







Terms.v

```

1  Require Import Autosubst.Autosubst.
2  Require Import Nat PeanoNat.
3  Require Import Coq.Arith.EqNat.
4  Require Import Coq.Logic.FunctionalExtensionality.
5  Require Import Coq.Logic.Classical_Pred_Type.
6  Require Import Coq.Lists.List.
7  Require Import Coq.Lists.ListSet.
8  Require Import Coq.Classes.EquivDec.
9  Require Import Coq.Bool.Sumbool.
10 Require Import Coq.Classes.DecidableClass.
11
12 Require Import PrincInh.Utills.
13
14 Import ListNotations.
15
16 Inductive term :=
17   | Var (x : var)
18   | App (p q : term)
19   | Lam (s : {bind term}).
20
21
22 Notation "'!' x" := (Var x) (at level 15).
23 Notation "p '@' q" := (App p q) (at level 31, left associativity).
24 Notation "'\_ ' p" := (Lam p) (at level 35, right associativity).
25
26 Instance Ids_term : Ids term. derive. Defined.
27 Instance Rename_term : Rename term. derive. Defined.
28 Instance Subst_term : Subst term. derive. Defined.
29 Instance SubstLemmas_term : SubstLemmas term. derive. Qed.
30
31 Definition tI := _ !0.
32 Definition tK := _ _ !0.
33 Definition tS := _ _ _ ((!2@!0)@( !1@!0)).
34
35 Fixpoint term_length (m: term) : nat :=
36   match m with
37   | Var _ => 1
38   | App p q => 1 + (term_length p) + (term_length q)
39   | Lam s => 1 + (term_length s)
40   end.
41
42 Instance eq_dec_term : EqDec term eq.
43 Proof.
44   unfold EqDec.
45   unfold equiv.
46   induction x.
47   - destruct y.
48     + destruct (x = x0).
49       { left. ainv. }
50       { right. unfold complement. intros F. inversion F. contradiction. }
51     + right. intros F. inversion F.
52     + right. intros F. inversion F.
53   - destruct y.
54     + right. intros F. inversion F.

```

```

55 + destruct (IHx1 y1).
56   { destruct (IHx2 y2).
57     - left. subst. reflexivity.
58     - right. intros F. inversion F. contradiction. }
59   { right. intros F. inversion F. contradiction. }
60 + right. intros F. ainv.
61 - destruct y.
62 + right. intros F. ainv.
63 + right. intros F. ainv.
64 + destruct (IHx s0).
65   { left. subst. reflexivity. }
66   { right. intros F. inversion F. contradiction. }
67 Defined.
68
69 Goal forall sigma,
70   (Lam (App (Var 0) (Var 3))).[sigma] = Lam (App (Var 0) (sigma
71     ↪ 2)).[ren(+1)]).
72 intros. asimpl. reflexivity. Qed.
73
74 Inductive step : term → term → Prop :=
75 | Step_beta (s1 s2 t : term) :
76   s1.[t/] = s2 → step (App (Lam s1) t) s2
77 | Step_appL (s1 s2 t : term) :
78   step s1 s2 → step (App s1 t) (App s2 t)
79 | Step_appR (s t1 t2 : term) :
80   step t1 t2 → step (App s t1) (App s t2)
81 | Step_lam (s1 s2 : term) :
82   step s1 s2 → step (Lam s1) (Lam s2).
83
84 Lemma substitutivity s1 s2 :
85   step s1 s2 → forall sigma, step s1.[sigma] s2.[sigma].
86 Proof.
87   induction 1; constructor; subst; try autosubst.
88 Qed.
89
90 Lemma term_not_rec_appL : forall s t, s <math>\Diamond</math> s @ t.
91 Proof.
92   intros s t F.
93   induction s.
94   - inversion F.
95   - inversion F. subst. contradiction.
96   - inversion F.
97 Qed.
98
99 Lemma term_not_rec_appR : forall s t, s <math>\Diamond</math> t @ s.
100 Proof.
101   intros s t F.
102   induction s.
103   - inversion F.
104   - inversion F. subst. contradiction.
105   - inversion F.
106 Qed.
107
108 Definition omega_term :=  $\Box$  !0 @ !0.

```

```

109 Definition Omega_term := omega_term@omega_term.
110
111 Example omega_step : step Omega_term Omega_term.
112 Proof.
113   constructor. reflexivity.
114 Qed.
115
116 Inductive subterm : term → term → Prop :=
117 | subterm_refl : forall t, subterm t t
118 | subterm_appL : forall s s' t, subterm s s' → subterm s (s' @ t)
119 | subterm_appR : forall s t t', subterm t t' → subterm t (s @ t')
120 | subterm_lam : forall t t', subterm t t' → subterm t (λ t').
121
122 Theorem subterm_dec : forall t t', (subterm t t') + {~subterm t t'}.
123 Proof.
124   intros.
125   induction t'.
126     + destruct (t = (!x)).
127       { left. ainv. constructor. }
128       { right. intros F. inversion F. subst. apply c. reflexivity. }
129     + destruct IHt'1.
130       { left. apply subterm_appL. apply s. }
131       { destruct IHt'2.
132         - left. apply subterm_appR. apply s.
133         - destruct (t = (t'1 @ t'2)).
134           + ainv. left. constructor.
135           + right. intros F. ainv. apply c. reflexivity. }
136     + destruct IHt'.
137       { left. constructor. assumption. }
138       { destruct (t = (λ s)); dec_eq.
139         - left. constructor.
140         - right. intros F. ainv. dec_eq. }
141 Defined.
142
143 Definition NF (t : term) := forall t', ~step t t'.
144
145 Theorem redex_no_NF : forall t, (exists m n, subterm ((λ m) @ n) t) → ~NF t.
146 Proof.
147   induction t.
148   - ainv.
149   - intros. unfold NF. intros F. ainv. inversion H.
150     + subst. pose proof (F x.[t2/]). apply H0. constructor. reflexivity.
151     + subst. apply IHt1.
152       { exists x. exists x0. assumption. }
153       { unfold NF. intros. intros Fstep. pose proof (F (t' @ t2)). apply H0.
154         constructor. assumption. }
155     + subst. apply IHt2.
156       { exists x. exists x0. assumption. }
157       { unfold NF. intros. intros Fstep. pose proof (F (t1 @ t')). apply H0.
158         constructor. assumption. }
159   - ainv. intros F. unfold NF in F. eapply IHt.
160     + exists x. exists x0. assumption.
161     + unfold NF. intros. intros Fstep. eapply F. constructor. apply Fstep.
162 Qed.
163

```

```

164 Theorem NF_no_redex : forall t, NF t → ~(exists m n, subterm ((λ _ m) @ n) t).
165 Proof.
166   intros. intros F. apply redex_no_NF in F. contradiction.
167 Qed.
168
169 Theorem no_redex_NF : forall t, ~(exists m n, subterm ((λ _ m) @ n) t) → NF t.
170 Proof.
171   intros.
172   induction t.
173   - unfold NF. intros. intros F. ainv.
174   - unfold NF. intros. intros F. inversion F.
175     + subst. apply H. exists s1. exists t2. constructor.
176     + subst. apply IHt1 with s2.
177       { intros Fex. ainv. apply H. exists x. exists x0. constructor.
178         → assumption. }
179       { assumption. }
180     + subst. apply IHt2 with t3.
181       { intros Fex. ainv. apply H. exists x. exists x0. constructor 3.
182         → assumption. }
183       { assumption. }
184   - unfold NF. intros. intros F. ainv. apply IHt with s2.
185     + intros Fex. ainv. apply H. exists x. exists x0. constructor. assumption.
186     + assumption.
187 Qed.
188
189 Theorem NF_iff_no_redex : forall t, NF t ↔ ~(exists m n, subterm ((λ _ m) @ n)
190   → t).
191 Proof.
192   intros t. split.
193   - apply NF_no_redex.
194   - apply no_redex_NF.
195 Qed.
196
197 Theorem exists_redex_dec : forall t ,
198   {(exists m n, subterm ((λ _ m) @ n) t)} + {~(exists m n, subterm ((λ _ m) @ n)
199   → t)}.
200 Proof.
201   intros t.
202   simpl.
203   induction t.
204   - right. intros F. ainv.
205   - destruct IHt1.
206     + left. ainv. exists x. exists x0. constructor. apply H0.
207     + destruct IHt2.
208       { left. ainv. exists x. exists x0. constructor 3. assumption. }
209       { destruct t1.
210         - right. intros F. ainv. inversion H0.
211           + subst. ainv.
212           + subst. apply n0. exists x0. exists x1. assumption.
213         - right. intros F. ainv. inversion H0.
214           + subst. apply n. exists x. exists x0. assumption.
215           + subst. apply n0. exists x. exists x0. assumption.
216         - left. exists s. exists t2. constructor. }
217   - destruct IHt.
218     + left. ainv. exists x. exists x0. constructor. assumption.

```

```

215       + right. intros F. apply n. ainv. exists x. exists x0. assumption.
216 Defined.
217
218 Theorem is_NF_dec : forall t, {NF t}+{~(NF t)}.
219 Proof.
220   intros.
221   destruct (exists_redex_dec t).
222   - right. intros F. apply NF_iff_no_redex in F. contradiction.
223   - left. apply NF_iff_no_redex. assumption.
224 Defined.
225
226 Definition curry (x:term) (terms: list term) : term :=
227   fold_left App terms x.
228
229 Fixpoint uncurry (m : term) : term * (list term) :=
230   match m with
231   | p @ q ⇒ let (h,t) := uncurry p in
232     (h, t ++ [q])
233   | s ⇒ (s, [])
234   end.
235
236 Lemma curry_tail : forall ms x a, curry x (ms ++ [a]) = curry x ms @ a.
237 Proof.
238   induction ms.
239   - reflexivity.
240   - simpl. intros. rewrite (IHms (x@a) a0). reflexivity.
241 Qed.
242
243 Example curry_ex : curry tS [tI ; tS ; tK ] = (tS@tI)@tS@tK.
244 Proof.
245   reflexivity.
246 Qed.
247
248 Lemma curry_if_nil : forall ms a x,
249   ! x = curry a ms →
250   a = (!x) /\ ms = [].
251 Proof.
252   induction ms.
253   - simpl in *. ainv. auto.
254   - intros. apply IHms in H. ainv.
255 Qed.
256
257 Lemma curry_split : forall x l a s t, curry (! x) (l ++ [a]) = s @ t →
258   s = curry (! x) l /\ t = a.
259 Proof.
260   intros.
261   rewrite curry_tail in H. ainv. split; reflexivity.
262 Qed.
263
264 Lemma term_app_split : forall m n, term_length (m@n) = 1 + term_length m +
265   ↪ term_length n.
266 Proof.
267   intros.
268   constructor.
269 Qed.

```

```

270 Lemma curry_le_cons : forall ms x a, term_length (curry x ms) ≤ term_length
    ↪ (curry x (a :: ms)).
271 Proof.
272   intros.
273   revert x.
274   induction ms using rev_ind.
275   - simpl. firstorder.
276   - intros. rewrite app_comm_cons.
277     repeat rewrite curry_tail.
278     repeat rewrite term_app_split.
279     firstorder.
280 Qed.
281
282 Lemma curry_le_last : forall ms x a, term_length (curry x ms) ≤ term_length
    ↪ (curry x (ms ++ [a])).
283 Proof.
284   intros.
285   revert x.
286   induction ms.
287   - simpl. firstorder.
288   - intros. simpl. apply IHms.
289 Qed.
290
291 Lemma curry_le : forall x ms n, term_length (curry x ms) ≤ n →
    Forall (fun m ⇒ term_length m < n) ms.
292
293 Proof.
294   intros x ms.
295   induction ms using rev_ind.
296   - intros; constructor.
297   - intros.
298     apply Forall_forall. intros.
299     eapply (Nat.lt_le_trans); [ | exact H ].
300   apply in_app_or in H0 as [H1 | H2].
301     + simpl. eapply (Nat.lt_le_trans); [ | apply curry_le_last ].
302     rewrite ← (curry_le_last ms x x0) in H.
303       generalize (proj1 (Forall_forall _ _) (IHms (term_length (curry x ms))
    ↪ (Nat.le_refl _))).
304       intros.
305       eapply H0. assumption.
306     + inversion H2.
307       { subst. rewrite curry_tail. simpl. firstorder. }
308       { ainv. }
309 Qed.
310
311 (* TODO Nicht mehr genutzt *)
312 Lemma curry_subst : forall ts t f, (curry t ts).[f] = curry (t.[f]) (map (subst
    ↪ f) ts).
313
314 Proof.
315   induction ts using rev_ind.
316   - reflexivity.
317   - intros.
318     rewrite map_app.
319     simpl.
320     repeat rewrite curry_tail.
321     simpl.

```

```
322      rewrite IHts.  
323      reflexivity.  
324 Qed.  
325  
326 Lemma curry_var : forall x, ! x = curry (! x) [].  
327 Proof.  
328   auto.  
329 Qed.
```