



软件理论基础与实践

Hoare: Hoare Logic, Part I

熊英飞
北京大学



动机

- 我们定义了IMP语言的语法语义
- 我们证明了IMP语言的性质，比如求值的确定性
- 我们定义了程序等价性，并论证了常见优化的正确性
- 这些都是关于语言设计和编译器的
- 能不能论证程序的（关于其语义的）性质？
 - 如何声明程序的性质？
 - 如何证明程序的性质？



霍尔逻辑

- Tony Hoare于1969年提出
- 受到Robert Floyd在流程图上类似工作的启发
- 也称Floyd-Hoare Logic
- 具体包括
 - 一种描述程序性质的方案：霍尔三元组
 - 一套推导霍尔三元组的规则



Tony Hoare
(80年图灵奖)



Robert Floyd
(78年图灵奖)



复习：形式系统

- 形式系统包括以下四个部分
 - 字母表Alphabet: 一个符号的集合 Σ
 - 文法Grammar: 一组语法规则, 定义 Σ^* 的一个子集, 为该形式系统中可以写的命题集合
 - 公理模式Axiom Schemata: 一组公理模板, 定义命题集合的一个子集, 代表为真的命题
 - 推导规则Inference Rules: 一组推导规则, 用于推导出公理以外为真的命题



霍尔三元组

- {前条件}语句{后条件}
- 如
 - $\{x > 0\} x := x + 5 \{x > 5\}$
 - $\{x > 0\} x := x + 5 \{x > 0\}$
 - $\{x = n \wedge y \neq 0\} x := x / y \{x * y = n\}$
 - $\{True\} \text{while}(\text{true}) x := x + 1 \{False\}$
- 如果霍尔三元组的前条件足够弱，后条件足够强，则精确描述了程序语义
- 所以霍尔逻辑又被称为公理语义



霍尔逻辑规则

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

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

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

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

$$\text{CONSEQUENCE} \frac{\vDash (P \Rightarrow P') \quad \{P'\} c \{Q'\} \quad \vDash (Q' \Rightarrow Q)}{\{P\} c \{Q\}}$$

$$\text{WHILE} \frac{\{P \wedge b\} c \{P\}}{\{P\} \text{ while } b \text{ do } c \{P \wedge \neg b\}}$$



用霍尔逻辑证明举例

- $\text{if } (x > 0) \ x := x+10 \ \text{else } x := 20$
 - 该程序执行结束后， x 是否一定大于0?
- 根据Assign，可得
 - $\{x+10>0\} \ x := x+10 \ \{x > 0\}$
 - $\{\text{True}\} \ x := 20 \ \{x > 0\}$
- 因为 $x > 0 \Rightarrow x+10 > 0$ 且 $\neg x > 0 \Rightarrow \text{True}$ ，根据Consequence，可得
 - $\{x > 0\} \ x := x+10 \ \{x > 0\}$
 - $\{\neg x > 0\} \ x := 20 \ \{x > 0\}$
- 根据If，可得
 - $\{\text{True}\} \ \text{if } (x > 0) \ x := x+10 \ \text{else } x := 20 \ \{x > 0\}$



用霍尔逻辑证明练习

- while ($x < 10$) $x := x+1$
 - 该程序执行结束后, x 是否一定大于0?
- 根据Assign, 可得
 - $\{\text{True}\} x := x+1 \{\text{True}\}$
- 根据Consequence, 可得
 - $\{x < 10 \wedge \text{True}\} x := x+1 \{\text{True}\}$
- 根据While, 可得
 - $\{\text{True}\} \text{while } (x < 10) x += 1; \{\text{True} \wedge x \geq 10\}$
- 根据Consequence, 可得
 - $\{\text{True}\} \text{while } (x < 10) x += 1; \{x > 0\}$



霍尔逻辑的性质

- 正确性Soundness: 所有用霍尔逻辑规则推导出来的霍尔三元组在IMP的语义下都是正确的, 即给定霍尔三元组 $\{P\}c\{Q\}$
 - 给定任意满足P的状态, 执行c后, Q一定满足
- 完整性Completeness: 所有在IMP语义下正确的霍尔三元组都可以用霍尔逻辑推导出来
- 本课程后续我们将证明这两个性质



Coq中的霍尔逻辑

- 基于IMP的语法和语义，将霍尔逻辑规则证明成定理
 - 即模型论的方法
- 基于IMP的语法，将霍尔逻辑规则定义成归纳定义命题的constructor
 - 即逻辑的方法
- 接下来我们首先用模型论的方法定义霍尔逻辑。



复习

- 什么是state?
 - **Definition** `state := total_map nat.`
- 什么是total_map?
 - **Definition** `total_map (A : Type) := string -> A.`
- st为状态, `X !-> 5`; st代表什么?
 - `t_update st "X" 5`
 - **Definition** `t_update {A : Type} (m : total_map A)(x : string) (v : A) := fun x' => if eqb_string x x' then v else m x'.`



断言：从状态到命题的函数

`Definition Assertion := state -> Prop.`

例子

- `fun st => st X = 3` holds if the value of `X` according to `st` is 3,
- `fun st => True` always holds, and
- `fun st => False` never holds.

后续将

`fun st => st X = m`

简写为

`X = m`

大写为IMP变量，小写为Coq变量



断言的蕴含关系

```
Definition assert_implies (P Q : Assertion) : Prop :=  
  forall st, P st -> Q st.
```

```
Declare Scope hoare_spec_scope.
```

```
Notation "P ->> Q" := (assert_implies P Q)  
  (at level 80) : hoare_spec_scope.
```

```
Open Scope hoare_spec_scope.
```

```
Notation "P <<->> Q" :=  
  (P ->> Q /\ Q ->> P) (at level 80) : hoare_spec_scope.
```



断言的简写语法

(* 注意区分Aexp和aexp *)

Definition Aexp : Type := state -> nat.

(* 自动转换普通Coq命题 *)

Definition assert_of_Prop (P : Prop) : Assertion := fun _ => P.

(* 自动转换整数 *)

Definition Aexp_of_nat (n : nat) : Aexp := fun _ => n.

(* 自动转换算术表达式aexp *)

Definition Aexp_of_aexp (a : aexp) : Aexp := fun st => aeval
st a.

Coercion assert_of_Prop : Sortclass >-> Assertion.

Coercion Aexp_of_nat : nat >-> Aexp.

Coercion Aexp_of_aexp : aexp >-> Aexp.

大致了解作用即可，无需知道细节



断言的简写语法

(* 自动展开函数定义 *)

Arguments assert_of_Prop /.

Arguments Aexp_of_nat /.

Arguments Aexp_of_aexp /.

(* 将三个scope的语法结合在一起 *)

Declare Scope assertion_scope.

Bind Scope assertion_scope with Assertion.

Bind Scope assertion_scope with Aexp.

Delimit Scope assertion_scope with assertion.

大致了解作用即可，无需知道细节



断言的简写语法

(* 用assert和mkAexp指定参数的类型 *)

Notation `assert P` := (P%assertion : Assertion).

Notation `mkAexp a` := (a%assertion : Aexp).

Notation `"~ P"` := (fun st => ~ assert P st) : assertion_scope.

Notation `"P /\ Q"` :=

(fun st => assert P st /\ assert Q st) : assertion_scope.

Notation `"P -> Q"` :=

(fun st => assert P st -> assert Q st) : assertion_scope.

Notation `"a = b"` :=

(fun st => mkAexp a st = mkAexp b st) : assertion_scope.

Notation `"a + b"` :=

(fun st => mkAexp a st + mkAexp b st) : assertion_scope.

(* 其他函数调用无法自动转换，所以用ap来显式调用 *)

Definition `ap {X} (f : nat -> X) (x : Aexp) :=`

`fun st => f (x st).`



断言书写举例

Definition ex1 : Assertion := X = 3.

Definition ex2 : Assertion := True.

Definition ex3 : Assertion := False.

Definition assn1 : Assertion := X <= Y.

Definition assn2 : Assertion := X = 3 \wedge X <= Y.

Definition assn3 : Assertion :=
Z * Z <= X \wedge \sim (((ap S Z) * (ap S Z)) <= X).

Definition assn4 : Assertion :=
Z = ap2 max X Y.



以下断言的写法均等价

- $X = 1 \rightarrow Y = 1$
- $\text{forall } st, \langle \{X=1\} \rangle st \rightarrow \langle \{Y=1\} \rangle st$
- $\text{forall } st, X \text{ st} = 1 \text{ st} \rightarrow Y \text{ st} = 1 \text{ st}$
- $\text{forall } st, st \ X = 1 \rightarrow st \ Y = 1$



霍尔三元组

Definition `hoare_triple`

```
(P : Assertion) (c : com) (Q : Assertion) : Prop :=  
forall st st',  
  st =[ c ]=> st' ->  
  P st ->  
  Q st'.
```

Notation "`{{ P }}` `c` `{{ Q }}`" :=

```
(hoare_triple P c Q) (at level 90, c custom com at level 99)  
: hoare_spec_scope.
```

Check (`{{True}}` X := 0 `{{True}}`).

将霍尔逻辑规则证明为定理 Skip



$$\frac{}{st = [\text{skip}] \Rightarrow st} \text{ (E_Skip)}$$

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

Theorem hoare_skip : forall P,
 {{P}} skip {{P}}.

Proof.

```
intros P st st' H HP.
```

```
(* H: st = [ skip ] => st'
```

```
HP: P st
```

```
Goal: P st' *)
```

```
inversion H; subst. assumption.
```

Qed.



Assignment

Definition `assn_sub X a (P:Assertion) : Assertion :=`
`fun (st : state) =>`
 `P (X !-> aeval st a ; st).`

Notation `"P [X |-> a]"` := (assn_sub X a P)
(at level 10, X at next level, a custom com).

注意Assertion定义为状态到命题的函数，没有语法



Assignment

$$\frac{\text{aeval st a = n}}{\text{st =[x := a]} \Rightarrow (\text{x} \mapsto \text{n} ; \text{st})} \quad (\text{E_Ass})$$

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

Theorem hoare_asgn : forall Q X a,
 {{Q [X ↦ a]}} X := a {{Q}}.

Proof.

```
intros Q X a st st' HE HQ.  
(* HE: st =[ X := a ] => st'  
   HQ: (Q [X ↦ a]) st  
   Goal: Q st' *)  
inversion HE. subst.  
(* HQ: (Q [X ↦ a]) st  
   Goal: Q (X ↦ aeval st a; st) *)  
assumption. Qed.
```



练习

- 如下这条霍尔逻辑规则正确吗？

$$\frac{}{\{ \text{True} \} X := a \{ X = a \}}$$

Consequence



$$\text{CONSEQUENCE} \frac{\vDash (P \Rightarrow P') \quad \{P'\} c \{Q'\} \quad \vDash (Q' \Rightarrow Q)}{\{P\} c \{Q\}}$$

Theorem `hoare_consequence_pre` : `forall` (P P' Q : Assertion) c,
 `{{P'}}` c `{{Q}}` ->
 P ->> P' ->
 `{{P}}` c `{{Q}}`.

Proof.

```
unfold hoare_triple, "->>".
intros P P' Q c Hhoare Himp st st' Heval Hpre.
(* Hhoare: {{P'}} c {{Q}}
   Hpre: P st
   Heval: st =[ c ]=> st'
   Himp: P ->> P'
   Goal: Q st' *)
apply Hhoare with (st := st).
- assumption.
- apply Himp. assumption.
```

Qed.



Consequence

Theorem `hoare_consequence_post` : `forall` (P Q Q' : Assertion) c,
 $\{\{P\}\} c \{\{Q'\}\} \rightarrow$
 $Q' \rightarrow\rightarrow Q \rightarrow$
 $\{\{P\}\} c \{\{Q\}\}.$

Proof.

```
intros P Q Q' c Hhoare Himp st st' Heval Hpre.
```

```
(* Hhoare:  $\{\{P\}\} c \{\{Q'\}\}$   
   Himp:  $Q' \rightarrow\rightarrow Q$   
   Heval:  $st = [ c ] \Rightarrow st'$   
   Hpre:  $P \ st$   
   Goal:  $Q \ st'$  *)
```

```
apply Himp.
```

```
apply Hhoare with (st := st).
```

```
- assumption.
```

```
- assumption.
```

Qed.



Consequence

```
Theorem hoare_consequence : forall (P P' Q Q' : Assertion) c,  
  {{P'}} c {{Q'}} ->  
  P ->> P' ->  
  Q' ->> Q ->  
  {{P}} c {{Q}}.
```

Proof.

```
intros P P' Q Q' c Htriple Hpre Hpost.  
(* Htriple: {{P'}} c {{Q'}}  
   Hpre: P ->> P'  
   Hpost: Q' ->> Q  
   Goal: {{P}} c {{Q}} *)  
apply hoare_consequence_pre with (P' := P').  
- apply hoare_consequence_post with (Q' := Q').  
  + assumption.  
  + assumption.  
- assumption.
```

Qed.



简化Consequence证明

Hint `Unfold assert_implies hoare_triple assn_sub t_update : core.`

Hint `Unfold assert_of_Prop Aexp_of_nat Aexp_of_aexp : core.`

Theorem `hoare_consequence_pre'` : `forall (P P' Q : Assertion) c,`
`{{P'}} c {{Q}} -> P ->> P' -> {{P}} c {{Q}}.`

Proof.

`eauto.`

Qed.

Theorem `hoare_consequence_post'` : `forall (P Q Q' : Assertion) c,`
`{{P}} c {{Q'}} -> Q' ->> Q -> {{P}} c {{Q}}.`

Proof.

`eauto.`

Qed.



证明示例

```
Example hoare_asgn_example1 :  
  {{True}} X := 1 {{X = 1}}.
```

Proof.

```
  apply hoare_consequence_pre with (P' := (X = 1) [X |-> 1]).  
  - (* {{(X = 1) [X |-> 1]}} X := 1 {{X = 1}} *)  
    apply hoare_asgn.  
  - (* True ->> (X = 1) [X |-> 1] *)  
    unfold "->>", assn_sub, t_update.  
    intros st _. simpl. reflexivity.
```

Qed.

或者

```
Example hoare_asgn_example1'' :  
  {{True}} X := 1 {{X = 1}}.
```

Proof.

```
  eauto using hoare_consequence_pre, hoare_asgn.
```

Qed.



证明示例

```
Example assn_sub_example2 :
```

```
  {{X < 4}}  
  X := X + 1  
  {{X < 5}}.
```

Proof.

```
apply hoare_consequence_pre with (P' := (X < 5) [X |-> X + 1]).  
- (* {{(X < 5) [X |-> X + 1]}} X := X + 1 {{X < 5}} *)  
  apply hoare_asgn.  
- (* X < 4 ->> (X < 5) [X |-> X + 1] *)  
  unfold "->>", assn_sub, t_update.  
  intros st H. simpl in *. lia.
```

Qed.

该证明用了lia，不能直接采用eauto证明。



自动化证明

- 证明所用序列其实对赋值证明非常通用

```
Ltac assn_auto :=  
  try auto;  
  try (unfold "->>", assn_sub, t_update;  
        intros; simpl in *; lia).
```

```
Example assn_sub_example2'' :  
  {{X < 4}}  
  X := X + 1  
  {{X < 5}}.
```

Proof.

```
eapply hoare_consequence_pre.  
- eauto.  
- assn_auto.
```

Qed.



Sequencing

$$\frac{\begin{array}{l} st = [c_1] \Rightarrow st' \\ st' = [c_2] \Rightarrow st'' \end{array}}{st = [c_1; c_2] \Rightarrow st''} \quad (\text{E_Seq})$$

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

Theorem `hoare_seq` : forall P Q R c1 c2,
 {{Q}} c2 {{R}} ->
 {{P}} c1 {{Q}} ->
 {{P}} c1; c2 {{R}}.

Proof.

```
unfold hoare_triple.  
intros P Q R c1 c2 H1 H2 st st' H12 Pre.  
(* H1: {{Q}} c2 {{R}}  
   H2: {{P}} c1 {{Q}}  
   H12: st = [ c1; c2 ] => st'  
   Pre: P st  
   Goal: Q st' *)  
inversion H12; subst.  
eauto.
```

Qed.



Sequencing证明示例

Example `hoare_asgn_example3` : `forall` (a:aexp) (n:nat),
 `{{a = n}}`
 `X := a; skip`
 `{{X = n}}`.

Proof.

```
intros a n. eapply hoare_seq.  
- (* {{?Q}} skip {{X = n}} *)  
  apply hoare_skip.  
- (* {{a = n}} X := a {{X = n}} *)  
  eapply hoare_consequence_pre.  
  + apply hoare_asgn.  
  + assn_auto.
```

Qed.



If

- If规则中用合取连接了Assertion和bexp

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

- 定义函数将bexp提升为Assertion

```
Definition bassn b : Assertion :=  
  fun st => (beval st b = true).
```

```
Coercion bassn : bexp -> Assertion.
```



If

```
Theorem hoare_if : forall P Q (b:bexp) c1 c2,  
  {{ P /\ b }} c1 {{Q}} ->  
  {{ P /\ ~ b }} c2 {{Q}} ->  
  {{P}} if b then c1 else c2 end {{Q}}.  
(* That is (unwrapping the notations):
```

```
  Theorem hoare_if : forall P Q b c1 c2,  
    {{fun st => P st /\ bassn b st}} c1 {{Q}} ->  
    {{fun st => P st /\ ~ (bassn b st)}} c2 {{Q}} ->  
    {{P}} if b then c1 else c2 end {{Q}}.
```

*)

Proof.

```
  intros P Q b c1 c2 HTrue HFalse st st' HE HP.  
  inversion HE; subst; eauto.
```

Qed.



If证明示例

```
Example if_example :  
  {{True}}  
  if (X = 0)  
  then Y := 2  
  else Y := X + 1  
end  
  {{X <= Y}}.
```

Proof.

```
apply hoare_if.  
- (* Then *)  
  eapply hoare_consequence_pre.  
  + apply hoare_asgn.  
  + (* True/\X=0 ->>  
      (X<=Y)[Y!->2] *)  
    assn_auto. (* no progress *)  
    unfold "->>", assn_sub,  
      t_update, bassn.  
    simpl. intros st [_ H].  
    (* H: (st X =? 0) = true  
       Goal: st X <= 2 *)  
    apply eqb_eq in H.  
    rewrite H. lia.  
- (* Else *)  
  eapply hoare_consequence_pre.  
  + apply hoare_asgn.  
  + assn_auto.
```

Qed.



If证明例子-改造策略

```
Ltac assn_auto' :=  
  unfold "->>", assn_sub, t_update, bassn;  
  intros; simpl in *;  
  try rewrite -> eqb_eq in *; (* for equalities *)  
  auto; try lia.
```

```
Example if_example'' :  
  {{True}}  
  if X = 0  
    then Y := 2  
    else Y := X + 1  
  end  
  {{X <= Y}}.
```

Proof.

```
  apply hoare_if; eapply hoare_consequence_pre;  
    try apply hoare_asgn; try assn_auto'.
```

Qed.

While

$$\frac{\text{beval } st \ b = \text{false}}{st = [\text{while } b \ \text{do } c \ \text{end}] \Rightarrow st} \quad (\text{E_WhileFalse})$$

$$\frac{\begin{array}{c} \text{beval } st \ b = \text{true} \\ st = [c] \Rightarrow st' \\ st' = [\text{while } b \ \text{do } c \ \text{end}] \Rightarrow st'' \end{array}}{st = [\text{while } b \ \text{do } c \ \text{end}] \Rightarrow st''} \quad (\text{E_WhileTrue})$$

$$\text{WHILE} \frac{\{P \wedge b\} c \{P\}}{\{P\} \text{ while } b \ \text{do } c \{P \wedge \neg b\}}$$

Theorem `hoare_while` : forall P (b:bexp) c,
 {{P /\ b}} c {{P}} ->
 {{P}} while b do c end {{P /\ ~ b}}.

Proof.

```
intros P b c Hhoare st st' Heval HP.  
(* Hhoare: {{P /\ b}} c {{P}}  
   Heval: st = [ while b do c end ] => st'  
   HP: P st  
   Goal: P st' /\ ~ b st'*)
```

```
remember <{while b do c end}> as original_command eqn:Horig.
```

```
induction Heval;
```

```
  try (inversion Horig; subst; clear Horig); (* 剩下以上两种情况 *)  
  eauto.
```

Qed.

为什么需要
remember?



While证明示例

```
Example while_example :  
  {{X <= 3}}  
  while (X <= 2) do  
    X := X + 1  
  end  
  {{X = 3}}.
```

Proof.

```
eapply hoare_consequence_post.  
(* {{X <= 3}} while X <= 2 do X := X + 1 {{?Q}}  
  ?Q ->> X = 3 *)  
- apply hoare_while.  
(* {{X <= 3 /\ X <= 2}} X := X + 1 {{X <= 3}} *)  
eapply hoare_consequence_pre.  
+ (* {{?P}} X := X + 1 {{X <= 3}}*)  
  apply hoare_asgn.  
+ (* (X <= 3 /\ X <= 2) ->> (X <= 3) [X |-> X + 1] *)  
  assn_auto'' .  
- (* (X <= 3 /\ ~ (X <= 2)) ->> X = 3 *)  
  assn_auto'' .
```

Qed.



不终止程序满足任何后条件

Theorem `always_loop_hoare` : forall Q,
 {{True}} while true do skip end {{Q}}.

Proof.

```
intros Q.  
eapply hoare_consequence_post.  
(* {{True}} while true do skip end {{?Q'}}  
  ?Q' ->> Q *)  
- apply hoare_while.  
  (* {{True /\ <{true}>}} skip {{True}}*)  
  apply hoare_post_true.  
  (* forall st : state, True st *)  
  auto.  
- (* (True /\ <{~true}>) ->> Q*)  
  simpl. intros st [Hinv Hguard]. congruence.
```

Qed.

`congruence`策略搜索两条矛盾的前提并推出任意结论
`congruence`可以替代之前定义的`find_rwd`策略



部分正确性 vs 完全正确性

- 标准霍尔逻辑是部分正确性的
 - 不保证程序终止
 - 程序不终止的时候允许任何后条件
- 可以扩展While规则实现完全正确性
 - 满足前条件的时候程序一定终止，
 - 且一定满足后条件

$$\frac{P \wedge b \rightarrow E \geq 0 \quad [P \wedge b \wedge E = n] S [P \wedge E < n]}{[P] \text{ while } b \text{ do } s [P \wedge \neg b]}$$



练习

- 考虑部分正确性，如果扩充语言加入如下成分，其霍尔规则是什么？
 - if b then c
 - 类似于C语言中没有else的if
 - repeat c until b
 - 同IMP部分的定义，重复执行至少一遍c直到b满足
 - assume b
 - $\frac{\text{beval st } b = \text{true}}{\text{st}=[\text{assume } b] \Rightarrow \text{st}}$
 - assert b
 - $\frac{\text{beval st } b = \text{true}}{\text{st}=[\text{assert } b] \Rightarrow \text{st}}$
 - $\frac{\text{beval st } b = \text{false}}{\text{st}=[\text{assert } b] \Rightarrow \text{error}}$
 - $\frac{}{\text{error}=[c] \Rightarrow \text{error}}$
 - 在error上任何assertion都不成立



答案

- if b then c

- $$\frac{\{P \wedge b\}c\{Q\} \quad P \wedge \neg b \rightarrow Q}{\{P\} \text{ if } b \text{ then } c \{Q\}}$$

- repeat c until b

- $$\frac{\{P\}c\{Q\} \quad \{\neg b \wedge Q\}c\{Q\}}{\{P\} \text{ repeat } c \text{ until } b \{Q \wedge b\}}$$

- assume b

- $\{P\} \text{ assume } b \{b \wedge P\}$

- assert b

- $\{b \wedge P\} \text{ assert } b \{b \wedge P\}$



作业

- 完成Hoare中standard非optional并不属于Additional Exercises的11道习题
 - 请使用最新英文版教材
 - 推荐也完成Havoc部分的习题