

Oct 24, 16 7:17

## ImplInterpProof.v

Page 1/5

```

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 ImpSemanticsFacts.
Require Import ImpInterp.

```

```

Lemma interp_op1_eval_op1 :
  forall op v v',
    interp_op1 op v = Some v' ->
      eval_unop op v v'.

```

```

Proof.
  unfold interp_op1; intros.
  repeat break_match; subst;
  discriminate || solve_by_inversion' ee.

```

**Qed.**

```

Lemma eval_op1_interp_op1 :
  forall op v v',
    eval_unop op v v' ->
      interp_op1 op v = Some v'.

```

```

Proof.
  inversion 1; auto.

```

**Qed.**

```

Lemma interp_op2_eval_op2 :
  forall op v1 v2 v',
    interp_op2 op v1 v2 = Some v' ->
      eval_binop op v1 v2 v'.

```

```

Proof.
  unfold interp_op2; intros.
  repeat break_match; subst;
  try discriminate;
  find_inversion; ee.

```

**Qed.**

```

Lemma eval_op2_interp_op2 :
  forall op v1 v2 v',
    eval_binop op v1 v2 v' ->
      interp_op2 op v1 v2 = Some v'.

```

```

Proof.
  inversion 1; auto.
  - simpl. break_match; [congruence | auto].
  - simpl. break_match; [congruence | auto].

```

**Qed.**

```

Lemma interp_e_eval_e :
  forall s h e v,
    interp_e s h e = Some v ->
      eval_e s h e v.

```

```

Proof.
  induction e; simpl; intros.
  - inv H; ee.
  - ee.
  - repeat break_match; try discriminate.
  ee. apply interp_op1_eval_op1; auto.
  - repeat break_match; try discriminate.
  ee. apply interp_op2_eval_op2; auto.
  - repeat break_match; try discriminate.
  + find_inversion. eapply eval_len_s; eauto.

```

Oct 24, 16 7:17

## ImplInterpProof.v

Page 2/5

```

+ find_inversion. eapply eval_len_a; eauto.
- repeat break_match; try discriminate.
+ find_inversion. eapply eval_idx_s; eauto.
+ eapply eval_idx_a; eauto.

```

**Qed.**

```

Lemma eval_e_interp_e :
  forall s h e v,
    eval_e s h e v ->
      interp_e s h e = Some v.

```

```

Proof.
  induction 1; simpl; auto.
  - repeat break_match; try discriminate.
  find_inversion. apply eval_op1_interp_op1; auto.
  - repeat break_match; try discriminate.
  repeat find_inversion. apply eval_op2_interp_op2; auto.
  - break_match; try discriminate. find_inversion.
  break_match; try discriminate. find_inversion.
  reflexivity.
  - break_match; try discriminate.
  find_inversion. reflexivity.
  - break_match; try discriminate.
  find_inversion. repeat find_rewrite.
  do 2 (break_match; try omega). auto.
  - break_match; try discriminate. find_inversion.
  break_match; try discriminate. find_inversion.
  break_match; try omega.
  break_match; try discriminate.
  find_inversion; auto.

```

**Qed.**

```

Lemma interps_e_evals_e :
  forall s h es vs,
    interps_e s h es = Some vs ->
      evals_e s h es vs.

```

```

Proof.
  induction es; simpl; intros.
  - find_inversion. ee.
  - repeat break_match; try discriminate.
  find_inversion. ee.
  apply interp_e_eval_e; auto.

```

**Qed.**

```

Lemma evals_e_interps_e :
  forall s h es vs,
    evals_e s h es vs ->
      interps_e s h es = Some vs.

```

```

Proof.
  induction 1; simpl; intros; auto.
  find_apply_lem_hyp eval_e_interp_e.
  repeat find_rewrite. auto.

```

**Qed.**

```

Lemma interp_s_step :
  forall env s h p s' h' p',
    interp_s env s h p = Some (s', h', p') ->
      step env s h p s' h' p'.

```

```

Proof.
  induction p; simpl; intros.
  - discriminate.
  - repeat break_match; try discriminate.
  find_inversion. ee; apply interp_e_eval_e; auto.
  - repeat break_match; try discriminate.
  find_inversion. ee; apply interp_e_eval_e; auto.
  - repeat break_match; try discriminate.
  find_inversion. ee; apply interp_e_eval_e; auto.
  - repeat break_match; try discriminate.
  + find_inversion. ee.
  * find_copy_apply_lem_hyp locate_inv.

```

Oct 24, 16 7:17

ImplInterpProof.v

Page 3/5

```

subst; auto.
* apply interps_e_evals_e; auto.
+ find_inversion. ee.
  eapply interps_e_evals_e; eauto.
  find_apply_lem_hyp extcall_ok; auto.
- repeat break_match; try discriminate.
+ invc H; ee. apply interp_e_eval_e; auto.
+ invc H; ee. apply interp_e_eval_e; auto.
- repeat break_match; try discriminate.
+ invc H; ee. apply interp_e_eval_e; auto.
+ invc H; ee. apply interp_e_eval_e; auto.
- break_if.
+ find_inversion; ee.
+ repeat break_match; try discriminate.
  find_inversion; ee.
- break_if.
+ break_match; try discriminate.
  find_inversion; ee. apply interp_e_eval_e; auto.
+ repeat break_match; try discriminate.
  find_inversion; ee.

```

**Qed.**

(\*\* Only true for deterministic Imp subst:

```

forall env s h p s' h' p',
  step env s h p s' h' p' ->
  interp_s env s h p = Some (s', h', p')
*)

```

\*)

**Lemma** interp\_s\_nostep :

```

forall env s h p s' h' p',
  interp_s env s h p = None ->
  ~ step env s h p s' h' p'.

```

Proof.

```

unfold not; intros. prep_induction H0.
induction H0; simpl; intros; subst.
- find_apply_lem_hyp eval_e_interp_e.
  find_rewrite; discriminate.
- repeat (find_apply_lem_hyp eval_e_interp_e).
  repeat find_rewrite.
  break_if. discriminate. omega.
- repeat (find_apply_lem_hyp eval_e_interp_e).
  repeat find_rewrite.
  repeat break_if; try discriminate; try omega.
- find_apply_lem_hyp evals_e_interps_e.
  repeat find_rewrite. discriminate.
- find_apply_lem_hyp evals_e_interps_e.
  repeat find_rewrite. repeat break_match.
  + discriminate.
  + on (extcall_spec _ _ _ _), inv;
    simpl in *; discriminate.
- find_apply_lem_hyp eval_e_interp_e.
  find_rewrite; discriminate.
- find_apply_lem_hyp eval_e_interp_e.
  find_rewrite; discriminate.
- find_apply_lem_hyp eval_e_interp_e.
  find_rewrite; discriminate.
- break_if; subst. discriminate. congruence.
- break_if; subst. discriminate.
  repeat break_match; subst. discriminate. auto.
- break_if; subst.
  find_apply_lem_hyp eval_e_interp_e.
  find_rewrite; discriminate. congruence.
- break_if; subst. auto.
  repeat break_match; try discriminate. auto.

```

**Qed.**

Oct 24, 16 7:17

ImplInterpProof.v

Page 4/5

```

Inductive result_ok (env : env) :
  store -> heap -> stmt -> expr -> result -> Prop :=
| result_ok_timeout :
  forall s1 h1 p1 s2 h2 p2 ret,
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    result_ok env
      s1 h1 p1 ret
    (Timeout s2 h2 p2 ret)
| result_ok_done :
  forall s1 h1 p1 s2 h2 ret v,
    step_star env
      s1 h1 p1
      s2 h2 Snop ->
    eval_e s2 h2 ret v ->
    result_ok env
      s1 h1 p1 ret
    (Done h2 v)
| result_ok_stuck_prog :
  forall s1 h1 p1 s2 h2 p2 ret,
    step_star env
      s1 h1 p1
      s2 h2 p2 ->
    p2 <> Snop ->
    (forall s3 h3 p3,
      ~ step env
        s2 h2 p2
        s3 h3 p3) ->
    result_ok env
      s1 h1 p1 ret
    (Stuck s2 h2 p2 ret)
| result_ok_stuck_ret :
  forall s1 h1 p1 s2 h2 ret,
    step_star env
      s1 h1 p1
      s2 h2 Snop ->
    (forall v, ~ eval_e s2 h2 ret v) ->
    result_ok env
      s1 h1 p1 ret
    (Stuck s2 h2 Snop ret).

```

**Lemma** interp\_s\_interps\_p :

```

forall n env s1 h1 p1 s2 h2 p2 ret res,
  interp_s env s1 h1 p1 = Some (s2, h2, p2) ->
  interps_p n env s2 h2 p2 ret = res ->
  interps_p (S n) env s1 h1 p1 ret = res.

```

Proof.

```

simpl; intros. break_match; subst.
- discriminate.
- find_rewrite; auto.

```

**Qed.****Lemma** interps\_p\_inv :

```

forall n env s h p ret res,
  interps_p n env s h p ret = res ->
  result_ok env s h p ret res.

```

Proof.

```

induction n; simpl; intros; subst.
- repeat ee.
- repeat break_match; subst.
  + eapply result_ok_done; eauto.
  repeat ee. eapply interp_e_eval_e; eauto.
  + eapply result_ok_stuck_ret; eauto.
  repeat ee. unfold not; intros.
  find_apply_lem_hyp eval_e_interp_e.
  congruence.
  + remember (interps_p n env s1 h0 s0 ret).
  symmetry in Heqr. find_copy_apply_hyp_hyp.

```

Oct 24, 16 7:17

ImplInterpProof.v

Page 5/5

```
on (result_ok _ _ _ _), inv.  
* repeat ee. eapply interp_s_step; eauto.  
* repeat ee. eapply interp_s_step; eauto.  
* repeat ee. eapply interp_s_step; eauto.  
* eapply result_ok_stuck_ret; eauto.  
  repeat ee. eapply interp_s_step; eauto.  
+ repeat ee. intros.  
  apply interp_s_nostep; auto.
```

**Qed.**

```
Lemma interp_p_steps_p :  
  forall n funcs main ret h' v,  
    interp_p n (Prog funcs main ret) = Done h' v ->  
    steps_p (Prog funcs main ret) v.
```

Proof.

```
unfold interp_p; intros.  
find_apply_lem_hyp interps_p_inv.  
on (result_ok _ _ _ _), inv.  
repeat ee.
```

**Qed.**