```
(** * Lecture 07 *)

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

(** ** Pseudo Denotational Semantics *)

Check exec_op.
Check Zplus.

Definition denote_binop (op: binop) : Z -> Z -> Z :=
  exec_op op.

Fixpoint denote_expr (e: expr) : heap -> Z :=
  match e with
    | Int i =>
      fun _ => i
    | Var v =>
      fun h => h v
    | BinOp op e1 e2 =>
      let f := denote_binop op in
      let x := denote_expr e1 in
      let y := denote_expr e2 in
      fun h =>
        f (x h) (y h)
      (**
        fun h => denote_binop op
                   (denote_expr e1 h)
                   (denote_expr e2 h)
      *)
  end.

Eval cbv in (denote_expr ("x" [+] "y")).
Eval cbv in ((denote_expr ("x" [+] "y")) empty).

Eval cbv in (denote_expr ("x" [+] 1)).
Eval cbv in ((denote_expr ("x" [+] 1)) empty).

Lemma denote_expr_interp_expr:
  forall e h,
    (denote_expr e) h = interp_expr h e.
Proof.
  induction e; simpl; intros; auto.
  unfold denote_binop. congruence.
Qed.

(** already connected interp_expr to eval,
    so now get denote connections "for free" *)
Lemma denote_expr_eval:
  forall e h i,
    (denote_expr e) h = i <-> eval h e i.
Proof.
  intros. rewrite denote_expr_interp_expr.
  split.
  - apply interp_expr_eval.
  - apply eval_interp_expr.
Qed.

Definition obind {A B: Type} (oa: option A) (f: A -> option B) : option B :=
  match oa with
    | None => None
    | Some a => f a
  end.

(** careful to detect timeout (running out of fuel)! *)
Fixpoint denote_stmt (s: stmt) : nat -> heap -> option heap :=
  match s with
```

```
    | Nop =>
      fun _ h => Some h
    | Assign v e =>
      let de := denote_expr e in
      fun _ h => Some (update h v (de h))
    | Seq s1 s2 =>
      let d1 := denote_stmt s1 in
      let d2 := denote_stmt s2 in
      fun n h => obind (d1 n h) (d2 n)
    | Cond e s =>
      let de := denote_expr e in
      let ds := denote_stmt s in
      fun n h =>
        if Z_eq_dec 0 (de h) then
          Some h
        else
          ds n h
    | While e s =>
      let de := denote_expr e in
      let ds := denote_stmt s in
      fix loop n h :=
      match n with
        | O => None
        | S m =>
          if Z_eq_dec 0 (de h) then
            Some h
          else
            obind (ds n h) (loop m)
      end
  end.
```