



软件理论基础与实践

Hoare2: Hoare Logic, Part II

胡振江 熊英飞
北京大学



装饰程序 Decorated Program

- 霍尔逻辑证明程序性质的过程基本和程序结构一致
- 可以用一种更紧凑的方式表达证明过程

```
    {{ X <= 3 }}  
while X <= 2 do  
    X := X + 1  
end  
    {{ X = 3 }}
```

装饰程序 Decorated Program



```
    {{ X <= 3 }}  
while X <= 2 do  
    {{ X <= 3 /\ X <= 2 }} ->>  
    {{ X + 1 <= 3 }}  
    X := X + 1  
    {{ X <= 3 }}  
end  
    {{ X <= 3 /\ ~(X <= 2) }} ->>  
    {{ X = 3 }}
```



装饰程序与霍尔逻辑规则

```
{{ P }} skip {{ P }}
```

```
{{ P }} c1; {{ Q }} c2 {{ R }}
```

```
{{ P [X ↦ a] }}  
X := a  
{{ P }}
```

```
{{ P }}  
while b do  
  {{ P ∧ b }}  
  c1  
  {{ P }}  
end  
{{ P ∧ ¬b }}
```

```
{{ P }} ->> {{ P' }}
```

```
{{ P }}  
if b then  
  {{ P ∧ b }}  
  c1  
  {{ Q }}  
else  
  {{ P ∧ ¬b }}  
  c2  
  {{ Q }}  
end  
{{ Q }}
```

只需要检查每一部分的正确性就得到了整体证明的正确性



程序证明过程： 顺序

(1) $\{ \{ X = m \wedge Y = n \} \}$

\rightarrow

(2) $\{ \{ (X + Y) - ((X + Y) - Y) = n \wedge (X + Y) - Y = m \} \}$

$X := X + Y;$

(3) $\{ \{ X - (X - Y) = n \wedge X - Y = m \} \}$

$Y := X - Y;$

(4) $\{ \{ X - Y = n \wedge Y = m \} \}$

$X := X - Y$

(5) $\{ \{ X = n \wedge Y = m \} \}$



程序证明过程：选择

```
(1)  {{True}}  
    if  $X \leq Y$  then  
(2)  {{True  $\wedge X \leq Y$ }}  
    ->>  
(3)  {{(Y - X) + X = Y  $\vee$  (Y - X) + Y = X}}  
      Z := Y - X  
(4)  {{Z + X = Y  $\vee$  Z + Y = X}}  
      else  
(5)  {{True  $\wedge \sim(X \leq Y)$  }}  
      ->>  
(6)  {{(X - Y) + X = Y  $\vee$  (X - Y) + Y = X}}  
      Z := X - Y  
(7)  {{Z + X = Y  $\vee$  Z + Y = X}}  
      end  
(8)  {{Z + X = Y  $\vee$  Z + Y = X}}
```



程序证明过程：循环

```
(1) {{ True }}  
    while  $\sim(X = 0)$  do  
(2) {{ True  $\wedge$   $X \neq 0$  }}  
    ->>  
(3) {{ True }}  
        X := X - 1  
(4) {{ True }}  
    end  
(5) {{ True  $\wedge$   $\sim(X \neq 0)$  }}  
    ->>  
(6) {{ X = 0 }}
```



从后条件获取循环不变式

(1) $\{\{ \text{True} \}\}$

\rightarrow

(2) $\{\{ n \times 0 + m = m \}\}$

$X := m;$

(3) $\{\{ n \times 0 + X = m \}\}$

$Y := 0;$

(4) $\{\{ n \times Y + X = m \}\}$

while $n \leq X$ do

(5) $\{\{ n \times Y + X = m \wedge n \leq X \}\}$

\rightarrow

(6) $\{\{ n \times (Y + 1) + (X - n) = m \}\}$

$X := X - n;$

(7) $\{\{ n \times (Y + 1) + X = m \}\}$

$Y := Y + 1$

(8) $\{\{ n \times Y + X = m \}\}$

end

(9) $\{\{ n \times Y + X = m \wedge \neg(n \leq X) \}\}$

\rightarrow

(10) $\{\{ n \times Y + X = m \wedge X < n \}\}$



循环不变式的条件

- 足够弱：能被前条件推出
- 足够强：能推出后条件
- 能保持：每一次循环都保持条件



根据终止条件泛化

```
    {{ X = m ∧ Y = n }}  
while ~(X = 0) do  
    Y := Y - 1;  
    X := X - 1  
end  
    {{ Y = n - m }}
```

- True作为循环不变式
 - 太弱，推不出后条件
- 后条件作为循环不变式
 - 太强，前条件推不出来，且循环也不保持
- 寻找一个条件，在 $X=0$ 的时候等价于后条件
 - $Y-X=n-m$



根据终止条件泛化

```
(1) {{ X = m ∧ Y = n }} ->> (a - OK)
(2) {{ Y - X = n - m }}
    while ~(X = 0) do
(3) {{ Y - X = n - m ∧ X ≠ 0 }} ->> (c - OK)
(4) {{ (Y - 1) - (X - 1) = n - m }}
    Y := Y - 1;
(5) {{ Y - (X - 1) = n - m }}
    X := X - 1
(6) {{ Y - X = n - m }}
    end
(7) {{ Y - X = n - m ∧ ~(X ≠ 0) }} ->> (b - OK)
(8) {{ Y = n - m }}
```



练习： 为下面的证明找到循环不变式

```
{ { X=m } }  
Z := 0;  
while (Z+1) * (Z+1) ≤ X do  
    Z := Z+1  
end  
{ { Z × Z ≤ m ∧ m < (Z+1) * (Z+1) } }
```



答案： 结合前后条件

```
{ { X=m } } ->>
{ { X=m  $\wedge$  0*0  $\leq$  m } }
Z := 0;
{ { X=m  $\wedge$  Z $\times$ Z  $\leq$  m } }
while (Z+1)*(Z+1)  $\leq$  X do
    { { X=m  $\wedge$  Z $\times$ Z $\leq$ m  $\wedge$  (Z+1)*(Z+1)  $\leq$ X } } ->>
    { { X=m  $\wedge$  (Z+1)*(Z+1)  $\leq$ m } }
    Z := Z + 1
    { { X=m  $\wedge$  Z $\times$ Z $\leq$ m } }
end
{ { X=m  $\wedge$  Z $\times$ Z $\leq$ m  $\wedge$   $\sim$ ((Z+1)*(Z+1)  $\leq$ X) } } ->>
{ { Z $\times$ Z $\leq$ m  $\wedge$  m < (Z+1)*(Z+1) } }
```



练习： 为下面的证明找到循环不变式

```
    { { X = m } }  
Y  := 0;  
Z  := 0;  
while ~ (Y = X) do  
    Z := Z + X;  
    Y := Y + 1  
end  
    { { Z = m × m } }
```



答案： 结合以上两种方法

```
{ { X = m } } ->> (a - OK)
{ { 0 = 0*m ∧ X = m } }
Y := 0;
{ { 0 = Y×m ∧ X = m } }
Z := 0;
{ { Z = Y×m ∧ X = m } }
while ~(Y = X) do
    { { Z = Y×m ∧ X = m ∧ Y ≠ X } } ->> (c - OK)
    { { Z+X = (Y+1)*m ∧ X = m } }
    Z := Z + X;
    { { Z = (Y+1)*m ∧ X = m } }
    Y := Y + 1
    { { Z = Y×m ∧ X = m } }
end
{ { Z = Y×m ∧ X = m ∧ ~(Y ≠ X) } } ->> (b - OK)
{ { Z = m×m } }
```



转换装饰程序到Coq证明

```
(1) {{ True }}  
    while ~(X = 0) do  
(2) {{ True ^ X ≠ 0 }}  
    ->>  
(3) {{ True }}  
    X := X - 1  
(4) {{ True }}  
    end  
(5) {{ True ^ ~(X ≠ 0) }}  
    ->>  
(6) {{ X = 0 }}
```




转换装饰程序到Coq证明

Theorem `reduce_to_zero_correct''` :

```
{{True}}  
reduce_to_zero'  
{{X = 0}}.
```

Proof.

```
unfold reduce_to_zero'.  
eapply hoare_consequence_post.  
- apply hoare_while.  
  + eapply hoare_consequence_pre.  
    * apply hoare_asgn.  
    * assn_auto''.  
- (* fun st => True st /\ ~ (<{ ~X=0}> st)) ->> X = 0*)  
  assn_auto''. (* doesn't succeed *)
```

Abort.



新的自动证明策略

```
Ltac verify_assn :=
  repeat split;
  simpl; unfold assert_implies;
  unfold ap in *; unfold ap2 in *;
  unfold bassn in *; unfold beval in *; unfold aeval in *;
  unfold assn_sub; intros;
  repeat (simpl in *;
    rewrite t_update_eq ||
    (try rewrite t_update_neq;
      [| (intro X; inversion X; fail)]));
  simpl in *;
  repeat match goal with [H : _ /\ _ |- _] => destruct H end;
  repeat rewrite not_true_iff_false in *;
  .....
```

无需了解细节，但对于大多数assign变换后的条件蕴含证明都可用



转换装饰程序到Coq证明

```
Theorem reduce_to_zero_correct''' :  
  {{True}}  
  reduce_to_zero'  
  {{X = 0}}.  
Proof.  
  unfold reduce_to_zero'.  
  eapply hoare_consequence_post.  
  - apply hoare_while.  
    + eapply hoare_consequence_pre.  
      * apply hoare_asgn.  
      * verify_assn.  
  - verify_assn.  
Qed.
```



能否自动化上述过程

- 定义装饰程序的语法，使得在Coq中可以直接书写
- 定义函数将装饰程序转化为命题
- 定义策略自动证明命题



装饰程序语法

```
Inductive dcom : Type :=
| DCSkip (Q : Assertion)
  (* skip {{ Q }} *)
| DCSeq (d1 d2 : dcom)
  (* d1 ;; d2 *)
| DCAsgn (X : string) (a : aexp) (Q : Assertion)
  (* X := a {{ Q }} *)
| DCIf (b : bexp) (P1 : Assertion) (d1 : dcom)
  (P2 : Assertion) (d2 : dcom) (Q : Assertion)
  (* if b then {{ P1 }} d1 else {{ P2 }} d2 end {{ Q }} *)
| DCWhile (b : bexp) (P : Assertion) (d : dcom) (Q : Assertion)
  (* while b do {{ P }} d end {{ Q }} *)
| DCPre (P : Assertion) (d : dcom)
  (* ->> {{ P }} d *)
| DCPost (d : dcom) (Q : Assertion)
  (* d ->> {{ Q }} *).
Inductive decorated : Type :=
| Decorated : Assertion -> dcom -> decorated.
```

避免重复，每种dcom默认只包括后条件，由decorated提供整个程序的前条件。



装饰程序语法

Declare **Scope** dcom_scope.

Notation "'skip' {{ P }}"

:= (DCSkip P)

(**in** custom com at level 0, P constr) : dcom_scope.

Notation "'while' b 'do' {{ Pbody }} d 'end' {{ Ppost }}"

:= (DCWhile b Pbody d Ppost)

(**in** custom com at level 89, b custom com at level 99,
Pbody constr, Ppost constr) : dcom_scope.

Notation "'if' b 'then' {{ P }} d 'else' {{ P' }} d 'end' {{ Q }}"

:= (DCIf b P d P' d Q)

(**in** custom com at level 89, b custom com at level 99,
P constr, P' constr, Q constr) : dcom_scope.

.....



装饰程序书写实例

```
Example dec_while : decorated :=
  <{
    {{ True }}
    while ~(X = 0)
    do
      {{ True /\ (X <> 0) }}
      X := X - 1
      {{ True }}
    end
    {{ True /\ X = 0 }} ->>
    {{ X = 0 }} }>.
```



从装饰程序变回普通程序

```
Fixpoint extract (d : dcom) : com :=
  match d with
  | DCSkip _           => CSkip
  | DCSeq d1 d2        => CSeq (extract d1) (extract d2)
  | DCAsgn X a _       => CAss X a
  | DCIf b _ d1 _ d2 _ => CIf b (extract d1) (extract d2)
  | DCWhile b _ d _    => CWhile b (extract d)
  | DCPre _ d          => extract d
  | DCPost d _         => extract d
  end.
```

```
Definition extract_dec (dec : decorated) : com :=
  match dec with
  | Decorated P d => extract d
  end.
```




获取装饰程序的前后条件

```
Definition pre_dec (dec : decorated) : Assertion :=  
  match dec with  
  | Decorated P d => P  
  end.
```

```
Definition post_dec (dec : decorated) : Assertion :=  
  match dec with  
  | Decorated P d => post d  
  end.
```



从装饰程序到命题

```
Fixpoint verification_conditions
  (P : Assertion) (d : dcom) : Prop :=
match d with
| DCSkip Q =>
  (P ->> Q)
| DCSeq d1 d2 =>
  verification_conditions P d1
  /\ verification_conditions (post d1) d2
| DCAsgn X a Q =>
  (P ->> Q [X |-> a])
| DCIf b P1 d1 P2 d2 Q =>
  ((P /\ b) ->> P1)%assertion
  /\ ((P /\ ~ b) ->> P2)%assertion
  /\ (post d1 ->> Q) /\ (post d2 ->> Q)
  /\ verification_conditions P1 d1
  /\ verification_conditions P2 d2
```



从装饰程序到命题

```
| DCWhile b Pbody d Ppost =>
    (* post d is the loop invariant and the initial
       precondition *)
    (P ->> post d)
    /\ ((post d /\ b) ->> Pbody)%assertion
    /\ ((post d /\ ~ b) ->> Ppost)%assertion
    /\ verification_conditions Pbody d
| DCPre P' d =>
    (P ->> P') /\ verification_conditions P' d
| DCPost d Q =>
    verification_conditions P d /\ (post d ->> Q)
end.
```



从装饰程序到命题：正确性

Theorem `verification_correct` : `forall` d P,
 `verification_conditions` P d -> `{{P}}` `extract` d `{{post d}}`.

Proof.

`induction` d; `intros`; `simpl in` *.

- (* Skip *)

`eapply` `hoare_consequence_pre`.

 + `apply` `hoare_skip`.

 + `assumption`.

(* 其他证明类似, 略 *)



从装饰程序到命题：正确性

```
Definition dec_correct (dec : decorated) :=  
  {{pre_dec dec}} extract_dec dec {{post_dec dec}}.
```

```
Definition verification_conditions_dec  
  (dec : decorated) : Prop :=  
  match dec with  
  | Decorated P d => verification_conditions P d  
  end.
```

```
Corollary verification_correct_dec : forall dec,  
  verification_conditions_dec dec -> dec_correct dec.
```



自动证明

- 多数情况借助之前定义的`verify_assn`可自动证明

```
Ltac verify :=  
  intros;  
  apply verification_correct;  
  verify_assn.
```

```
Theorem Dec_while_correct :  
  dec_correct dec_while.  
Proof. verify. Qed.
```

- 通常可以先尝试`verify`，对于证明不了的分支再手动证明



谓词转换计算

- 最弱前条件: $\{P\}$ 是 $c\{Q\}$ 的最弱前条件, 如果
 - $\{P\}c\{Q\}$
 - $\forall P'. \{P'\}c\{Q\} \Rightarrow \{P\}c\{Q\}$
- 最强后条件: $\{Q\}$ 是 $\{P\}c$ 的最强前条件, 如果
 - $\{P\}c\{Q\}$
 - $\forall Q'. \{P\}c\{Q'\} \Rightarrow Q \rightarrow Q'$
- 最弱前条件计算: 给定后条件和语句, 求能形成霍尔三元组的最弱前条件
- 最强后条件计算: 给定前条件和语句, 求能形成霍尔三元组的最强后条件



最弱前条件计算

- $wp(skip, Q) = Q$

$$\text{SKIP} \frac{}{\{P\} \text{ skip } \{P\}}$$

- $wp(x := a, Q) = Q[a/x]$

$$\text{ASSIGN} \frac{}{\{P[a/x]\} x := a \{P\}}$$

- $wp(c_1; c_2, Q) =$
 $wp(c_1, wp(c_2, Q))$

$$\text{SEQ} \frac{\{P\} c_1 \{R\} \quad \{R\} c_2 \{Q\}}{\{P\} c_1; c_2 \{Q\}}$$

- $wp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q) =$
 $(b \rightarrow wp(c_1, Q))$
 $\wedge (\neg b \rightarrow wp(c_2, Q))$

$$\text{IF} \frac{\{P \wedge b\} c_1 \{Q\} \quad \{P \wedge \neg b\} c_2 \{Q\}}{\{P\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \{Q\}}$$



最弱前条件： 举例

- $\text{wp}(\text{if } (x > 0) \ x \ += \ 10; \text{ else } \ x = 20, \ x > 0)$
 - $= (x > 0 \rightarrow \text{wp}(x \ += \ 10, \ x > 0)) \wedge (x \leq 0 \rightarrow \text{wp}(x = 20, \ x > 0))$
 - $= (x > 0 \rightarrow x + 10 > 0) \wedge (x \leq 0 \rightarrow 20 > 0)$
 - $= \text{True}$



最弱前条件计算：循环

- $wp(\text{while } b \text{ do } c, Q) = \exists i \in \text{Nat}. L_i(Q)$
 - where
 - $L_0(Q) = \text{false}$
 - $L_{i+1}(Q) = (\neg b \Rightarrow Q) \wedge (b \Rightarrow wp(c, L_i(Q)))$
- i 代表循环最多执行了 $i - 1$ 次
- 注意这个最弱前条件蕴含了循环必然终止



最强后条件计算

- $sp(P, skip) = P$
- $sp(P, x := a) = \exists n. x = a[n/x] \wedge P[n/x]$
- $sp(P, c_1; c_2) = sp(sp(P, c_1), c_2)$
- $sp(P, if\ b\ then\ c_1\ else\ c_2) = sp(b \wedge P, c_1) \vee sp(\neg b \wedge P, c_2)$
- $sp(P, while\ b\ do\ c) = \neg b \wedge \exists i. L_i(P)$
 - where
 - $L_0(P) = P$
 - $L_{i+1}(P) = sp(b \wedge L_i(P), c)$

因为约束更复杂，实际使用较少



作业

- 完成Hoare2中standard非optional习题
 - 请使用最新英文版教材