用单子描述计算 2

1 程序验证案例一：归并排序中的归并步骤

```
Definition body_merge:
  list Z * list Z * list Z ->
  SetMonad.M (ContinueOrBreak (list Z * list Z * list Z) (list Z)) :=
  fun '(l1, l2, l3) =>
    match l1, l2 with
    | nil, _ => break (l3 ++ l2)
    | _, nil => break (l3 ++ l1)
    | x :: l1', y :: l2' =>
        choice
          (test (x <= y);; continue (l1', l2, l3 ++ x :: nil))
          (test (y <= x);; continue (l1, l2', l3 ++ y :: nil))
  end.
```

```
Definition merge l l0 :=
  repeat_break body_merge (l, l0, nil).
```

```
Fixpoint incr_aux (l: list Z) (x: Z): Prop :=
  match l with
  | nil => True
  | y :: l0 => x <= y /\ incr_aux l0 y
  end.
```

```
Definition incr (l: list Z): Prop :=
  match l with
  | nil => True
  | x :: l0 => incr_aux l0 x
  end.
```

```
Theorem functional_correctness_merge:
  forall l1 l2,
    incr l1 ->
    incr l2 ->
    Hoare (merge l1 l2)
          (fun l3 => Permutation (l1 ++ l2) l3 /\ incr l3).
```

2 程序验证案例二：归并排序

```
Definition ext_sort (l: list Z): SetMonad.M (list Z) :=
  fun l' => Permutation l l' /\ incr l'.
```

```
Definition ext_split (l: list Z): SetMonad.M (list Z * list Z) :=
  fun '(l0, l1) => Permutation l (l0 ++ l1).
```

```
Definition gmergesort_f (W: list Z -> SetMonad.M (list Z)):
  list Z -> SetMonad.M (list Z) :=
  fun x =>
    choice
      (ext_sort x)
      (test (length x >= 2)%nat;;
       '(p1, q1) <- ext_split x ;;
       p2 <- W (p1) ;;
       q2 <- W (q1) ;;
       merge p2 q2).
```

```
Definition gmergesort := Kleene_LFix gmergesort_f.
```

```
Lemma ext_sort_fact:
  forall l,
    Hoare (ext_sort l) (fun l0 => Permutation l l0 /\ incr l0).
Proof.
  unfold Hoare, ext_sort; sets_unfold.
  intros.
  tauto.
Qed.
```

```
Lemma ext_split_fact:
  forall l,
    Hoare (ext_split l) (fun '(l1, l2) => Permutation l (l1 ++ l2)).
Proof.
  unfold Hoare, ext_split; sets_unfold.
  intros.
  tauto.
Qed.
```

```
Theorem functional_correctness_mergesort:
  forall l,
    Hoare (gmergesort l) (fun l0 => Permutation l l0 /\ incr l0).
```

```
Fixpoint list_iter
           {A B: Type}
           (f: A -> B -> SetMonad.M B)
           (l: list A)
           (b: B):
  SetMonad.M B :=
  match l with
  | nil => ret b
  | a :: l0 => b0 <- f a b;; list_iter f l0 b0
  end.
```

```
Theorem Hoare_list_iter {A B: Type}:
  forall (f: A -> B -> SetMonad.M B)
         (P: list A -> B -> Prop),
    (forall b l a,
       P l b ->
       Hoare (f a b) (fun b0 => P (l ++ a :: nil) b0)) ->
    (forall b l, P nil b -> Hoare (list_iter f l b) (fun b0 => P l b0)).
```

```
Import SetMonadExamples4.


Definition insertion (x: Z) (l: list Z): SetMonad.M (list Z) :=
  fun l' => exists l1 l2,
    l = l1 ++ l2 /\ l' = l1 ++ x :: l2 /\ incr l'.
```

```
Definition ins_sort (l: list Z) :=
  list_iter insertion l nil.
```

习题 1.

```
Theorem ins_sort_perm:
  forall L,
    Hoare
      (ins_sort L)
      (fun l => Permutation L l).
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 2.

```
Theorem ins_sort_incr:
  forall L,
    Hoare
      (ins_sort L)
      (fun l => incr l).
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

```
Theorem functional_correctness_ins_sort:
  forall L,
    Hoare
      (ins_sort L)
      (fun l => Permutation L l /\ incr l).
Proof.
  intros.
  apply Hoare_conjunct.
  + apply ins_sort_perm.
  + apply ins_sort_incr.
Qed.
```