_nfterm_fix. Defined.




Fixpoint NFterm_term nft : term :=
  match nft with
  | !! x @@ ms ⇒ curry (! x) (map NFterm_term ms)
  | \__ s ⇒ \_ NFterm_term s
  end.

Fixpoint term_NFterm t : option nfterm :=
  match t with
  | ! x ⇒ Some ( !! x @@ [])
  | \_ s ⇒ match term_NFterm s with
             | None ⇒ None
             | Some s' ⇒ Some (\__ s')
             end
  | p @ q ⇒ match term_NFterm p with
              | None ⇒ None
              | Some ( !! x @@ ms) ⇒ match term_NFterm q with
                                       | None ⇒ None
```

```
                                          | Some q' => Some (!! x @@ (ms ++ [q']))
                                          end
                  | Some (\__ s) => None
                  end
  end.

Lemma NFterm_term_inv1 : forall t, term_NFterm (NFterm_term t) = Some t.
Proof.
  intros.
  induction t using nfterm_ind'.
  - induction ms using rev_ind.
    + reflexivity.
    + rewrite Forall_forall in IHms.
      rewrite Forall_forall in H.
      simpl. rewrite map_app. simpl. rewrite curry_tail.
      simpl. simpl in IHms. rewrite IHms.
      * erewrite H.
        ** reflexivity.
        ** apply in_or_app. right. constructor. reflexivity.
      * intros. apply H. apply in_or_app. left. assumption.
  - simpl. rewrite IHt. reflexivity.
Qed.

Lemma NF_if_no_redex_all : forall t, (forall m n, ~subterm ((\_m) @ n) t) -> NF t.
Proof.
  induction t; intros.
  - unfold NF. intros. isfalse.
  - unfold NF. intros t F. inversion F.
    + eapply H. instantiate (1:=t2). instantiate (1:=s1). subst. constructor.
    + subst. eapply IHt1.
      * intros. intros Fsub. eapply H. constructor. exact Fsub. Unshelve. apply s2.
      * assumption.
    + subst. eapply IHt2.
      * intros. intros Fsub. eapply H. constructor 3. exact Fsub. Unshelve. apply t3.
      * assumption.
  - unfold NF. intros t F. inv F. eapply IHt.
    + intros m n Fsub. eapply H. constructor. exact Fsub.
    + exact H1.
Qed.

Lemma no_redex_if_NF_all : forall t, NF t -> forall m n, ~subterm ((\_m) @ n) t.
Proof.
  intros.
  - induction t.
    + intros. intros F. inversion F.
    + intros. intros F. inv F.
      * unfold NF in H. pose proof (H m.[t2/]).
        apply H0. constructor. reflexivity.
      * revert H2. apply IHt1. unfold NF.
        intros. unfold NF in H. pose proof (H (t' @ t2)). intros Fstep.
        apply H0. constructor. assumption.
      * revert H2. apply IHt2. unfold NF.
```

```
        intros. unfold NF in H. pose proof (H (t1 @ t')). intros Fstep.
        apply H0. constructor. assumption.
    + intros. unfold NF in H. intros F. eapply IHt.
      * unfold NF. intros. pose proof (H (\_ t')). intros Fstep. apply H0. constructor. assumption.
      * inversion F. assumption.
Qed.

Lemma NF_iff_no_redex: forall t, NF t <-> forall m n, ~subterm ((\_m) @ n) t.
Proof.
  split.
  - apply no_redex_if_NF_all.
  - apply NF_if_no_redex_all.
Qed.

Lemma NFterm t : NF t -> { t' & term_NFterm t = Some t'}.
Proof.
  intros.
  rewrite NF_iff_no_redex in H.
  induction t.
  - simpl. intros. eexists. reflexivity.
  - simpl. assert (forall (m : {bind term}) (n : term), ~ subterm ((\_ m) @ n) t1).
    {
      intros m n F. eapply H. constructor. exact F.
    }
    apply IHt1 in H0. destruct H0. rewrite e. destruct x.
    + assert (forall (m : {bind term}) (n : term), ~ subterm ((\_ m) @ n) t2).
      {
        intros m n F. eapply H. constructor 3. exact F.
      }
      apply IHt2 in H0 as [t H1]. rewrite H1. exists (!! x @@ (ms ++ [t])).
      reflexivity.
    + exfalso. assert (exists s, t1 = \_ s).
      {
        destruct t1.
        + ainv.
        + ainv. destruct (term_NFterm t1_1); try discriminate H2.
                destruct n; try discriminate H2.
                destruct (term_NFterm t1_2); try discriminate H2.
        + exists s0. reflexivity.
      }
      destruct H0.
      eapply H. instantiate (1 :=t2). instantiate (1 := x). subst. constructor.
  - simpl. assert (forall (m : {bind term}) (n : term), ~ subterm ((\_ m) @ n) (s)).
    {
      intros m n F. eapply H. constructor. exact F.
    }
    apply IHt in H0. destruct H0. rewrite e. exists (\__ x). reflexivity.
Defined.

Lemma NF_lam : forall s, NF (\_ s) -> NF s.
Proof.
  intros.
```

```
  unfold NF in *.
  intros t' F.
  eapply H.
  constructor.
  exact F.
Qed.

Lemma NF_is_no_redex : forall s q, ~(NF ((\_s) @ q)).
Proof.
  intros.
  rewrite NF_iff_no_redex. simpl.
  intros F.
  apply (F s q).
  constructor.
Qed.

Fixpoint max_fvar (m: nfterm) : var :=
  match m with
  | !! x @@ ms ⇒ fold_left Nat.max (map max_fvar ms) (S x)
  | \__ s ⇒ pred (max_fvar s)
  end.

Definition all_var_in_repo {A} m (Delta : list A) := max_fvar m < S (length
 ↪  Delta).



Fixpoint term_NFterm_proof (t: term) : NF t → nfterm.
Proof.
  intros proof.
  apply NFterm in proof.
  destruct proof.
  apply x.
Defined.
```