```
Require Import List.
Require Import String.
Require Import ZArith.

Open Scope list_scope.
Open Scope string_scope.
Open Scope Z_scope.

Require Import StructTactics.
Require Import ImpSyntax.
Require Import ImpCommon.
Require Import ImpEval.
Require Import ImpStep.


(** eval facts *)

Lemma eval_unop_det :
  forall op v v1 v2,
    eval_unop op v v1 ->
    eval_unop op v v2 ->
    v1 = v2.
Proof.
  intros.
  repeat on (eval_unop _ _ _), invc; auto.
Qed.

Lemma eval_binop_det :
  forall op vL vR v1 v2,
    eval_binop op vL vR v1 ->
    eval_binop op vL vR v2 ->
    v1 = v2.
Proof.
  intros.
  repeat on (eval_binop _ _ _ _), invc; auto.
Qed.

Lemma eval_e_det :
  forall s e v1 v2,
    eval_e s e v1 ->
    eval_e s e v2 ->
    v1 = v2.
Proof.
  intros. prep_induction H.
  induction H; intros;
    on (eval_e _ _ _), invc;
    repeat find_apply_hyp_hyp;
    subst; try congruence.
  - find_eapply_lem_hyp eval_unop_det; eauto.
  - find_eapply_lem_hyp eval_binop_det; eauto.
Qed.

(** step facts *)

Lemma nostep_nop :
  forall s s' p',
    ~ step
      s Snop
      s' p'.
Proof.
  unfold not; intros. inv H.
Qed.

Lemma nostep_self' :
  forall s1 p1 s2 p2,
    step
      s1 p1
      s2 p2 ->
    p1 <> p2.
```



```
Proof.
  induction 1; try congruence.
  - induction p1; congruence.
  - induction p1; congruence.
  - induction p2; congruence.
  - induction p2; congruence.
Qed.

Lemma nostep_self :
  forall s1 p1 s2,
    ~ step
      s1 p1
      s2 p1.
Proof.
  unfold not; intros.
  find_apply_lem_hyp nostep_self'.
  congruence.
Qed.

Lemma step_star_1 :
  forall s1 p1 s2 p2,
    step
      s1 p1
      s2 p2 ->
    step_star
      s1 p1
      s2 p2.
Proof.
  repeat ee.
Qed.

Lemma step_star_r :
  forall s1 p1 s2 p2 s3 p3,
    step_star
      s1 p1
      s2 p2 ->
    step
      s2 p2
      s3 p3 ->
    step_star
      s1 p1
      s3 p3.
Proof.
  induction 1; repeat ee.
Qed.

Lemma step_star_app :
  forall s1 p1 s2 p2 s3 p3,
    step_star
      s1 p1
      s2 p2 ->
    step_star
      s2 p2
      s3 p3 ->
    step_star
      s1 p1
      s3 p3.
Proof.
  intros. prep_induction H.
  induction H; intros.
  - assumption.
  - ee.
Qed.

Lemma step_star_seq :
  forall s1 p1 s2 p2 pB,
    step_star
      s1 p1
      s2 p2 ->
    step_star
```

```
      s1 (Sseq p1 pB)
      s2 (Sseq p2 pB).
Proof.
  intros. prep_induction H.
  induction H; repeat ee.
Qed.

Lemma eval_step_star :
  forall s p s',
    eval_s
      s p s' ->
    step_star
      s p
      s' Snop.
Proof.
  induction 1;
    try (do 2 ee; fail).
  - eapply step_star_l; [ee|].
    eapply step_star_app.
    + eapply step_star_seq; eauto.
    + eapply step_star_l; [ee|].
      assumption.
  - eapply step_star_app.
    + eapply step_star_seq; eauto.
    + eapply step_star_l; [ee|].
      assumption.
Qed.

Ltac invc_eval_s :=
  on (eval_s _ _ _), invc.

Lemma step_eval_eval :
  forall s1 p1 s2 p2 s3,
    step
      s1 p1
      s2 p2 ->
    eval_s
      s2 p2 s3 ->
    eval_s
      s1 p1 s3.
Proof.
  intros. prep_induction H.
  induction H; intros.
  - invc_eval_s; repeat ee.
  - eauto using eval_ifelse_t.
  - eauto using eval_ifelse_f.
  - invc_eval_s; repeat ee.
  - invc_eval_s.
    eauto using eval_while_f.
  - repeat ee.
  - invc_eval_s; repeat ee.
Qed.

Lemma step_star_eval :
  forall s1 p1 s2,
    step_star
      s1 p1
      s2 Snop ->
    eval_s
      s1 p1 s2.
Proof.
  intros. prep_induction H.
  induction H; intros; subst.
  - ee.
  - eauto using step_eval_eval.
Qed.
```