```
(** ** Termination for IMP programs *)

Require Import Bool.
Require Import ZArith.
Require Import IMPSyntax.
Require Import IMPSemantics.

Ltac ecea := econstructor; eauto.
Ltac erep := repeat ecea.

Lemma can_eval:
  forall h e,
  exists i, eval h e i.
Proof.
  intros; induction e.
  - erep.
  - erep.
  - destruct IHe1; destruct IHe2.
    (** take apart IH before eauto *)
    erep.
Qed.

Lemma can_step:
  forall s,
    s <> Nop ->
    forall h, exists h', exists s', step h s h' s'.
Proof.
  intros; induction s.
  - congruence.
  - destruct (can_eval h e). erep.
  - destruct (isNop s1) eqn:?.
    + apply isNop_ok in Heqb; subst. erep.
    + destruct IHs1.
      * intro; subst. discriminate.
      * destruct H0; erep.
  - destruct (can_eval h e).
    destruct (Z_eq_dec 0 x).
    + ecea; ecea; eapply step_cond_false; eauto.
    + erep.
  - destruct (can_eval h e).
    destruct (Z_eq_dec 0 x).
    + ecea; ecea; eapply step_while_false; eauto.
    + erep.
Qed.

Definition diverges (s: stmt) : Prop :=
  forall h n,
  exists h', exists s',
    step_n h s n h' s'.

Definition furnace : stmt :=
  while 1 {{ Nop }}.

(** stupid auto indent *)
Ltac zex x := exists x.

Lemma warming_up:
  diverges furnace.
Proof.
  unfold diverges. intros.
  induction n.
  - zex h. zex furnace. constructor.
  - destruct IHn as [h' [s' H]].
    (** hmm, need to add next step to the *end* *)
Abort.

Lemma step_n_r:
  forall h1 s1 n h2 s2 h3 s3,
    step_n h1 s1 n h2 s2 ->
```



```
    step h2 s2 h3 s3 ->
    step_n h1 s1 (S n) h3 s3.
Proof.
  intros. induction H.
  - econstructor; eauto.
    constructor.
  - econstructor; eauto.
Qed.

Lemma warming_up:
  diverges furnace.
Proof.
  unfold diverges. intros.
  induction n.
  - zex h. zex furnace. constructor.
  - destruct IHn as [h' [s' H]].
    (** just need to show that s' can take one more step *)
    (** we know everything but Nop can step... *)
    (** hmmm, do not know much about h' s' *)
    (** need stronger IH ! *)
Abort.

Lemma warming_up:
  forall h n,
  exists h', exists s',
    step_n h furnace n h' s' /\
    s' <> Nop.
Proof.
  intros. induction n.
  - zex h. zex furnace.
    split.
    + constructor.
    + discriminate.
  - destruct IHn as [h' [s' [Hs Hn]]].
    destruct (can_step s' Hn h') as [h'' [s'' HS]].
    zex h''; zex s''. split.
    + eapply step_n_r; eauto.
    + (** ugh, don't know enough about s'' ! *)
      (** IH still too weak *)
Abort.

Lemma warming_up:
  forall h n,
  exists h',
    step_n h furnace n h' furnace.
Proof.
  intros. induction n.
  - zex h. constructor.
  - destruct IHn as [h' IH].
    eexists. eapply step_n_r; eauto.
    (** stuck! furnance doesn't step to itself! *)
    (** IH too strong!!! *)
Abort.

Definition furnace' : stmt :=
  Nop ;; while 1 {{ Nop }}.

Lemma warming_up:
  forall h n,
  exists h',
    step_n h furnace n h' furnace \/
    step_n h furnace n h' furnace'.
Proof.
  intros. induction n.
  - zex h. left. constructor.
  - destruct IHn as [h' [IH | IH]].
    + eexists. right.
      eapply step_n_r; eauto.
      unfold furnace'.
```

```
      econstructor; eauto.
      econstructor; eauto.
      omega.
    + eexists. left.
      eapply step_n_r; eauto.
      unfold furnace'.
      econstructor; eauto.
Qed.

(**
  Finally IH is strong enough!
  Now prove our original goal as a special case.
*)
Lemma full_blast:
  diverges furnace.
Proof.
  unfold diverges; intros.
  pose proof (warming_up h n).
  destruct H as [h' [H1 | H2]].
  - erep.
  - erep.
Qed.
```