

课后阅读：WhileD 表达式指称语义

```
Record state: Type := {  
  var: var_name -> Z; (** 表示每个变量的存储地址 *)  
  mem: Z -> mem_val; (** 既包含变量存储的内存空间也包含其他内存空间 *)  
}.
```

```
Definition deref_sem (D: state -> SetMonadE.M Z):  
  state -> SetMonadE.M Z :=  
  fun s =>  
    x <- D s;;  
    match s.(mem) x with  
    | Mem_NoPerm => abort  
    | Mem_HasPerm Var_U => abort  
    | Mem_HasPerm (Var_I n) => ret n  
  end.
```

```
Definition var_addr_sem (X: var_name):  
  state -> SetMonadE.M Z :=  
  fun s => ret (s.(var) X).
```

```
Definition var_sem (X: var_name):  
  state -> SetMonadE.M Z :=  
  fun s =>  
    y <- ret (s.(var) X);;  
    match s.(mem) y with  
    | Mem_NoPerm => abort  
    | Mem_HasPerm Var_U => abort  
    | Mem_HasPerm (Var_I n) => ret n  
  end.
```

其他语义算子可以沿用原先定义。
表达式的指称语义要分左值和右值定义。

```
Record EDenote: Type := {  
  lvalue: state -> SetMonadE.M Z;  
  rvalue: state -> SetMonadE.M Z;  
}.
```

最终的递归定义：

```

Fixpoint eval_expr (e: expr): EDenote :=
  match e with
  | EConst n =>
    { | lvalue := fun _ => abort;
      rvalue := const_sem n | }
  | EVar X =>
    { | lvalue := var_addr_sem X;
      rvalue := var_sem X | }
  | EBinop op e1 e2 =>
    { | lvalue := fun _ => abort;
      rvalue := binop_sem op
        (eval_expr e1).(rvalue)
        (eval_expr e2).(rvalue) | }
  | EUnop op e1 =>
    { | lvalue := fun _ => abort;
      rvalue := unop_sem op (eval_expr e1).(rvalue) | }
  | EDeref e1 =>
    { | lvalue := (eval_expr e1).(rvalue);
      rvalue := deref_sem (eval_expr e1).(rvalue) | }
  | EAddrOf e1 =>
    { | lvalue := fun _ => abort;
      rvalue := (eval_expr e1).(lvalue) | }
  end.

```