

Nov 09, 15 7:02

L11\_in\_class.v

Page 1/11

```

Require Import List.
Require Import ZArith.
Require Import String.
Open Scope string_scope.

Ltac inv H := inversion H; subst.

Ltac break_match :=
  match goal with
  | _ : context [ if ?cond then _ else _ ] |- _ =>
    destruct cond as [] eqn:?
  | |- context [ if ?cond then _ else _ ] =>
    destruct cond as [] eqn:?
  | _ : context [ match ?cond with _ => _ end ] |- _ =>
    destruct cond as [] eqn:?
  | |- context [ match ?cond with _ => _ end ] =>
    destruct cond as [] eqn:?
  end.

(** syntax *)

Inductive expr : Set :=
| Bool : bool -> expr
| Int  : Z -> expr
| Var  : string -> expr
| App  : expr -> expr -> expr
| Lam  : string -> expr -> expr.

Coercion Bool : bool ->> expr.
Coercion Int  : Z ->> expr.
Coercion Var  : string ->> expr.

Notation "X@Y" := (App X Y) (at level 49).
Notation "\X,Y" := (Lam X Y) (at level 50).

(** substitution *)

(** e1[e2/x] = e3 *)
Inductive Subst : expr -> expr -> string ->
  expr -> Prop :=
| SubstBool:
  forall b e x,
  Subst (Bool b) e x
  (Bool b)
| SubstInt:
  forall i e x,
  Subst (Int i) e x
  (Int i)
| SubstVar_same:
  forall e x,
  Subst (Var x) e x
  e
| SubstVar_diff:
  forall e x1 x2,
  x1 <> x2 ->
  Subst (Var x1) e x2
  (Var x1)
| SubstApp:
  forall e1 e2 e x e1' e2',
  Subst e1 e x e1' ->
  Subst e2 e x e2' ->
  Subst (App e1 e2) e x
  (App e1' e2')
| SubstLam_same:
  forall e1 x e,
  Subst (Lam x e1) e x
  (Lam x e1)
| SubstLam_diff:
  forall e1 x1 x2 e e1',

```

Nov 09, 15 7:02

L11\_in\_class.v

Page 2/11

```

  x1 <> x2 ->
  Subst e1 e x2 e1' ->
  Subst (Lam x1 e1) e x2
  (Lam x1 e1').

(** careful to make IH sufficiently strong *)
Lemma subst_det:
  forall e1 e2 x e3,
  Subst e1 e2 x e3 ->
  forall e3',
  Subst e1 e2 x e3' ->
  e3 = e3'.
Proof.
  induction 1; intros; auto.
  - inv H; auto.
  - inv H; auto.
  - inv H; auto. congruence.
  - inv H0; auto. congruence.
  - inv H1.
  erewrite IHSubst1; eauto.
  erewrite IHSubst2; eauto.
  - inv H; auto. congruence.
  - inv H1; auto. congruence.
  erewrite IHSubst; eauto.
Qed.

Lemma can_subst:
  forall e1 e2 x,
  exists e3, Subst e1 e2 x e3.
Proof.
  induction e1; intros.
  - econstructor; constructor.
  - econstructor; constructor.
  - case (string_dec x s); intros.
  + subst. econstructor; constructor.
  + econstructor; constructor; auto.
  - edestruct IHel_1; edestruct IHel_2.
  econstructor; econstructor; eauto.
  - edestruct IHel.
  case (string_dec x s); intros.
  + subst. econstructor; constructor.
  + econstructor; constructor; eauto.
Qed.

(** define free variables *)
Inductive free : expr -> string -> Prop :=
| FreeVar:
  forall x,
  free (Var x) x
| FreeApp_l:
  forall x e1 e2,
  free e1 x ->
  free (App e1 e2) x
| FreeApp_r:
  forall x e1 e2,
  free e2 x ->
  free (App e1 e2) x
| FreeLam:
  forall x1 x2 e,
  free e x1 ->
  x1 <> x2 ->
  free (Lam x2 e) x1.

Lemma subst_only_free:
  forall e1 e2 x e3,
  Subst e1 e2 x e3 ->
  ~ free e1 x ->
  e1 = e3.
Proof.

```

Nov 09, 15 7:02 **L11\_in\_class.v** Page 3/11

```

induction l; intros; auto.
- destruct H. constructor.
- f_equal.
+ apply IHSubst1; intuition.
  apply H1; apply FreeApp_l; auto.
+ apply IHSubst2; intuition.
  apply H1; apply FreeApp_r; auto.
- rewrite IHSubst; auto.
  intuition. apply H1.
  constructor; auto.
Qed.

(** closed terms have no free variables *)
Definition closed (e: expr) : Prop :=
  forall x, ~ free e x.

Lemma closed_app_intro:
  forall e1 e2,
    closed e1 ->
    closed e2 ->
    closed (e1 @ e2).
Proof.
  unfold closed, not; intros.
  inv H1.
  - eapply H; eauto.
  - eapply H0; eauto.
Qed.

Lemma closed_app_inv:
  forall e1 e2,
    closed (e1 @ e2) ->
    closed e1 /\ closed e2.
Proof.
  unfold closed, not; split; intros.
  - eapply H; eauto.
  apply FreeApp_l; eauto.
  - eapply H; eauto.
  apply FreeApp_r; eauto.
Qed.

Lemma closed_lam_intro:
  forall x e,
    (forall y, y <> x -> ~ free e y) ->
    closed (\x, e).
Proof.
  unfold closed, not; intros.
  inv H0. eapply H; eauto.
Qed.

Lemma closed_lam_inv:
  forall x e,
    closed (\x, e) ->
    (forall y, y <> x -> ~ free e y).
Proof.
  unfold closed, not; intros.
  cut (free (\x, e) y); intros.
  - eapply H; eauto.
  - constructor; auto.
Qed.

(** closed-ness preserved by substitution *)
Lemma subst_pres_closed:
  forall e1 e2 x e3,
    Subst e1 e2 x e3 ->
    closed e1 ->
    closed e2 ->
    closed e3.
Proof.
  induction l; intros; auto.

```

Nov 09, 15 7:02 **L11\_in\_class.v** Page 4/11

```

- apply closed_app_inv in H1.
  apply closed_app_intro; intuition.
- apply subst_only_free in H0; subst; auto.
  unfold closed in *; intuition.
  eapply H1; eauto.
  econstructor; eauto.
Qed.

(**
Call By Name
<<
      e1 --> e1'
-----
      e1 e2 --> e1' e2
-----
      (\x. e1) e2 --> e1[e2/x]
>>
*)

Inductive step_cbn : expr -> expr -> Prop :=
| CBN_crunch:
  forall e1 e1' e2,
    step_cbn e1 e1' ->
    step_cbn (App e1 e2) (App e1' e2)
| CBN_subst:
  forall x e1 e2 e1',
    Subst e1 e2 x e1' ->
    step_cbn (App (Lam x e1) e2) e1'.

Notation "e1==>e2" := (step_cbn e1 e2) (at level 51).

Inductive star_cbn : expr -> expr -> Prop :=
| scbn_refl:
  forall e,
    star_cbn e e
| scbn_step:
  forall e1 e2 e3,
    step_cbn e1 e2 ->
    star_cbn e2 e3 ->
    star_cbn e1 e3.

Notation "e1==>*e2" := (star_cbn e1 e2) (at level 51).

Definition stuck (e: expr) : Prop :=
  forall e', ~ e ==> e'.

Lemma step_cbn_det:
  forall e e1,
    e ==> e1 ->
    forall e2,
    e ==> e2 ->
    e1 = e2.
Proof.
  induction l; intros.
  - inv H0.
    + f_equal. apply IHstep_cbn; auto.
    + inv H.
  - inv H0.
    + inv H4.
    + eapply subst_det; eauto.
Qed.

(** values *)

Inductive value : expr -> Prop :=
| VBool:
  forall b,
    value (Bool b)

```

Nov 09, 15 7:02 **L11\_in\_class.v** Page 5/11

```

| VInt:
  forall i,
    value (Int i)
| VLam:
  forall x e,
    value (Lam x e).

Lemma value_stuck:
  forall e,
    value e ->
    stuck e.
Proof.
  unfold stuck, not; intros;
  inv H; inv H0.
Qed.

(** types and typing *)

Inductive typ : Set :=
| TBool : typ
| TInt  : typ
| TFun  : typ -> typ -> typ.

Notation "X~>Y" := (TFun X Y) (at level 60).

Definition env : Type :=
  string -> option typ.

Definition E0 : env :=
  fun _ => None.

Definition extend (e: env) x t : env :=
  fun y =>
    if string_dec y x then
      Some t
    else
      e y.

Inductive typed : env -> expr -> typ -> Prop :=
| WTBool:
  forall env b,
    typed env (Bool b) TBool
| WTInt:
  forall env i,
    typed env (Int i) TInt
| WTVar:
  forall env x t,
    env x = Some t ->
    typed env (Var x) t
| WTApp:
  forall env e1 e2 tA tB,
    typed env e1 (tA ~> tB) ->
    typed env e2 tA ->
    typed env (e1 @ e2) tB
| WTLam:
  forall env x e tA tB,
    typed (extend env x tA) e tB ->
    typed env (\x, e) (tA ~> tB).

(** env must bind all free vars to type *)
Lemma typed_free_env:
  forall env e t,
    typed env e t ->
    forall x,
      free e x ->
      exists tx, env x = Some tx.
Proof.
  induction 1; intros.
  - inv H.

```

Nov 09, 15 7:02 **L11\_in\_class.v** Page 6/11

```

- inv H.
- inv H0; eauto.
- inv H1.
+ apply IHtyped1; auto.
+ apply IHtyped2; auto.
- inv H0. apply IHtyped in H3.
  destruct H3 as [tx Htx].
  exists tx. unfold extend in Htx.
  break_match; congruence.
Qed.

(** therefore, typing in empty env
implies term is closed *)
Lemma typed_E0_closed:
  forall e t,
    typed E0 e t ->
    closed e.
Proof.
  unfold closed, not; intros.
  eapply typed_free_env in H0; eauto.
  destruct H0. discriminate.
Qed.

(** canonical forms *)

Lemma cannon_bool:
  forall env e,
    value e ->
    typed env e TBool ->
    exists b, e = Bool b.
Proof.
  intros.
  inv H; inv H0; eauto.
Qed.

Lemma cannon_int:
  forall env e,
    value e ->
    typed env e TInt ->
    exists i, e = Int i.
Proof.
  intros.
  inv H; inv H0; eauto.
Qed.

Lemma cannon_fun:
  forall env e tA tB,
    value e ->
    typed env e (tA ~> tB) ->
    exists x, exists b, e = \x, b.
Proof.
  intros.
  inv H; inv H0; eauto.
Qed.

(** progress *)

Lemma progress:
  forall e t,
    typed E0 e t ->
    (exists e', e ==> e') \/ value e.
Proof.
  remember E0.
  induction 1; subst; intros.
  - right; constructor.
  - right; constructor.
  - unfold E0 in H; congruence.
  - left. destruct IHtyped1; auto.
    + destruct H1 as [e1']. exists (e1' @ e2).

```

Nov 09, 15 7:02

L11\_in\_class.v

Page 7/11

```

    constructor; auto.
+ eapply cannon_fun in H1; eauto.
    destruct H1 as [x [e1' Hel']]; subst.
    destruct (can_subst e1' e2 x) as [e3].
    exists e3. constructor; auto.
- right; constructor.
Qed.

(** preservation *)

Definition env_equiv (e1 e2: env) : Prop :=
  forall s, e1 s = e2 s.

Lemma env_equiv_refl:
  forall env,
    env_equiv env env.
Proof.
  unfold env_equiv; auto.
Qed.

Lemma env_equiv_sym:
  forall e1 e2,
    env_equiv e1 e2 ->
    env_equiv e2 e1.
Proof.
  unfold env_equiv; auto.
Qed.

Lemma env_equiv_trans:
  forall e1 e2 e3,
    env_equiv e1 e2 ->
    env_equiv e2 e3 ->
    env_equiv e1 e3.
Proof.
  unfold env_equiv; intros.
  congruence.
Qed.

Lemma env_equiv_extend:
  forall env1 env2 x t,
    env_equiv env1 env2 ->
    env_equiv (extend env1 x t) (extend env2 x t).
Proof.
  unfold env_equiv, extend; intros.
  break_match; auto.
Qed.

Lemma env_equiv_overwrite:
  forall env x t1 t2,
    env_equiv (extend (extend env x t1) x t2)
      (extend env x t2).
Proof.
  unfold env_equiv, extend; intros.
  break_match; auto.
Qed.

Lemma env_equiv_neq:
  forall env1 env2 x1 t1 x2 t2,
    x1 <> x2 ->
    env_equiv env1 env2 ->
    env_equiv (extend (extend env1 x1 t1) x2 t2)
      (extend (extend env2 x2 t2) x1 t1).
Proof.
  unfold env_equiv, extend; intros.
  break_match; break_match; congruence.
Qed.

Lemma env_equiv_typed:
  forall env1 e t,

```

Nov 09, 15 7:02

L11\_in\_class.v

Page 8/11

```

  typed env1 e t ->
  forall env2,
    env_equiv env1 env2 ->
    typed env2 e t.
Proof.
  unfold env_equiv.
  induction l; intros.
  - constructor.
  - constructor.
  - constructor; congruence.
  - econstructor; eauto.
  - econstructor; eauto.
  apply IHtyped; auto.
  intros; apply env_equiv_extend; auto.
Qed.

Lemma strengthen:
  forall e env t x t',
    typed (extend env x t') e t ->
    ~ free e x ->
    typed env e t.
Proof.
  induction e; intros; inv H.
  - constructor.
  - constructor.
  - constructor. unfold extend in H3.
    break_match; subst; auto.
    destruct H0. constructor.
  - econstructor; eauto.
  + eapply IHel; eauto. intuition.
    apply H0; apply FreeApp_l; auto.
  + eapply IHe2; eauto. intuition.
    apply H0; apply FreeApp_r; auto.
  - constructor.
  case (string_dec s x); intros; subst.
  + eapply env_equiv_typed; eauto.
    apply env_equiv_overwrite.
  + cut (~ free e x); intros.
    * eapply IHe; eauto.
    eapply env_equiv_typed; eauto.
    apply env_equiv_neq; auto.
    apply env_equiv_refl.
    * intuition. apply H0.
    constructor; auto.
Qed.

Lemma weaken:
  forall env e t,
    typed env e t ->
    forall x t',
      ~ free e x ->
      typed (extend env x t') e t.
Proof.
  induction l; intros.
  - constructor.
  - constructor.
  - constructor. unfold extend.
    break_match; subst; auto.
    destruct H0. constructor.
  - econstructor; eauto.
  + apply IHtyped1. intuition.
    apply H1; apply FreeApp_l; auto.
  + apply IHtyped2. intuition.
    apply H1; apply FreeApp_r; auto.
  - constructor.
  case (string_dec x x0); intros; subst.
  + eapply env_equiv_typed; eauto.
    apply env_equiv_sym.
    apply env_equiv_overwrite.

```

Nov 09, 15 7:02

L11\_in\_class.v

Page 9/11

```
+ cut (~ free e x0); intros.
* apply IHtyped with (t' := t') in H1; auto.
  eapply env_equiv_typed; eauto.
  apply env_equiv_neq; auto.
  apply env_equiv_refl.
* intuition. apply H0.
  constructor; auto.
```

**Qed.**

```
Definition free_env_equiv (E: expr) (e1 e2: env) : Prop :=
forall x,
  free E x ->
  e1 x = e2 x.
```

```
Lemma free_env_equiv_refl:
forall E env,
  free_env_equiv E env env.
```

```
Proof.
  unfold free_env_equiv; auto.
```

**Qed.**

```
Lemma free_env_equiv_sym:
forall E e1 e2,
  free_env_equiv E e1 e2 ->
  free_env_equiv E e2 e1.
```

```
Proof.
  unfold free_env_equiv; intros.
  symmetry. apply H; auto.
```

**Qed.**

```
Lemma free_env_equiv_trans:
forall E e1 e2 e3,
  free_env_equiv E e1 e2 ->
  free_env_equiv E e2 e3 ->
  free_env_equiv E e1 e3.
```

```
Proof.
  unfold free_env_equiv; intros.
  apply eq_trans with (y := e2 x); auto.
```

**Qed.**

```
Lemma free_env_equiv_extend:
forall E env1 env2 x t,
  free_env_equiv E env1 env2 ->
  free_env_equiv E (extend env1 x t) (extend env2 x t).
```

```
Proof.
  unfold free_env_equiv, extend; intros.
  break_match; auto.
```

**Qed.**

```
Lemma free_env_equiv_overwrite:
forall E env x t1 t2,
  free_env_equiv E (extend (extend env x t1) x t2)
    (extend env x t2).
```

```
Proof.
  unfold free_env_equiv, extend; intros.
  break_match; auto.
```

**Qed.**

```
Lemma free_env_equiv_neq:
forall E env1 env2 x1 t1 x2 t2,
  x1 <> x2 ->
  free_env_equiv E env1 env2 ->
  free_env_equiv E (extend (extend env1 x1 t1) x2 t2)
    (extend (extend env2 x2 t2) x1 t1).
```

```
Proof.
  unfold free_env_equiv, extend; intros.
  break_match; break_match; subst; auto.
  congruence.
```

**Qed.**

Nov 09, 15 7:02

L11\_in\_class.v

Page 10/11

```
Lemma free_env_equiv_typed:
forall env1 e t,
  typed env1 e t ->
  forall env2,
  free_env_equiv e env1 env2 ->
  typed env2 e t.
```

**Proof.**

```
  unfold free_env_equiv.
  induction 1; intros.
  - constructor.
  - constructor.
  - constructor. symmetry.
    rewrite <- H. apply H0.
    constructor.
  - econstructor; eauto.
    + apply IHtyped1; intuition.
      apply H1; apply FreeApp_l; auto.
    + apply IHtyped2; intuition.
      apply H1; apply FreeApp_r; auto.
  - econstructor; eauto.
    apply IHtyped; auto.
  unfold extend; intros.
  break_match; auto.
  apply H0. constructor; auto.
```

**Qed.**

```
Lemma typed_closed:
forall env e t,
  typed env e t ->
  closed e ->
  typed E0 e t.
```

**Proof.**

```
  induction 1; intros.
  - constructor.
  - constructor.
  - unfold closed in H0.
    destruct H0 with (x0 := x).
    constructor.
  - apply closed_app_inv in H1; intuition.
    econstructor; eauto.
  - constructor.
    eapply free_env_equiv_typed; eauto.
    unfold free_env_equiv; intros.
    unfold extend. break_match; auto.
    apply closed_lam_inv with (y := x0) in H0; auto.
    contradiction.
```

**Qed.**

```
Lemma subst_pres_typed:
forall e1 e2 x e3,
  Subst e1 e2 x e3 ->
  closed e2 ->
  forall env tA tB,
  typed (extend env x tA) e1 tB ->
  typed env e2 tA ->
  typed env e3 tB.
```

**Proof.**

```
  induction 1; intros; auto.
  - inv H0. constructor.
  - inv H0. constructor.
  - inv H0. unfold extend in H4.
    break_match; congruence.
  - inv H1. unfold extend in H5.
    break_match; try congruence.
    constructor; auto.
  - inv H2. econstructor; eauto.
  - eapply free_env_equiv_typed; eauto.
    unfold free_env_equiv, extend; intros.
```

Nov 09, 15 7:02

L11\_in\_class.v

Page 11/11

```

break_match; auto; subst.
inv H2; congruence.
- inv H2. constructor.
  eapply IHSubst; eauto.
+ eapply env_equiv_typed; eauto.
  apply env_equiv_neq; auto.
  apply env_equiv_refl.
+ apply weaken; auto.
Qed.

Lemma preserve:
  forall e e',
    e ==> e' ->
      closed e ->
        forall env t,
          typed env e t ->
            typed env e' t.
Proof.
  induction 1; intros.
- apply closed_app_inv in H0; intuition.
  inv H1. apply H0 in H7.
  econstructor; eauto.
- apply closed_app_inv in H0; intuition.
  inv H1. inv H6.
  eapply subst_pres_typed in H; eauto.
Qed.

(** type soundness *)

Lemma soundness:
  forall e t e',
    typed E0 e t ->
      e ==>* e' ->
        (exists e'', e' ==> e'') \/ value e'.
Proof.
  intros. induction H0.
- eapply progress; eauto.
- apply IHstar_cbn.
  eapply preserve; eauto.
  eapply typed_E0_closed; eauto.
Qed.

```