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

Inductive expr : Set :=
| Var : string -> expr
| App : expr -> expr -> expr
| Lam : string -> expr -> expr.

Coercion Var : string >-> expr.

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

Check (\"x", \"y", "x").
Check (\"x", \"y", "y").
Check ((\"x", "x" @ "x") @ (\"x", "x" @ "x")).

(** e1[e2/x] = e3 *)
Inductive Subst : expr -> expr -> string ->
                  expr -> Prop :=
| 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',
      x1 <> x2 ->
      Subst e1 e x2 e1' ->
      Subst (Lam x1 e1) e x2 (Lam x1 e1').

Lemma subst_test_1:
  Subst (\"x", "y") "z" "y"
        (\"x", "z").
Proof.
  constructor.
  - discriminate.
  - constructor.
Qed.

Lemma subst_test_2:
  Subst (\"x", "x") "z" "x"
        (\"x", "x").
Proof.
  constructor.
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).

Lemma sstep_test_1:
  (\"x", "x") @ "z" ==> "z".
Proof.
  apply CBN_subst.
  apply SubstVar_same.
Qed.

Lemma Lam_nostep_cbn:
  forall x e1 e2,
  ~ (\x, e1 ==> e2).
Proof.
  intros. intro. inversion H.
Qed.

Ltac inv H := inversion H; subst.

(** 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.
  - inv H; auto. congruence.
  - inv H0; auto. congruence.
  - inv H1.
    apply IHSubst1 in H4.
    apply IHSubst2 in H8.
    subst; auto.
  - inv H; auto. congruence.
  - inv H1; auto. congruence.
    apply IHSubst in H8; subst; auto.
Qed.

Lemma step_cbn_det:
  forall e e1,
  e ==> e1 ->
  forall e2,
  e ==> e2 ->
  e1 = e2.
Proof.
  induction 1; intros.
  - inv H0.
    + f_equal. apply IHstep_cbn; auto.
    + exfalso. apply Lam_nostep_cbn in H; auto.
  - inv H0.
    + exfalso. apply Lam_nostep_cbn in H4; auto.
    + eapply Subst_det; eauto.
Qed.

(**
Call By Value
<<
```

```
v ::= \ x . e

       e1 --> e1'
  ---------------------
    e1 e2 --> e1' e2

       e2 --> e2'
  ---------------------
    v e2 --> v e2'

  -----------------------------
    (\x. e1) v --> e1[v/x]
>>
*)

Inductive value : expr -> Prop :=
| VLam :
    forall x e,
      value (Lam x e).

Inductive step_cbv : expr -> expr -> Prop :=
| CBV_crunch_l:
    forall e1 e1' e2,
      step_cbv e1 e1' ->
      step_cbv (App e1 e2) (App e1' e2)
| CBV_crunch_r:
    forall v e2 e2',
      value v ->
      step_cbv e2 e2' ->
      step_cbv (App v e2) (App v e2')
| CBV_subst:
    forall x e1 v e1',
      value v ->
      Subst e1 v x e1' ->
      step_cbv (App (Lam x e1) v) e1'.

Notation "e1 --> e2" := (step_cbv e1 e2) (at level 51).
```