

## LongTyping.v

```
1 Require Import Coq.Arith.PeanoNat.
2 Require Import Coq.Lists.List.
3 Require Import Autosubst.Autosubst.
4
5 Require Import PrincInh.Terms.
6 Require Import PrincInh.Types.
7 Require Import PrincInh.Typing.
8 Require Import PrincInh.NFTerms.
9 Require Import PrincInh.Utills.
10
11 Import ListNotations.
12 Import EqNotations.
13
14 (* Long typings for terms and nfterms *)
15 Inductive long_ty_T (Gamma : repo) : term → type → Type :=
16 | Long_I_T s A B : long_ty_T (A :: Gamma) s B →
17   long_ty_T Gamma (Lam s) (Arr A B)
18 | Long_E_T x ms ts a : nth_error Gamma x = Some (make_arrow_type ts (? a))
19   → Forall2_T (long_ty_T Gamma) ms ts →
20   long_ty_T Gamma (curry (! x) ms) (? a).
21
22 Inductive nfty_long (Gamma : repo) : nfterm → type → Type :=
23 | NFTy_lam_long s sigma tau : nfty_long (sigma :: Gamma) s tau → nfty_long
24   ↪ Gamma (N s) (sigma ↪ tau)
25 | NFTy_var_long : forall x a ts ms (Gammaok : nth_error Gamma x = Some
26   ↪ (make_arrow_type ts (? a)))
27   (Lenproof : length ms = length ts),
28   (forall n (pms : n < length ms),
29     nfty_long Gamma (nth_ok ms n pms) (nth_ok ts n (rew Lenproof in pms)))
30   ↪ →
31   nfty_long Gamma (!!x @@ ms) (? a).
32
33 Inductive nfty_long_subj : forall Gamma Gamma' m m' rho rho', nfty_long Gamma m
34   ↪ rho → nfty_long Gamma' m' rho' → Type :=
35 | nfty_long_refl : forall Gamma m rho (proof : nfty_long Gamma m rho),
36   ↪ nfty_long_subj _ _ _ _ _ proof proof
37 | nfty_long_trans : forall Gamma Gamma' Gamma'' m m' m'' rho rho' rho''
38   (proof1 : nfty_long Gamma m rho)
39   (proof2 : nfty_long Gamma' m' rho')
40   (proof3 : nfty_long Gamma'' m'' rho''),
41   nfty_long_subj _ _ _ _ _ proof1 proof2 →
42   nfty_long_subj _ _ _ _ _ proof2 proof3 →
43   nfty_long_subj _ _ _ _ _ proof1 proof3
44 | nfty_long_subj_I : forall Gamma sigma tau s (proof : nfty_long (sigma ::
45   ↪ Gamma) s tau),
46   nfty_long_subj _ _ _ _ _ proof (NFTy_lam_long _ _ _ _ proof)
47 | nfty_long_subj_E : forall Gamma x ts ms a
48   (Gammaok : nth_error Gamma x = Some (make_arrow_type ts
49   ↪ (? a)))
50   (Lenproof : length ms = length ts)
51   (proofs : (forall n (pms : n < length ms),
52     nfty_long Gamma (nth_ok ms n pms) (nth_ok
53   ↪ ts n (rew Lenproof in pms))))
54   n (len : n < length ms),
```

```

47   nfty_long_subj _ _ _ _ _ (proofs n len) (NFTy_var_long _ _ _ _ _ Gammaok
    ↪ Lenproof proofs).
48
49 Definition nflong_princ (rho: type) (M: nfterm) : Type :=
50   nfty_long [] M rho * forall rho', nfty_long [] M rho' → { Su & rho.[Su] =
    ↪ rho' }.
51
52 Lemma nfty_long_subterm : forall n m, subterm_nf n m → forall tau Gamma,
    ↪ nfty_long Gamma m tau → {Gamma' & {tau' & nfty_long Gamma' n tau'}}.
53 Proof.
54   induction 1; intros.
55   - exists Gamma. exists tau. assumption.
56   - inv X0. eapply IHX. exact X1.
57   - inv X0. apply In_nth_error_set in i. destruct i as [n H].
58     apply nth_error_nth_ok in H. destruct H as [lp H]. pose proof (X1 n lp).
    ↪ eapply IHX. rewrite H in X0. exact X0.
59 Qed.
60
61
62
63 Lemma Long_E_aux_T : forall Gamma x ms ts a curr v,
64   nth_error Gamma x = Some (make_arrow_type ts (? a))
65     → Forall2_T (long_ty_T Gamma) ms ts →
66     curr = curry (! x) ms → v = (? a) →
67     long_ty_T Gamma curr v.
68 Proof.
69   intros. subst. econstructor; try assumption.
70   - apply H.
71   - assumption.
72 Qed.
73
74 Definition long_ty_T_ind' :
75   forall P : repo → term → type → Type,
76   (forall (Gamma : repo) (s : term) (A B : type),
77     long_ty_T (A :: Gamma) s B →
78     P (A :: Gamma) s B → P Gamma (□ s) (A → B)) →
79   (forall (Gamma : repo) (x : var)
80     (ms : list term) (ts : list type) (a : var),
81     nth_error Gamma x = Some (make_arrow_type ts (? a)) →
82     Forall2_T (long_ty_T Gamma) ms ts →
83     Forall2_T (P Gamma) ms ts →
84     P Gamma (curry (! x) ms) (? a)) →
85   forall (Gamma : repo) (t : term) (t0 : type),
86   long_ty_T Gamma t t0 → P Gamma t t0 :=
87   fun P icase ecase ⇒
88   fix long_ty_ind'_rec (Gamma : repo) (t : term) (t0 : type)
89     (proof : long_ty_T Gamma t t0) {struct proof} : P Gamma t t0 :=
90     match proof with
91     | Long_I_T _ s A B proof' ⇒ icase Gamma s A B proof'
92       (long_ty_ind'_rec (A :: Gamma) s B proof')
93     | Long_E_T _ x ms ts a eqproof forallproof ⇒
94       ecase Gamma x ms ts a eqproof forallproof
95       ((fix forall_rec (ms : list term) (ts : list type)
96         (proof : Forall2_T (long_ty_T Gamma) ms ts) {struct
    ↪ proof}

```

```

97       : Forall2_T (P Gamma) ms ts :=
98       match proof with
99       | Forall2_T_nil _ => Forall2_T_nil _
100      | Forall2_T_cons _ m t ms ts headproof tailproof =>
101        Forall2_T_cons _ m t ms ts
102          (long_ty_ind'_rec Gamma m t headproof)
103          (forall_rec _ _ tailproof)
104      end) ms ts forallproof
105    )
106  end.
107  (*
108  Lemma Forall2_if_long_rel_T : forall Gamma ms ts, long_rel_T Gamma ms ts →
109    ⇔ Forall2_T (long_ty_T Gamma) ms ts.
110  Proof.
111    intros Gamma ms ts.
112    induction 1; constructor; try constructor; assumption.
113  Qed.
114  Lemma long_rel_if_Forall2_T : forall Gamma ms ts, Forall2_T (long_ty_T Gamma)
115    ⇔ ms ts → long_rel_T Gamma ms ts.
116  Proof.
117    intros Gamma ms ts.
118    induction 1; constructor; try constructor; assumption.
119  Qed.*)
120  Lemma Forall2_inh {B C}: forall (A : B → C → Type) ms ts, Forall2 (fun a b =>
121    ⇔ inhabited (A a b)) ms ts → inhabited (Forall2_T (fun a b => A a b) ms ts).
122  Proof.
123    induction 1.
124    - constructor. constructor.
125    - ainv. constructor. constructor.
126    + assumption.
127    + assumption.
128  Qed.
129  Lemma mkArrow_curry_ty_T : forall Gamma ms ts a ,
130    Forall2_T (fun m t => ty_T Gamma m t) ms ts
131    → forall x, ty_T Gamma x (make_arrow_type ts a)
132    → ty_T Gamma (curry x ms) a.
133  Proof.
134    induction 1.
135    - intros. simpl in *. assumption.
136    - intros. simpl in *. apply IHX. econstructor.
137    + apply X0.
138    + assumption.
139  Qed.
140  Lemma long_impl_ty_T : forall Gamma m t, long_ty_T Gamma m t → ty_T Gamma m t.
141  Proof.
142    intros. induction X using long_ty_T_ind'.
143    - constructor. assumption.
144    - eapply mkArrow_curry_ty_T.
145    + apply X0.
146    + constructor. assumption.
147  Qed.
148
149

```

```

150 Definition is_long_ty (t: term) (ty: type) := long_ty_T [] t ty.
151 Definition is_ty (t: term) (typ : type) := ty_T [] t typ.
152
153 Lemma long_ty_var_T : forall Gamma x t, nth_error Gamma x = Some (? t) →
  ↪ long_ty_T Gamma (! x) (? t).
154 Proof.
155   intros. assert (! x = curry (! x) []). { reflexivity. } rewrite H0.
156   ↪ econstructor.
157   - instantiate (1:=[]). auto.
158   - constructor.
159 Qed.
160 (*
161 Lemma long_rel_rev_T : forall ms ts Gamma, long_rel_T Gamma ms ts → long_rel_T
  ↪ Gamma (rev ms) (rev ts).
162 Proof.
163   intros. apply long_rel_if_Forall2_T. apply Forall2_T_is_rev. repeat rewrite
  ↪ rev_involutive.
164   apply Forall2_if_long_rel_T. assumption.
165 Qed.
166
167 Lemma rev_long_rel_T : forall ms ts Gamma, long_rel_T Gamma (rev ms) (rev ts) →
  ↪ long_rel_T Gamma ms ts.
168   intros. apply long_rel_if_Forall2_T. apply Forall2_T_is_rev_r. apply
  ↪ Forall2_if_long_rel_T. assumption.
169 Qed.)*
170
171 Lemma long_ty_app_T : forall Gamma n m ms t ts a x,
172   n = curry (! x) (ms) →
173   long_ty_T Gamma m t →
174   Forall2_T (long_ty_T Gamma) ms ts →
175   nth_error Gamma x = Some (make_arrow_type ts (t ↪ ? a))
176   → long_ty_T Gamma (n @ m) (? a).
177 Proof.
178   intros.
179   subst. rewrite ← curry_tail. econstructor.
180   - instantiate (1:=(ts ++ [t])).
181     rewrite make_arrow_type_last. assumption.
182   - apply Forall2_T_is_rev_r. repeat rewrite rev_unit.
183     constructor.
184     + assumption.
185     + apply Forall2_T_is_rev in X0. assumption.
186 Qed.
187
188 Lemma long_ty_lam_aux_T : forall m Gamma, { s & { t & long_ty_T (s :: Gamma) m t
  ↪ } } →
189   { t0 & long_ty_T Gamma (λ m) t0}.
190 Proof.
191   intros.
192   ainv. exists (x ↪ x0). constructor. assumption.
193 Qed.
194
195 Lemma long_general_T : forall m Su tau Gamma,
196   ty_T Gamma m tau → long_ty_T Gamma..[Su] m tau.[Su] → long_ty_T Gamma m tau.
197 Proof.

```

```

198 intros m.
199 remember (term_length m) as lengthm.
200 assert (term_length m ≤ lengthm). { firstorder. }
201 clear Heqlengthm.
202 revert m H.
203 induction (lengthm).
204 - intros. ex falso. ainv.
205 - intros. destruct m.
206 + ainv. symmetry in H4. apply curry_if_nil in H4. ainv.
207 apply subst_var_is_var_T in H1. ainv. apply Long_E_T with [].
208   { simpl. inv H1. reflexivity. }
209   { constructor. }
210 + inversion X0. apply subst_var_is_var_T in H1 as [b H1]. rewrite ← H0 in
    ↪ X.
211 apply mp_gen_T in X as [sigmas [HForall HGamma]].
212 rewrite H1 in *. apply Long_E_T with sigmas.
213 { assumption. }
214 { assert (Forall2_T (fun t sigma ⇒ t = sigma.[Su]) ts sigmas).
215   { rewrite subst_repo in H2. rewrite HGamma in H2. revert HForall H2 X1.
216     clear ...
217     revert ts sigmas.
218     induction ms.
219     - intros. inv HForall. inv X1. constructor.
220     - intros. inversion HForall. inversion X1. constructor.
221       { ainv. }
222       { ainv. apply IHms; try assumption.
223         - simpl. apply f_equal. assumption. } }
224 rewrite ← H0 in H.
225 generalize (curry_le (! x) ms _ H).
226 clear HGamma H2.
227 revert X1 HForall H3 IHn.
228 clear ...
229 revert sigmas ts.
230 induction ms.
231 - ainv. constructor.
232 - intros. ainv. constructor.
233   + apply IHn with Su.
234     { ainv. firstorder. }
235     { assumption. }
236     { assumption. }
237   + eapply IHms.
238     { eassumption. }
239     { assumption. }
240     { assumption. }
241     { assumption. }
242     { ainv. } }
243 + inversion X0. symmetry in H2. apply subst_arr_is_arr_or_T in H2 as [Harr |
    ↪ Hvar].
244   { destruct Harr as [st [st0 [Htau [HstSu Hst0su]]]].
245     rewrite Htau. constructor. apply IHn with Su.
246     - simpl in H. firstorder.
247     - inversion X. rewrite Htau in H0. ainv.
248     - rewrite ← HstSu in X1. rewrite subst_repo_cons in X1. rewrite Hst0su.
249       assumption.
250   }
251 { ainv. }

```

{ ainv. }

252

253

**Qed.**