```
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.
Require Import ImpKStep.

(** misc *)

Lemma locate_inv :
  forall env x name params body ret,
    locate env x = Some (Func name params body ret) ->
    x = name.
Proof.
  induction env; simpl; intros.
  - congruence.
  - repeat break_match; subst.
    + congruence.
    + find_apply_hyp_hyp; auto.
Qed.

(** 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 h e v1 v2,
    eval_e s h e v1 ->
    eval_e s h 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.

Lemma eval_e_len_a_inv :
  forall s h e x a l,
    eval_e s h (Elen e) x ->
```



```
    eval_e s h e (Vaddr a) ->
    read h a = Some (Vint l) ->
    x = Vint l.
Proof.
  intros.
  invc H.
  - eapply ImpSemanticsFacts.eval_e_det in H0; eauto. congruence.
  - eapply ImpSemanticsFacts.eval_e_det in H0; eauto. congruence.
Qed.

Lemma eval_e_idx_a_inv :
  forall s h e1 e2 x a i l,
    eval_e s h (Eidx e1 e2) x ->
    eval_e s h e1 (Vaddr a) ->
    eval_e s h e2 (Vint i) ->
    read h a = Some (Vint l) ->
    read h (Z.succ (a + i)) = Some x.
Proof.
  intros.
  invc H.
  - eapply ImpSemanticsFacts.eval_e_det in H0; eauto.
    eapply ImpSemanticsFacts.eval_e_det in H1; eauto.
    congruence.
  - eapply ImpSemanticsFacts.eval_e_det in H0; eauto.
    congruence.
Qed.

(** step facts *)

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

Lemma nostep_self' :
  forall env s1 h1 p1 s2 h2 p2,
    step env
      s1 h1 p1
      s2 h2 p2 ->
    p1 <> p2.
Proof.
  induction 1; try congruence.
  - induction p1; congruence.
  - induction p2; congruence.
  - induction p2; congruence.
Qed.

Lemma nostep_self :
  forall env s1 h1 p1 s2 h2,
    ~ step env
      s1 h1 p1
      s2 h2 p1.
Proof.
  unfold not; intros.
  find_apply_lem_hyp nostep_self'.
  congruence.
Qed.

Lemma step_star_1 :
  forall env s1 h1 p1 s2 h2 p2,
    step env
      s1 h1 p1
      s2 h2 p2 ->
    step_star env
      s1 h1 p1
```

```
      s2 h2 p2.
Proof.
  repeat ee.
Qed.

Lemma step_star_r :
  forall env s1 h1 p1 s2 h2 p2 s3 h3 p3,
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    step env
      s2 h2 p2
      s3 h3 p3 ->
    step_star env
      s1 h1 p1
      s3 h3 p3.
Proof.
  induction 1; repeat ee.
Qed.

Lemma step_star_app :
  forall env s1 h1 p1 s2 h2 p2 s3 h3 p3,
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    step_star env
      s2 h2 p2
      s3 h3 p3 ->
    step_star env
      s1 h1 p1
      s3 h3 p3.
Proof.
  intros. prep_induction H.
  induction H; intros.
  - assumption.
  - ee.
Qed.

Lemma step_star_seq :
  forall env s1 h1 p1 s2 h2 p2 pB,
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    step_star env
      s1 h1 (Sseq p1 pB)
      s2 h2 (Sseq p2 pB).
Proof.
  intros. prep_induction H.
  induction H; repeat ee.
Qed.

Lemma step_star_incall :
  forall env s1 h1 p1 s2 h2 p2 rl ret rs,
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    step_star env
      s1 h1 (Sincall p1 rl ret rs)
      s2 h2 (Sincall p2 rl ret rs).
Proof.
  intros. prep_induction H.
  induction H; repeat ee.
Qed.

Lemma eval_step_star :
  forall env s h p s' h',
    eval_s env
      s h p
      s' h' ->
```



```
    step_star env
      s h p
      s' h' Snop.
Proof.
  induction 1;
    try (do 2 ee; fail).
  - eapply step_star_l; [ee|].
    eapply step_star_r.
    + eapply step_star_incall; eauto.
    + ee.
  - 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.
  - eapply step_star_r.
    + eapply step_star_incall; eauto.
    + ee.
Qed.

Lemma eval_p_steps_p :
    forall p v,
      eval_p p v ->
      steps_p p v.
Proof.
  intros. on (eval_p _ _), invc. ee.
  eauto using eval_step_star.
Qed.

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

Lemma step_eval_eval :
  forall env s1 h1 p1 s2 h2 p2 s3 h3,
    step env
      s1 h1 p1
      s2 h2 p2 ->
    eval_s env
      s2 h2 p2
      s3 h3 ->
    eval_s env
      s1 h1 p1
      s3 h3.
Proof.
  intros. prep_induction H.
  induction H; intros.
  - invc_eval_s; repeat ee.
  - invc_eval_s; repeat ee.
  - invc_eval_s; repeat ee.
  - invc_eval_s; repeat ee.
  - invc_eval_s.
    eauto using eval_call_external.
  - 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.
  - invc_eval_s; repeat ee.
  - invc_eval_s; repeat ee.
Qed.

Lemma step_star_eval :
  forall env s1 h1 p1 s2 h2,
```

```
    step_star env
      s1 h1 p1
      s2 h2 Snop ->
    eval_s env
      s1 h1 p1
      s2 h2.
Proof.
  intros. prep_induction H.
  induction H; intros; subst.
  - ee.
  - eauto using step_eval_eval.
Qed.

Lemma steps_p_eval_p :
  forall p v,
    steps_p p v ->
    eval_p p v.
Proof.
  intros. on (steps_p _ _), invc. ee.
  eauto using step_star_eval.
Qed.

(** kstep facts *)

Ltac inv_kstep  := on (kstep _ _ _ _ _ _ _ _ _), inv.
Ltac invc_kstep := on (kstep _ _ _ _ _ _ _ _ _), invc.
Ltac inv_kstep_star  := on (kstep_star _ _ _ _ _ _ _ _ _), inv.
Ltac invc_kstep_star := on (kstep_star _ _ _ _ _ _ _ _ _), invc.

Lemma nop_nokstep :
  forall env s h s' h' p' k',
    ~ kstep env
        s h Snop Kstop
        s' h' p' k'.
Proof.
  unfold not; intros. inv H.
Qed.

Lemma kstep_star_1 :
  forall env s1 h1 p1 k1 s2 h2 p2 k2,
    kstep env
      s1 h1 p1 k1
      s2 h2 p2 k2 ->
    kstep_star env
      s1 h1 p1 k1
      s2 h2 p2 k2.
Proof.
  repeat ee.
Qed.

Lemma kstep_star_r :
  forall env s1 h1 p1 k1 s2 h2 p2 k2 s3 h3 p3 k3,
    kstep_star env
      s1 h1 p1 k1
      s2 h2 p2 k2 ->
    kstep env
      s2 h2 p2 k2
      s3 h3 p3 k3 ->
    kstep_star env
      s1 h1 p1 k1
      s3 h3 p3 k3.
Proof.
  induction 1; repeat ee.
Qed.

Lemma kstep_star_app :
  forall env s1 h1 p1 k1 s2 h2 p2 k2 s3 h3 p3 k3,
    kstep_star env
      s1 h1 p1 k1
```



```
      s2 h2 p2 k2 ->
    kstep_star env
      s2 h2 p2 k2
      s3 h3 p3 k3 ->
    kstep_star env
      s1 h1 p1 k1
      s3 h3 p3 k3.
Proof.
  intros. prep_induction H.
  induction H; intros.
  - assumption.
  - ee.
Qed.

Inductive kstep_plus (env : env) :
  store -> heap -> stmt -> cont ->
  store -> heap -> stmt -> cont -> Prop :=
| kstep_plus_l :
    forall s1 h1 p1 k1 s2 h2 p2 k2 s3 h3 p3 k3,
      kstep env
        s1 h1 p1 k1
        s2 h2 p2 k2 ->
      kstep_star env
        s2 h2 p2 k2
        s3 h3 p3 k3 ->
      kstep_plus env
        s1 h1 p1 k1
        s3 h3 p3 k3.

Ltac inv_kstep_plus  := on (kstep_plus _ _ _ _ _ _ _ _ _), inv.
Ltac invc_kstep_plus := on (kstep_plus _ _ _ _ _ _ _ _ _), invc.

Lemma kstep_plus_star :
  forall env s1 h1 p1 k1 s2 h2 p2 k2,
    kstep_plus env
      s1 h1 p1 k1
      s2 h2 p2 k2 ->
    kstep_star env
      s1 h1 p1 k1
      s2 h2 p2 k2.
Proof.
  intros. inv_kstep_plus. ee.
Qed.

Fixpoint kapp (k1 k2 : cont) : cont :=
  match k1 with
  | Kstop => k2
  | Kseq p k => Kseq p (kapp k k2)
  | Kcall ret x sf k => Kcall ret x sf (kapp k k2)
  end.

Lemma kapp_kstop_l :
  forall k,
    k = kapp Kstop k.
Proof.
  auto.
Qed.

Lemma kapp_kstop_r :
  forall k,
    kapp k Kstop = k.
Proof.
  induction k; simpl; auto.
  - find_rewrite; auto.
  - find_rewrite; auto.
Qed.

Lemma kapp_kseq :
  forall p k1 k2,
```

```
    Kseq p (kapp k1 k2) = kapp (Kseq p k1) k2.
Proof.
  auto.
Qed.

Lemma kapp_kcall :
  forall ret x s k1 k2,
    Kcall ret x s (kapp k1 k2) =
      kapp (Kcall ret x s k1) k2.
Proof.
  auto.
Qed.

Lemma kapp_assoc :
  forall k1 k2 k3,
    kapp (kapp k1 k2) k3 = kapp k1 (kapp k2 k3).
Proof.
  induction k1; simpl; intros.
  - auto.
  - f_equal; auto.
  - f_equal; auto.
Qed.

Lemma kstep_kapp :
  forall env s1 h1 p1 k1 s2 h2 p2 k2 kB,
    kstep env
      s1 h1 p1 k1
      s2 h2 p2 k2 ->
    kstep env
      s1 h1 p1 (kapp k1 kB)
      s2 h2 p2 (kapp k2 kB).
Proof.
  intros. prep_induction H.
  induction H; repeat ee.
Qed.

Lemma kstep_star_kapp :
  forall env s1 h1 p1 k1 s2 h2 p2 k2 kB,
    kstep_star env
      s1 h1 p1 k1
      s2 h2 p2 k2 ->
    kstep_star env
      s1 h1 p1 (kapp k1 kB)
      s2 h2 p2 (kapp k2 kB).
Proof.
  intros. prep_induction H.
  induction H; repeat ee.
  apply kstep_kapp; auto.
Qed.

Lemma kstep_plus_kapp :
  forall env s1 h1 p1 k1 s2 h2 p2 k2 kB,
    kstep_plus env
      s1 h1 p1 k1
      s2 h2 p2 k2 ->
    kstep_plus env
      s1 h1 p1 (kapp k1 kB)
      s2 h2 p2 (kapp k2 kB).
Proof.
  intros. inv H. ee.
  - eapply kstep_kapp; eauto.
  - eapply kstep_star_kapp; eauto.
Qed.

Inductive kmatch :
  stmt -> stmt -> cont -> Prop :=
| kmatch_refl :
    forall p,
      kmatch p p Kstop
```



```
| kmatch_seq :
    forall p1 pk k p2,
      kmatch p1 pk k ->
      kmatch (Sseq p1 p2)
             pk (kapp k (Kseq p2 Kstop))
| kmatch_incall :
    forall p1 pk k ret x s,
      kmatch p1 pk k ->
      kmatch (Sincall p1 ret x s)
             pk (kapp k (Kcall ret x s Kstop)).

Ltac inv_kmatch  := on (kmatch _ _ _), inv.
Ltac invc_kmatch := on (kmatch _ _ _), invc.

Lemma step_kstep_plus :
  forall env s1 h1 p1 s2 h2 p2 pA kA,
    kmatch p1 pA kA ->
    step env
      s1 h1 p1
      s2 h2 p2 ->
    exists pB, exists kB,
      kstep_plus env
        s1 h1 pA kA
        s2 h2 pB kB
      /\ kmatch p2 pB kB.
Proof.
  intros. prep_induction H0.
  induction H0; intros; try (
    inv_kmatch; do 5 ee; fail).
  - inv_kmatch. exists body; repeat ee.
    rewrite kapp_kstop_l; repeat ee.
  - inv_kmatch. exists p; repeat ee.
    rewrite kapp_kstop_l; repeat ee.
  - inv_kmatch.
    + exists p2; repeat ee.
    + on (kmatch Snop _ _), invc.
      exists p2; repeat ee.
  - inv_kmatch.
    + assert (kmatch p1 p1 Kstop) by ee.
      find_apply_hyp_hyp.
      break_exists; break_and.
      exists pB. exists (kapp kB (Kseq p2 Kstop)).
      split; [|ee]. ee. ee.
      rewrite kapp_kstop_l at 1.
      find_apply_lem_hyp kstep_plus_star.
      apply kstep_star_kapp; auto.
    + find_apply_hyp_hyp.
      break_exists; break_and.
      exists pB. exists (kapp kB (Kseq p2 Kstop)).
      split; [|ee].
      apply kstep_plus_kapp; auto.
  - inv_kmatch.
    + exists Snop; repeat ee.
    + on (kmatch Snop _ _), invc.
      exists Snop; repeat ee.
  - inv_kmatch.
    + assert (kmatch p p Kstop) by ee.
      find_apply_hyp_hyp.
      break_exists; break_and.
      exists pB. exists (kapp kB (Kcall e lr sr Kstop)).
      split; [|ee]. ee. ee.
      rewrite kapp_kstop_l at 1.
      find_apply_lem_hyp kstep_plus_star.
      apply kstep_star_kapp; auto.
    + find_apply_hyp_hyp.
      break_exists; break_and.
      exists pB. exists (kapp kB (Kcall e lr sr Kstop)).
      split; [|ee].
      apply kstep_plus_kapp; auto.
```

```
Qed.

Lemma step_star_kstep_star :
  forall env s1 h1 p1 s2 h2 p2 pA kA,
    kmatch p1 pA kA ->
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    exists pB, exists kB,
      kstep_star env
        s1 h1 pA kA
        s2 h2 pB kB
      /\ kmatch p2 pB kB.
Proof.
  intros. prep_induction H0.
  induction H0; intros.
  - repeat ee.
  - find_eapply_lem_hyp step_kstep_plus; eauto.
    break_exists; break_and.
    find_apply_hyp_hyp; eauto.
    break_exists; break_and.
    exists pB0; exists kB0; split; auto.
    invc_kstep_plus.
    eapply kstep_star_l; eauto.
    eapply kstep_star_app; eauto.
Qed.

Lemma steps_p_ksteps_p :
  forall p v,
    steps_p p v ->
    ksteps_p p v.
Proof.
  intros. on (steps_p _ _), invc. ee.
  apply step_star_kstep_star
    with (pA := main) (kA := Kstop) in H0.
  - break_exists; break_and.
    on (kmatch _ _ _), inv.
    assumption.
  - ee.
Qed.

Fixpoint uncont (p : stmt) (k : cont) : stmt :=
  match k with
  | Kstop =>
      p
  | Kseq p' k' =>
      uncont (Sseq p p') k'
  | Kcall ret x s k' =>
      uncont (Sincall p ret x s) k'
  end.

Lemma uncont_kstop :
  forall p,
    p = uncont p Kstop.
Proof.
  auto.
Qed.

Lemma uncont_kseq :
  forall p1 p2 k,
    uncont p1 (Kseq p2 k) = uncont (Sseq p1 p2) k.
Proof.
  auto.
Qed.

Lemma uncont_kcall :
  forall p1 ret x s k,
    uncont p1 (Kcall ret x s k) =
      uncont (Sincall p1 ret x s) k.
```



```
Proof.
  auto.
Qed.

Lemma step_uncont :
  forall k env s1 h1 p1 s2 h2 p2,
    step env
      s1 h1 p1
      s2 h2 p2 ->
    step env
      s1 h1 (uncont p1 k)
      s2 h2 (uncont p2 k).
Proof.
  induction k; intros.
  - auto.
  - apply IHk. ee.
  - apply IHk. ee.
Qed.

(** simulation relation is equality up to uncont *)
Lemma kstep_step :
  forall env s1 h1 pA kA s2 h2 pB kB,
    kstep env
      s1 h1 pA kA
      s2 h2 pB kB ->
    step env
      s1 h1 (uncont pA kA)
      s2 h2 (uncont pB kB)
    \/ (  s1 = s2
       /\ h1 = h2
       /\ uncont pA kA = uncont pB kB
       /\ size_s pB < size_s pA)%nat.
Proof.
  intros. inv_kstep; try (
    simpl; left; apply step_uncont;
    repeat ee; fail).
  - right; simpl; firstorder.
  - right; simpl; firstorder.
Qed.

Lemma kstep_step_star_step_star :
  forall env s1 h1 pA kA s2 h2 pB kB s3 h3 p3,
    kstep env
      s1 h1 pA kA
      s2 h2 pB kB ->
    step_star env
      s2 h2 (uncont pB kB)
      s3 h3 p3 ->
    step_star env
      s1 h1 (uncont pA kA)
      s3 h3 p3.
Proof.
  intros.
  inv H; simpl in *; auto;
    repeat ee; apply step_uncont; repeat ee.
Qed.

Lemma kstep_star_step_star :
  forall env s1 h1 s2 h2 pA kA pB kB,
    kstep_star env
      s1 h1 pA kA
      s2 h2 pB kB ->
    step_star env
      s1 h1 (uncont pA kA)
      s2 h2 (uncont pB kB).
Proof.
  induction 1; intros.
  - ee.
  - find_apply_lem_hyp kstep_step.
```

```
    on (or _ _), invc. ee.
    repeat break_and; subst.
    find_rewrite; auto.
Qed.

Lemma ksteps_p_steps_p :
  forall p v,
    ksteps_p p v ->
    steps_p p v.
Proof.
  intros. on (ksteps_p _ _), invc. ee.
  find_apply_lem_hyp kstep_star_step_star; auto.
Qed.
```