

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
24   ↪ associativity).
25 Notation "'\_' p" := (Lam p) (at level 35, right
26   ↪ associativity).
27
28 Instance Ids_term : Ids term. derive. Defined.
29 Instance Rename_term : Rename term. derive. Defined.
30 Instance Subst_term : Subst term. derive. Defined.
31 Instance SubstLemmas_term : SubstLemmas term. derive. Qed.
32
33 Definition tI :=  $\lambda \_ . !0$ .
34 Definition tK :=  $\lambda \_ . \lambda \_ . !0$ .
35 Definition tS :=  $\lambda \_ . \lambda \_ . \lambda \_ . ((!2 @ !0) @ (!1 @ !0))$ .
36
37 Fixpoint term_length (m: term) : nat :=
38   match m with
39   | Var _ ⇒ 1
40   | App p q ⇒ 1 + (term_length p) + (term_length q)
41   | Lam s ⇒ 1 + (term_length s)
42   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.
51         ↪ contradiction. }
52     + right. intros F. inversion F.
53     + right. intros F. inversion F.
54   - destruct y.
55     + right. intros F. inversion F.
56     + destruct (IHx1 y1).
57       { destruct (IHx2 y2).
58         - left. subst. reflexivity.
59         - right. intros F. inversion F. contradiction. }
60       { right. intros F. inversion F. contradiction. }
61     + right. intros F. ainv.
62   - destruct y.
63     + right. intros F. ainv.
64     + right. intros F. ainv.
65     + destruct (IHx s0).
66       { left. subst. reflexivity. }
67       { right. intros F. inversion F. contradiction. }
68
69 Defined.
70
71 Goal forall sigma,
72   (Lam (App (Var 0) (Var 3))).[sigma] = Lam (App (Var 0)
73     ↪ (sigma 2).[ren(+1)]).
74 intros. asimpl. reflexivity. Qed.
75
76 Inductive step : term → term → Prop :=
77 | Step_beta (s1 s2 t : term) :
78   s1.[t/] = s2 → step (App (Lam s1) t) s2
79 | Step_appL (s1 s2 t : term) :
80   step s1 s2 → step (App s1 t) (App s2 t)
81 | Step_appR (s t1 t2 : term) :
82   step t1 t2 → step (App s t1) (App s t2)
83 | Step_lam (s1 s2 : term) :
84   step s1 s2 → step (Lam s1) (Lam s2).

```

```

83 Lemma substitutivity s1 s2 :
84     step s1 s2 → forall sigma, step s1.[sigma] s2.[sigma].
85 Proof.
86     induction 1; constructor; subst; try autosubst.
87 Qed.
88
89 Lemma term_not_rec_appL : forall s t, s <math>\Diamond</math> s @ t.
90 Proof.
91     intros s t F.
92     induction s.
93     - inversion F.
94     - inversion F. subst. contradiction.
95     - inversion F.
96 Qed.
97
98 Lemma term_not_rec_appR : forall s t, s <math>\Diamond</math> t @ s.
99 Proof.
100     intros s t F.
101     induction s.
102     - inversion F.
103     - inversion F. subst. contradiction.
104     - inversion F.
105 Qed.
106
107 Definition omega_term :=  $\Box$  !0 @ !0.
108
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' @
119   ↪ t)
120 | subterm_appR : forall s t t', subterm t t' → subterm t (s @
121   ↪ t')
122 | subterm_lam : forall t t', subterm t t' → subterm t ( $\Box$  t').
123
124 Theorem subterm_dec : forall t t', (subterm t t') + {~subterm t
125   ↪ 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.
129          $\hookrightarrow$  reflexivity. }
130     + destruct IHt'1.
131       { left. apply subterm_appL. apply s. }
132       { destruct IHt'2.
133         - left. apply subterm_appR. apply s.
134         - destruct (t = (t'1 @ t'2)).
135           + ainv. left. constructor.
136           + right. intros F. ainv. apply c. reflexivity. }
137     + destruct IHt'.
138       { left. constructor. assumption. }
139       { destruct (t = ( $\lambda$ _s)); dec_eq.
140         - left. constructor.
141         - right. intros F. ainv. dec_eq. }
142
143 Defined.
144
145 Definition NF (t : term) := forall t', ~step t t'.
146
147 Theorem redex_no_NF : forall t, (exists m n, subterm (( $\lambda$ _ m) @
148    $\hookrightarrow$  n) t)  $\rightarrow$  ~NF t.
149
150 Proof.
151   induction t.
152   - ainv.
153   - intros. unfold NF. intros F. ainv. inversion H.
154     + subst. pose proof (F x.[t2/]). apply H0. constructor.
155        $\hookrightarrow$  reflexivity.
156     + subst. apply IHt1.
157       { exists x. exists x0. assumption. }
158       { unfold NF. intros. intros Fstep. pose proof (F (t' @
159          $\hookrightarrow$  t2)). apply H0.
160       constructor. assumption. }
161     + subst. apply IHt2.
162       { exists x. exists x0. assumption. }
163       { unfold NF. intros. intros Fstep. pose proof (F (t1 @
164          $\hookrightarrow$  t')). apply H0.
165       constructor. assumption. }
166   - ainv. intros F. unfold NF in F. eapply IHt.
167   + 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
      ↪ (( $\lambda$  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 (( $\lambda$  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. assumption. }
178       { assumption. }
179     + subst. apply IHt2 with t3.
180       { intros Fex. ainv. apply H. exists x. exists x0.
          ↪ constructor 3. assumption. }
181       { assumption. }
182   - unfold NF. intros. intros F. ainv. apply IHt with s2.
183     + intros Fex. ainv. apply H. exists x. exists x0.
      ↪ constructor. assumption.
184     + assumption.
185 Qed.
186
187 Theorem NF_iff_no_redex : forall t, NF t  $\rightarrow$  ~(exists m n,
      ↪ subterm (( $\lambda$  m) @ n) t).
188 Proof.
189   intros t. split.
190   - apply NF_no_redex.
191   - apply no_redex_NF.
192 Qed.
193
194 Theorem exists_redex_dec : forall t ,
195   {(exists m n, subterm (( $\lambda$  m) @ n) t)} + {~(exists m n,
      ↪ subterm (( $\lambda$  m) @ n) t)}.

```

```

196 Proof.
197   intros t.
198   simpl.
199   induction t.
200   - right. intros F. ainv.
201   - destruct IHt1.
202     + left. ainv. exists x. exists x0. constructor. apply
203        $\hookrightarrow$  H0.
204     + destruct IHt2.
205       { left. ainv. exists x. exists x0. constructor 3.
206          $\hookrightarrow$  assumption. }
207       { destruct t1.
208         - right. intros F. ainv. inversion H0.
209         + subst. ainv.
210         + subst. apply n0. exists x0. exists x1.
211            $\hookrightarrow$  assumption.
212         - right. intros F. ainv. inversion H0.
213         + subst. apply n. exists x. exists x0. assumption.
214         + subst. apply n0. exists x. exists x0.
215            $\hookrightarrow$  assumption.
216         - left. exists s. exists t2. constructor. }
217   - destruct IHt.
218   + left. ainv. exists x. exists x0. constructor.
219      $\hookrightarrow$  assumption.
220   + right. intros F. apply n. ainv. exists x. exists x0.
221      $\hookrightarrow$  assumption.
222 Defined.
223
224 Theorem is_NF_dec : forall t, {NF t}+{~(NF t)}.
225 Proof.
226   intros.
227   destruct (exists_redex_dec t).
228   - right. intros F. apply NF_iff_no_redex in F.
229      $\hookrightarrow$  contradiction.
230   - left. apply NF_iff_no_redex. assumption.
231 Defined.
232
233 Definition curry (x:term) (terms: list term) : term :=
234   fold_left App terms x.
235
236 Fixpoint uncurry (m : term) : term * (list term) :=
237   match m with
238   | p @ q  $\Rightarrow$  let (h,t) := uncurry p in
239     (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 + term_length n.
265 Proof.
266   intros.
267   constructor.
268 Qed.
269
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
  ⇨ →
292   Forall (fun m ⇒ term_length m < n) ms.
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
312 (* TODO Nicht mehr genutzt *)

```



```

313 Lemma curry_subst : forall ts t f, (curry t ts).[f] = curry
    ↪ (t.[f]) (map (subst f) ts).
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.

```