单子程序上的霍尔逻辑

1 单子程序上的霍尔逻辑

集合单子上，霍尔三元组会退化为霍尔二元组。

```
Definition Hoare {A: Type} (c: SetMonad.M A) (P: A -> Prop): Prop :=
  forall a, a ∈ c -> P a.
```

可以证明，各个单子算子满足下面性质。

```
Theorem Hoare_bind {A B: Type}:
  forall (f: SetMonad.M A)
         (g: A -> SetMonad.M B)
         (P: A -> Prop)
         (Q: B -> Prop),
    Hoare f P ->
    (forall a, P a -> Hoare (g a) Q) ->
    Hoare (bind f g) Q.
```

```
Theorem Hoare_ret {A: Type}:
  forall (a: A) (P: A -> Prop),
    P a -> Hoare (ret a) P.
```

```
Theorem Hoare_conseq {A: Type}:
  forall (f: SetMonad.M A) (P Q: A -> Prop),
    (forall a, P a -> Q a) ->
    Hoare f P ->
    Hoare f Q.
```

```
Theorem Hoare_conjunct {A: Type}:
  forall (f: SetMonad.M A) (P Q: A -> Prop),
    Hoare f P ->
    Hoare f Q ->
    Hoare f (fun a => P a /\ Q a).
```

```
Theorem Hoare_choice {A: Type}:
  forall (f g: SetMonad.M A)
         (P: A -> Prop),
    Hoare f P ->
    Hoare g P ->
    Hoare (choice f g) P.
```

```
Theorem Hoare_assume_bind {A: Type}:
  forall (P: Prop)
         (f: SetMonad.M A)
         (Q: A -> Prop),
    (P -> Hoare f Q) ->
    (Hoare (assume P;; f) Q).
```

```
Theorem Hoare_repeat_break {A B: Type}:
  forall (body: A -> SetMonad.M (ContinueOrBreak A B))
         (P: A -> Prop)
         (Q: B -> Prop),
    (forall a, P a ->
               Hoare (body a) (fun x => match x with
                                        | by_continue a => P a
                                        | by_break b => Q b
                                        end)) ->
    (forall a, P a -> Hoare (repeat_break body a) Q).
```

2 霍尔逻辑证明

2.1 3x + 1

```
Theorem functional_correctness_3x1:
  forall n: Z,
    n >= 1 ->
    Hoare (run_3x1 n) (fun m => m = 1).
```

2.2 二分查找

```
Theorem functional_correctness_binary_search:
  forall (P: Z -> Prop) lo hi,
    (forall n m, n <= m -> P m -> P n) ->
    P lo ->
    ~ P hi ->
    Hoare (binary_search P lo hi)
          (fun x => P x /\ ~ P (x + 1)).
```

2.3 归并

```
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.
```

```
Lemma incr_app_cons: forall l1 x l2,
  incr (l1 ++ [x]) ->
  incr (x :: l2) ->
  incr (l1 ++ x :: l2).
```

```
Lemma incr_app_cons_inv1: forall l1 x l2,
  incr (l1 ++ x :: l2) ->
  incr (l1 ++ [x]).
```

```
Lemma incr_app_cons_inv2: forall l1 x l2,
  incr (l1 ++ x :: l2) ->
  incr (x :: l2).
```

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

2.4 拓展欧几里得

```
Definition any (A: Type): SetMonad.M A := Sets.full.
```

```
Definition body_ext_euclid
             (W: Z -> Z -> SetMonad.M (Z * Z))
             (a b: Z): SetMonad.M (Z * Z) :=
  choice
    (assume (b = 0);;
     choice
       (assume (a >= 0);; ret (1, 0))
       (assume (a <= 0);; ret (-1, 0)))
    (assume (b <> 0);;
     q <- any Z;;
     r <- any Z;;
     assume (a = r + q * b);;
     '(x, y) <- W b r;;
     ret (y, x - q * y)).
```

```
Definition ext_euclid (a b: Z): SetMonad.M (Z * Z) :=
  Kleene_LFix body_ext_euclid a b.
```

```
Lemma Hoare_Kleene_LFix2 {A1 A2 B: Type}:
  forall (a1: A1) (a2: A2)
         (Q: B -> Prop)
         (f: (A1 -> A2 -> SetMonad.M B) ->
             (A1 -> A2 -> SetMonad.M B)),
    (forall n, Hoare (Nat.iter n f ∅ a1 a2) Q) ->
    Hoare (Kleene_LFix f a1 a2) Q.
```

```
Theorem Hoare_any_bind {A B: Type}:
  forall (f: A -> SetMonad.M B) (Q: B -> Prop),
    (forall a, Hoare (f a) Q) ->
    Hoare (bind (any A) f) Q.
```

```
Lemma functional_correctness_ext_euclid_aux:
  forall W,
    (forall a b, Hoare
                  (W a b)
                  (fun '(x, y) => a * x + b * y = Z.gcd a b)) ->
    (forall a b, Hoare
                  (body_ext_euclid W a b)
                  (fun '(x, y) => a * x + b * y = Z.gcd a b)).
```

```
Theorem functional_correctness_ext_euclid:
  forall a b,
    Hoare (ext_euclid a b)
          (fun '(x, y) => a * x + b * y = Z.gcd a b).
```

2.5 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)
      (assume(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).
```

2.6 依次处理列表

```
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)).
```