不动点定理

1 不动点分析

函数 $F(X) \triangleq \text{test\_true}([\![e]\!]) \circ [\![c]\!] \circ X \cup \text{test\_false}([\![e]\!])$ 的不动点具有下面这些性质。

- 如果从 s_1 出发执行循环语句 while (e) do $\{c\}$ 会以 s_2 为终止状态结束运行，那么对于任意一个 F 的不动点 X，都有
  - $(s_1, s_2) \in X$
  - 不存在其他的 s 使得 $(s_1, s) \in X$
- 如果从 s_1 出发执行循环语句 while (e) do $\{c\}$ 会在某一次执行循环体的过程中在循环体内部陷入死循环，那么对于任意一个 F 的不动点 X，都有
  - 不存在 s 使得 $(s_1, s) \in X$
- 如果从 s_1 出发执行循环语句 while (e) do $\{c\}$ 时每一次执行循环体后依次经过程序状态 $s_2, s_3, \ldots$ 循行不终止，那么对于任意一个 F 的不动点 X，以及任意程序状态 s，都有
  - 要么 $(s_1, s), (s_2, s), (s_3, s), \ldots$ 全部都是 X 的元素
  - 要么 $(s_1, s), (s_2, s), (s_3, s), \ldots$ 全部都不是 X 的元素

$[\![\text{while } (e) \text{ do } \{c\}]\!]$ 可以被定义为下述函数的最小不动点：

$$F(X) \triangleq \text{test\_true}([\![e]\!]) \circ [\![c]\!] \circ X \cup \text{test\_false}([\![e]\!])$$

2 Kleene 不动点定理

下面介绍 Kleene 不动点定理。这将统一的回答为什么有不动点，如何构造最小不动点。

- 定义：偏序集。满足下面三个条件的 $(A, \leqslant_A)$ 成为一个偏序集（partial ordering）：
  - 自反性：对于任意 $a \in A$， $a \leqslant_A a$
  - 传递性：对于任意 $a, b, c \in A$，如果 $a \leqslant_A b$、$b \leqslant_A c$，那么 $a \leqslant_A c$
  - 反对称性：对于任意 $a, b \in A$，如果 $a \leqslant_A b$、$b \leqslant_A a$，那么 $a = b$
- 例子：
  - $(\mathbb{R}, \leqslant)$ 是一个偏序集。
  - 如果 D 表示自然数之间的整除关系，即 $(a, b) \in D$ 当且仅当 $a \mid b$，那么 $(\mathbb{N}, D)$ 是一个偏序集。值得一提的是，在这个偏序关系下，两个自然数之间不一定可以相互比较；例如 $2 \nmid 3$ 并且 $3 \nmid 2$。
  - 如果 X 是一个集合，$\mathcal{P}(X)$ 表示 X 的幂集，那么 $(\mathcal{P}(X), \subseteq)$ 构成一个偏序集。

- 定义：完备偏序集。如果偏序集 $(A, \leqslant_A)$ 还满足下面性质，那么它是一个完备偏序集（complete partial ordering，CPO）：
  - 完备性：对于任意 $S \subseteq A$，如果 S 中任意两个元素之间都可以大小比较，那么 S 有上确界（least upper bound, lub），记做 $\text{lub}(S)$，即：(1) 对于任意 $a \in S$，$a \leqslant_A \text{lub}(S)$；(2) 如果某个 $b \in A$ 使得每一个 $a \in S$ 都有 $a \leqslant_A b$，那么 $\text{lub}(S) \leqslant_A b$。
  - 注：符合上述性质的 S 称为偏序集 A 上的一条链。
- 例子：
  - $(\mathbb{R}, \leqslant)$ 是偏序集但是不是完备偏序集，因为 $\mathbb{Z} \subseteq \mathbb{R}$ 是一条链，但是它没有上确界。
  - 如果 X 是一个集合，$\mathcal{P}(X)$ 表示 X 的幂集，那么 $(\mathcal{P}(X), \subseteq)$ 是一个完备偏序集。其中，对于任意 $U \subseteq \mathcal{P}(X)$，都有 $\text{lub}(U) = \bigcup_{V \in U} V$ 是 U 的上确界。
  - 如果 D 表示自然数之间的整除关系，即 $(a, b) \in D$ 当且仅当 $a \mid b$，那么 $(\mathbb{N}, D)$ 是一个完备偏序集。特别的，如果 $U \subseteq \mathbb{N}$ 是一条链还是一个有穷集，那么 $\text{lub}(U)$ 就是 U 的最小公倍数；如果 $U \subseteq \mathbb{N}$ 是一条链还是一个无穷集，那么 $\text{lub}(U)$ 就是 0。
  - 如果 D^+ 表示正整数之间的整除关系，那么 $(\mathbb{Z}^+, D^+)$ 是一个偏序集，但不是一个完备偏序集。例如，$\{1, 2, 4, 8, ..., 2^n, ...\}$ 是整除关系上的一条链，但是它没有整除关系意义下的上确界。
- 定义：单调函数。如果 $(A, \leqslant_A)$ 是一个偏序集，那么 $F : A \to A$ 是一个单调函数当且仅当：对于任意 $a, b \in A$，如果 $a \leqslant_A b$，那么 $F(a) \leqslant_A F(b)$。
- 引理：如果 S 是偏序集 $(A, \leqslant_A)$ 上的一条链，$F : A \to A$ 是一个单调函数，那么 $F(S) \triangleq \{F(a) \mid a \in S\}$ 也是一条链。
- 证明：任给 $a, b \in S$，要么 $a \leqslant_A b$，要么 $b \leqslant_A a$。若前者成立，那么 $F(a) \leqslant_A F(b)$；若后者成立，那么 $F(b) \leqslant_A F(a)$，因此 $F(S)$ 中的元素间两两可以比较大小。
- 定义：单调连续函数。如果 $(A, \leqslant_A)$ 是一个完备偏序集，那么单调函数 $F : A \to A$ 是连续的当且仅当：对于任意一条非空链 S，$F(\text{lub}(S)) = \text{lub}(F(S))$。
- 引理：如果 $(A, \leqslant_A)$ 是一个完备偏序集，那么这个集合上有最小元，记做 $\perp$。
- 证明：空集 $\varnothing$ 是 $(A, \leqslant_A)$ 上的一条链。而任何一个 A 中元素，都是空集的上界。因此，对于任意 $a \in A$ 都有，$\text{lub}(\varnothing) \leqslant_A a$。
- 引理：如果 F 是完备偏序集 $(A, \leqslant_A)$ 上的单调连续函数，那么 $\{\perp, F(\perp), F(F(\perp)), ...\}$ 是 $(A, \leqslant_A)$ 上的一条链。
- 证明：

  由于 $\perp$ 是最小元，所以 $\perp \leqslant_A F(\perp)$。由于 F 是单调函数，所以 $F(\perp) \leqslant_A F(F(\perp))$。依次类推，$\perp \leqslant_A F(\perp) \leqslant_A F(F(\perp)) \leqslant_A ...$
- 定理：如果 F 是完备偏序集 $(A, \leqslant_A)$ 上的单调连续函数，$\text{lub}(\perp, F(\perp), F(F(\perp)), ...)$ 是 F 的一个不动点。
- 证明：

$$
\begin{aligned}
& F(\text{lub}(\perp, F(\perp), F(F(\perp)), ...)) \\
= \quad & \text{lub}(F(\perp), F(F(\perp)), F(F(F(\perp))), ...) \\
= \quad & \text{lub}(\perp, F(\perp), F(F(\perp)), F(F(F(\perp))), ...)
\end{aligned}
$$

- 定理:如果 F 是完备偏序集 $(A, \leqslant_A)$ 上的单调连续函数且 $F(a) = a$,那么 $\mathrm{lub}(\perp, F(\perp), F(F(\perp)), \ldots) \leqslant_A a$。
- 证明:

$$
\begin{aligned}
&\perp \leqslant_A a \\
&F(\perp) \leqslant_A F(a) = a \\
&F(F(\perp)) \leqslant_A F(a) = a \\
&\ldots
\end{aligned}
$$

用 Kleene 不动点定义 while 语句语义

- $(\mathcal{P}(\text{state} \times \text{state}), \subseteq)$ 是一个完备偏序集;
- $F(X) \triangleq \text{test\_true}(\llbracket e \rrbracket) \circ \llbracket c \rrbracket \circ X \cup \text{test\_false}(\llbracket e \rrbracket)$ 是一个单调连续函数;
  - $G(X) = Y \circ X$ 是单调连续函数;
  - $H(X) = X \cup Y$ 是单调连续函数;
  - 如果 $G(X)$ 与 $H(X)$ 都是单调连续函数,那么 $G(H(X))$ 也是单调连续函数;
- F 的最小不动点是:

$$
\bigcup_{n \in \mathbb{N}} (F^{(n)}(\emptyset))
$$

两种定义的对应关系 $F^{(n)}(\emptyset) = \text{boundedLB}(\llbracket e \rrbracket, \llbracket c \rrbracket)$。

3 While 语言的指称语义

下面定义程序语句的语义。程序语句的语义包含两种情况:正常运行终止和运行出错。

```
Record CDenote: Type := {
  nrm: state -> state -> Prop;
  err: state -> Prop
}.
```

空语句的语义:

```
Definition skip_sem: CDenote :=
  {|
    nrm := Rels.id;
    err := ∅;
  |}.
```

赋值语句的语义:

```
Definition asgn_sem
             (X: var_name)
             (D: state -> SetMonadE.M Z): CDenote :=
  {|
    nrm := fun s1 s2 =>
      exists i,
        i ∈ (D s1).(nrm) /\ s2 X = Var_I i /\
        (forall Y, X <> Y -> s2 Y = s1 Y);
    err := fun s1 => (D s1).(err);
  |}.
```

顺序执行语句的语义：

- $[\![c_1; c_2]\!].(\mathrm{nrm}) = [\![c_1]\!].(\mathrm{nrm}) \circ [\![c_2]\!].(\mathrm{nrm})$
- $c_1; c_2$ 程序出错的情况有两种：c_1 出错，或 c_1 运行终止后 c_2 出错；

  $[\![c_1; c_2]\!].(\mathrm{err}) = [\![c_1]\!].(\mathrm{err}) \cup [\![c_1]\!].(\mathrm{nrm}) \circ [\![c_2]\!].(\mathrm{err})$

```
Definition seq_sem (D1 D2: CDenote): CDenote :=
  {|
    nrm := D1.(nrm) ∘ D2.(nrm);
    err := D1.(err) ∪ (D1.(nrm) ∘ D2.(err));
  |}.
```

条件分支语句的语义：

```
Definition test_true (D: state -> SetMonadE.M Z):
  state -> state -> Prop :=
  Rels.test
    (fun s =>
       exists i, i ∈ (D s).(nrm) /\ i <> 0).

Definition test_false (D: state -> SetMonadE.M Z):
  state -> state -> Prop :=
  Rels.test (fun s => 0 ∈ (D s).(nrm)).

Definition err_set (D: state -> SetMonadE.M Z):
  state -> Prop :=
  fun s => (D s).(err).

Definition if_sem
             (D0: state -> SetMonadE.M Z)
             (D1 D2: CDenote): CDenote :=
  {|
    nrm := (test_true D0 ∘ D1.(nrm)) ∪
           (test_false D0 ∘ D2.(nrm));
    err := err_set D0 ∪
           (test_true D0 ∘ D1.(err)) ∪
           (test_false D0 ∘ D2.(err))
  |}.

Definition while_sem
             (D0: state -> SetMonadE.M Z)
             (D1: CDenote): CDenote :=
  {|
    nrm := Kleene_LFix
             (fun X =>
                test_true D0 ∘ D1.(nrm) ∘ X ∪
                test_false D0);
    err := Kleene_LFix
             (fun X =>
                test_true D0 ∘ D1.(nrm) ∘ X ∪
                test_true D0 ∘ D1.(err) ∪
                err_set D0);
  |}.
```

程序语句的语义可以最后表示成下面递归函数。

```
Fixpoint eval_com (c: com): CDenote :=
  match c with
  | CSkip =>
      skip_sem
  | CAsgn X e =>
      asgn_sem X (eval_expr e)
  | CSeq c1 c2 =>
      seq_sem (eval_com c1) (eval_com c2)
  | CIf e c1 c2 =>
      if_sem (eval_expr e) (eval_com c1) (eval_com c2)
  | CWhile e c1 =>
      while_sem (eval_expr e) (eval_com c1)
  end.
```

4 加入 break、continue 语句的指称语义

考虑以下 While 程序语言的拓展

```
Inductive com : Type :=
  | CSkip: com
  | CAsgn (X: var_name) (e: expr): com
  | CSeq (c1 c2: com): com
  | CIf (e: expr) (c1 c2: com): com
  | CWhile (e: expr) (c: com): com
  | CContinue: com
  | CBreak: com.


Record CDenote: Type := {
  nrm: state -> state -> Prop;
  brk: state -> state -> Prop;
  cnt: state -> state -> Prop;
  err: state -> Prop
}.
```

空语句的语义：

```
Definition skip_sem: CDenote :=
  {|
    nrm := Rels.id;
    brk := ∅;
    cnt := ∅;
    err := ∅;
  |}.
```

Break 语句的语义

```
Definition brk_sem: CDenote :=
  {|
    nrm := ∅;
    brk := Rels.id;
    cnt := ∅;
    err := ∅;
  |}.
```

Continue 语句的语义

```
Definition cnt_sem: CDenote :=
  {|
    nrm := ∅;
    brk := ∅;
    cnt := Rels.id;
    err := ∅;
  |}.
```

顺序执行语句的语义

```
Definition seq_sem (D1 D2: CDenote): CDenote :=
  {|
    nrm := D1.(nrm) ∘ D2.(nrm);
    brk := D1.(brk) ∪ (D1.(nrm) ∘ D2.(brk));
    cnt := D1.(cnt) ∪ (D1.(nrm) ∘ D2.(cnt));
    err := D1.(err) ∪ (D1.(nrm) ∘ D2.(err));
  |}.
```

赋值语句的语义：

```
Definition asgn_sem
             (X: var_name)
             (D: state -> SetMonadE.M Z): CDenote :=
  {|
    nrm := fun s1 s2 =>
      exists i,
        i ∈ (D s1).(nrm) /\ s2 X = Var_I i /\
        (forall Y, X <> Y -> s2 Y = s1 Y);
    brk := ∅;
    cnt := ∅;
    err := fun s1 => (D s1).(err);
  |}.
```

以下定义沿用

```
Definition test_true (D: state -> SetMonadE.M Z):
  state -> state -> Prop :=
  Rels.test
    (fun s =>
       exists i, i ∈ (D s).(nrm) /\ i <> 0).

Definition test_false (D: state -> SetMonadE.M Z):
  state -> state -> Prop :=
  Rels.test (fun s => 0 ∈ (D s).(nrm)).

Definition err_set (D: state -> SetMonadE.M Z):
  state -> Prop :=
  fun s => (D s).(err).
```

If 语句的语义

```
Definition if_sem
             (D0: state -> SetMonadE.M Z)
             (D1 D2: CDenote): CDenote :=
  {|
    nrm := (test_true D0 ∘ D1.(nrm)) ∪
           (test_false D0 ∘ D2.(nrm));
    brk := (test_true D0 ∘ D1.(brk)) ∪
           (test_false D0 ∘ D2.(brk));
    cnt := (test_true D0 ∘ D1.(cnt)) ∪
           (test_false D0 ∘ D2.(cnt));
    err := err_set D0 ∪
           (test_true D0 ∘ D1.(err)) ∪
           (test_false D0 ∘ D2.(err));
  |}.

Definition while_sem
             (D0: state -> SetMonadE.M Z)
             (D1: CDenote): CDenote :=
  {|
    nrm := Kleene_LFix
             (fun X =>
               test_true D0 ∘ D1.(nrm) ∘ X ∪
               test_true D0 ∘ D1.(cnt) ∘ X ∪
               test_true D0 ∘ D1.(brk) ∪
               test_false D0);
    brk := ∅;
    cnt := ∅;
    err := Kleene_LFix
             (fun X =>
               test_true D0 ∘ D1.(nrm) ∘ X ∪
               test_true D0 ∘ D1.(cnt) ∘ X ∪
               test_true D0 ∘ D1.(err) ∪
               err_set D0);
  |}.
```

程序语句的语义可以最后表示成下面递归函数。

```
Fixpoint eval_com (c: com): CDenote :=
  match c with
  | CSkip =>
      skip_sem
  | CAsgn X e =>
      asgn_sem X (eval_expr e)
  | CSeq c1 c2 =>
      seq_sem (eval_com c1) (eval_com c2)
  | CIf e c1 c2 =>
      if_sem (eval_expr e) (eval_com c1) (eval_com c2)
  | CWhile e c1 =>
      while_sem (eval_expr e) (eval_com c1)
  | CBreak =>
      brk_sem
  | CContinue =>
      cnt_sem
  end.
```

5 SetMonad 中加入循环

如果要用单子表示带循环的计算过程，那就需要引入新的循环算子。

首先定义循环体的计算结果，其结果要么是 continue 终止，要么是 break 终止。

```
Inductive ContinueOrBreak (A B: Type): Type :=
| by_continue (a: A)
| by_break (b: B).
```

下面用不动点定义 repeat 循环。

```
Definition repeat_break_f
             {A B: Type}
             (body: A -> SetMonad.M (ContinueOrBreak A B))
             (W: A -> SetMonad.M B)
             (a: A): SetMonad.M B :=
  x <- body a;;
  match x with
  | by_continue a' => W a'
  | by_break b => ret b
  end.
```

```
Definition repeat_break
             {A B: Type}
             (body: A -> SetMonad.M (ContinueOrBreak A B)):
  A -> SetMonad.M B :=
  Kleene_LFix (repeat_break_f body).
```

下面还可以定义循环体中的 continue 语句和 break 语句。

```
Definition continue {A B: Type} (a: A):
  SetMonad.M (ContinueOrBreak A B) :=
  ret (by_continue a).
```

```
Definition break {A B: Type} (b: B):
  SetMonad.M (ContinueOrBreak A B) :=
  ret (by_break b).
```

```
Definition body_3x1 (x: Z): SetMonad.M (ContinueOrBreak Z Z) :=
  choice
    (assume (x <= 1);; break x)
    (choice
      (assume (exists k, x = 2 * k);;
       continue (x / 2))
      (assume (exists k, k <> 0 /\ x = 2 * k + 1);;
       continue (3 * x + 1))).
```

```
Definition run_3x1: Z -> SetMonad.M Z :=
  repeat_break body_3x1.
```

```
Definition body_binary_search (P: Z -> Prop):
  Z * Z -> SetMonad.M (ContinueOrBreak (Z * Z) Z) :=
  fun '(lo, hi) =>
  choice
    (assume (lo + 1 = hi);; break lo)
    (assume (lo + 1 < hi);;
     let mid := (lo + hi) / 2 in
     choice
       (assume (P mid);; continue (mid, hi))
       (assume (~ P mid);; continue (lo, mid))).
```

```
Definition binary_search (P: Z -> Prop) (lo hi: Z):
  SetMonad.M Z :=
  repeat_break (body_binary_search P) (lo, hi).
```

```
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
          (assume (x <= y);; continue (l1', l2, l3 ++ x :: nil))
          (assume (y <= x);; continue (l1, l2', l3 ++ y :: nil))
  end.
```

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