```
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 ImpExprTransf.
Require Import ImpInterpNock.
Require Import ImpConstFold.

Require Import ImpEval.
Require Import ImpStep.
Require Import ImpSemanticsFacts.
Require Import ImpInterpProof.
Require Import ImpInterpNockProof.
Require Import ImpExprTransfProof.

Lemma cfold_aux_fwd :
  forall s h e v,
    ImpEval.eval_e s h e v ->
    ImpEval.eval_e s h (cfold_aux e) v.
Proof.
  induction e; simpl; intros; auto.
  - break_match; auto.
    find_apply_lem_hyp eval_e_interp_e.
    find_apply_lem_hyp nock_e_ok.
    simpl in *. find_rewrite.
    ee.
  - repeat break_match; auto.
    repeat find_apply_lem_hyp eval_e_interp_e.
    repeat find_apply_lem_hyp nock_e_ok.
    simpl in *. find_rewrite.
    ee.
Qed.

Lemma cfold_aux_bwd' :
  forall s h e v,
    I.interp_e s h (cfold_aux e) = Some v ->
    I.interp_e s h e = Some v \/
    (forall v', ~ ImpEval.eval_e s h e v').
Proof.
  induction e; simpl; intros; auto.
  - break_match_hyp; simpl; auto.
    find_apply_lem_hyp nock_e_ok.
    simpl in *.
    destruct o; destruct v0;
      simpl in *; subst; auto;
      right; unfold not in *; intros;
      repeat on (eval_e _ _ _ _), invc;
      on (eval_unop _ _ _), inv.
  - repeat break_match_hyp; simpl; auto.
    repeat find_apply_lem_hyp nock_e_ok.
    simpl in *.
    destruct o; destruct v0; destruct v1;
    simpl in *; subst; auto; try (
      right; unfold not in *; intros;
      repeat on (eval_e _ _ _ _), invc;
      on (eval_binop _ _ _ _), inv;
      fail).
    + break_match; subst; auto.
      right; unfold not in *; intros.
      repeat on (eval_e _ _ _ _), invc.
      on (eval_binop _ _ _ _), inv.
      congruence.
```



```
    + break_match; subst; auto.
      right; unfold not in *; intros.
      repeat on (eval_e _ _ _ _), invc.
      on (eval_binop _ _ _ _), inv.
      congruence.
Qed.

Lemma cfold_aux_bwd :
  forall s h e v,
    eval_e s h (cfold_aux e) v ->
    eval_e s h e v \/
    (forall v', ~ eval_e s h e v').
Proof.
  intros.
  find_apply_lem_hyp eval_e_interp_e.
  find_apply_lem_hyp cfold_aux_bwd'.
  on (or _ _), invc.
  + left. apply interp_e_eval_e; auto.
  + right; auto.
Qed.

Lemma cfold_p_fwd :
  forall p v,
    steps_p p v ->
    steps_p (cfold p) v.
Proof.
  apply transf_p_fwd.
  apply transf_e_fwd.
  apply cfold_aux_fwd.
Qed.

Lemma cfold_p_bwd :
  forall p v,
    steps_p (cfold p) v ->
    steps_p p v \/ can_get_stuck_prog p.
Proof.
  apply transf_p_bwd.
  apply transf_e_bwd.
  apply cfold_aux_fwd.
  apply cfold_aux_bwd.
Qed.
```