

NFTerms.v

```

1 Require Import Coq.Lists.List.
2 Require Import Coq.Classes.EquivDec.
3 Require Import Autosubst.Autosubst.
4
5 Require Import PrincInh.Utills.
6 Require Import PrincInh.Terms.
7
8 Import ListNotations.
9 Import EqNotations.
10
11 Inductive nfterm :=
12 | NFcurr (ms: list nfterm) (x : var)
13 | NFLam (s: {bind nfterm})
14 .
15
16 Instance Ids_term : Ids nfterm := fun var ⇒ NFcurr [] var.
17 Instance Rename_term : Rename nfterm :=
18   fun ren ⇒
19     fix dummy m := match m as n return (annot nfterm n) with
20       | NFcurr ms x ⇒ NFcurr (mmap dummy ms) (ren x)
21       | NFLam s ⇒ NFLam (dummy s)
22     end.
23
24
25
26 Definition nfterm_ind' : forall P : nfterm → Prop,
27   (forall (ms : list nfterm) (x : var), (Forall P ms) → P (NFcurr ms x))
28   ⇨ →
29   (forall s : {bind nfterm}, P s → P (NFLam s)) → forall n : nfterm, P n
30   ⇨ :=
31 fun (P : nfterm → Prop) f
32   (f0 : forall s : {bind nfterm}, P s → P (NFLam s)) ⇒
33 fix F (n : nfterm) : P n :=
34   match n as n0 return (P n0) with
35   | NFcurr ms x ⇒ f ms x ((fix ms_rec ms : Forall P ms
36     := match ms with
37       | [] ⇒ Forall_nil _
38       | x :: xs ⇒ @Forall_cons _ _ x xs (F x) (ms_rec
39         ⇨ xs)
40     end
41     ) ms)
42   | NFLam s ⇒ f0 s (F s)
43 end.
44
45 Definition nfterm_rect'
46   : forall P : nfterm → Type,
47   (forall (ms : list nfterm) (x : var) (p: forall n (lp : n < length ms), P
48     ⇨ (nth_ok ms n lp)), P (NFcurr ms x)) →
49   (forall s : {bind nfterm}, P s → P (NFLam s)) → forall n : nfterm, P n.
50
51 Proof.
52   intros.
53   revert n.
54   fix F 1.
55   destruct n.

```

```

51 - apply X. revert ms. fix ih 1. destruct ms.
52 + intros. exfalse. inversion lp.
53 + destruct n0.
54   * intros. simpl. apply F.
55   * simpl. intros. apply ih.
56 - apply X0. apply F.
57 Qed.
58
59 Notation "'!!' x '@@' ms" := (NFcurr ms x) (at level 31, left associativity).
60 Notation "'\__' s" := (NFLam s) (at level 35, right associativity).
61
62 Inductive subterm_nf : nfterm → nfterm → Type :=
63 | sub_ref : forall m, subterm_nf m m
64 | sub_lam m1 m2 : subterm_nf m1 m2 → subterm_nf m1 (λ__ m2)
65 | sub_app m1 m2 ms x : subterm_nf m1 m2 → In m2 ms → subterm_nf m1 (!! x @@
  ↪ ms).
66
67
68 Definition eqdec_nfterm_fix (m1 m2 : nfterm) : {m1 = m2} + {m1 < m2}.
69   revert m1 m2. fix ih 1. intros. destruct m1; destruct m2.
70   - destruct (x = x0).
71     + destruct (@list_eqdec _ _ ih ms ms0).
72       * left. ainv.
73       * right. intros F. ainv. apply c. reflexivity.
74     + right. intros F. ainv. apply c. reflexivity.
75   - right. intros F. discriminate F.
76   - right. intros F. discriminate F.
77   - destruct (ih s s0).
78     + left. ainv.
79     + right. intros F. ainv. apply n. reflexivity.
80 Defined.
81
82 Instance eqdec_nfterm : EqDec nfterm eq. unfold EqDec. apply eqdec_nfterm_fix.
  ↪ Defined.
83
84
85
86
87 Fixpoint NFterm_term nft : term :=
88   match nft with
89   | !! x @@ ms ⇒ curry (! x) (map NFterm_term ms)
90   | λ__ s ⇒ λ__ NFterm_term s
91   end.
92
93 Fixpoint term_NFterm t : option nfterm :=
94   match t with
95   | ! x ⇒ Some (!! x @@ [])
96   | λ__ s ⇒ match term_NFterm s with
97     | None ⇒ None
98     | Some s' ⇒ Some (λ__ s')
99   end
100 | p @ q ⇒ match term_NFterm p with
101   | None ⇒ None
102   | Some (!! x @@ ms) ⇒ match term_NFterm q with
103     | None ⇒ None

```

```

104         | Some q' ⇒ Some (!! x @ (ms ++ [q']))
105     end
106     | Some (⊔ s) ⇒ None
107 end
108 end.
109
110 Lemma NFterm_term_inv1 : forall t, term_NFterm (NFterm_term t) = Some t.
111 Proof.
112   intros.
113   induction t using nfterm_ind'.
114   - induction ms using rev_ind.
115     + reflexivity.
116     + rewrite Forall_forall in IHms.
117       rewrite Forall_forall in H.
118       simpl. rewrite map_app. simpl. rewrite curry_tail.
119       simpl. simpl in IHms. rewrite IHms.
120       * erewrite H.
121         ** reflexivity.
122         ** apply in_or_app. right. constructor. reflexivity.
123         * intros. apply H. apply in_or_app. left. assumption.
124   - simpl. rewrite IHt. reflexivity.
125 Qed.
126
127 Lemma NF_if_no_redex_all : forall t, (forall m n, ~subterm ((⊔_m) @ n) t) → NF
128   ↪ t.
129 Proof.
130   induction t; intros.
131   - unfold NF. intros. isfalse.
132   - unfold NF. intros t F. inversion F.
133     + eapply H. instantiate (1:=t2). instantiate (1:=s1). subst. constructor.
134     + subst. eapply IHt1.
135       * intros. intros Fsub. eapply H. constructor. exact Fsub. Unshelve. apply
136         ↪ s2.
137       * assumption.
138     + subst. eapply IHt2.
139       * intros. intros Fsub. eapply H. constructor 3. exact Fsub. Unshelve.
140       ↪ apply t3.
141       * assumption.
142   - unfold NF. intros t F. inv F. eapply IHt.
143     + intros m n Fsub. eapply H. constructor. exact Fsub.
144     + exact H1.
145 Qed.
146
147 Lemma no_redex_if_NF_all : forall t, NF t → forall m n, ~subterm ((⊔_m) @ n) t.
148 Proof.
149   intros.
150   - induction t.
151     + intros. intros F. inversion F.
152     + intros. intros F. inv F.
153       * unfold NF in H. pose proof (H m.[t2/]).
154         apply H0. constructor. reflexivity.
155       * revert H2. apply IHt1. unfold NF.
156         intros. unfold NF in H. pose proof (H (t' @ t2)). intros Fstep.
157         apply H0. constructor. assumption.
158       * revert H2. apply IHt2. unfold NF.

```

```

156      intros. unfold NF in H. pose proof (H (t1 @ t')). intros Fstep.
157      apply H0. constructor. assumption.
158 + intros. unfold NF in H. intros F. eapply IHt.
159   * unfold NF. intros. pose proof (H (λ_ t')). intros Fstep. apply H0.
    ↪ constructor. assumption.
160   * inversion F. assumption.
161 Qed.
162
163 Lemma NF_iff_no_redex: forall t, NF t ⟶ forall m n, ~subterm ((λ_ m) @ n) t.
164 Proof.
165   split.
166   - apply no_redex_if_NF_all.
167   - apply NF_if_no_redex_all.
168 Qed.
169
170 Lemma NFterm t : NF t → { t' & term_NFterm t = Some t' }.
171 Proof.
172   intros.
173   rewrite NF_iff_no_redex in H.
174   induction t.
175   - simpl. intros. eexists. reflexivity.
176   - simpl. assert (forall (m : {bind term}) (n : term), ~ subterm ((λ_ m) @ n)
    ↪ t1).
    {
177     intros m n F. eapply H. constructor. exact F.
178   }
179   apply IHt1 in H0. destruct H0. rewrite e. destruct x.
180 + assert (forall (m : {bind term}) (n : term), ~ subterm ((λ_ m) @ n) t2).
    {
181     intros m n F. eapply H. constructor 3. exact F.
182   }
183   apply IHt2 in H0 as [t H1]. rewrite H1. exists (!! x @ (ms ++ [t])).
184   reflexivity.
185 + exfalso. assert (exists s, t1 = λ_ s).
    {
186     destruct t1.
187     + ainv.
188     + ainv. destruct (term_NFterm t1_1); try discriminate H2.
189       destruct n; try discriminate H2.
190       destruct (term_NFterm t1_2); try discriminate H2.
191     + exists s0. reflexivity.
192   }
193   destruct H0.
194   eapply H. instantiate (1 := t2). instantiate (1 := x). subst. constructor.
195 - simpl. assert (forall (m : {bind term}) (n : term), ~ subterm ((λ_ m) @ n)
    ↪ (s)).
    {
196     intros m n F. eapply H. constructor. exact F.
197   }
198   apply IHt in H0. destruct H0. rewrite e. exists (λ_ x). reflexivity.
199 Defined.
200
201 Lemma NF_lam : forall s, NF (λ_ s) → NF s.
202 Proof.
203   intros.

```

```

208   unfold NF in *.
209   intros t' F.
210   eapply H.
211   constructor.
212   exact F.
213 Qed.
214
215 Lemma NF_is_no_redex : forall s q, ~(NF ((λ _ s) @ q)).
216 Proof.
217   intros.
218   rewrite NF_iff_no_redex. simpl.
219   intros F.
220   apply (F s q).
221   constructor.
222 Qed.
223
224 Fixpoint max_fvar (m: nfterm) : var :=
225   match m with
226   | !! x @@ ms ⇒ fold_left Nat.max (map max_fvar ms) (S x)
227   | λ _ s ⇒ pred (max_fvar s)
228   end.
229
230 Definition all_var_in_repo {A} m (Delta : list A) := max_fvar m < S (length
  ↪ Delta).
231
232
233
234 Fixpoint term_NFterm_proof (t: term) : NF t → nfterm.
235 Proof.
236   intros proof.
237   apply NFterm in proof.
238   destruct proof.
239   apply x.
240 Defined.

```