SOFTWARE FOUNDATIONS

VOLUME 2: PROGRAMMING LANGUAGE FOUNDATIONS

HOAREASLOGIC

HOARE LOGIC AS A LOGIC

The presentation of Hoare logic in chapter Hoare could be described as "model-theoretic": the proof rules for each of the constructors were presented as *theorems* about the evaluation behavior of programs, and proofs of program correctness (validity of Hoare triples) were constructed by combining these theorems directly in Coq.

Another way of presenting Hoare logic is to define a completely separate proof system — a set of axioms and inference rules that talk about commands, Hoare triples, etc. — and then say that a proof of a Hoare triple is a valid derivation in *that* logic. We can do this by giving an inductive definition of *valid derivations* in this new logic.

This chapter is optional. Before reading it, you'll want to read the ProofObjects chapter in *Logical Foundations* (*Software Foundations*, volume 1).

```
Require Import Imp.
Require Import Hoare.
```

Definitions

```
Inductive hoare_proof : Assertion → com → Assertion → Type :=
  | H_Skip : ∀ P,
      hoare_proof P (SKIP) P
  | H_Asgn : ∀ Q V a,
      hoare_proof (assn_sub V a Q) (V ::= a) Q
  | H_Seq : ∀ P c Q d R,
      hoare_proof P c Q → hoare_proof Q d R → hoare_proof P
(c;;d) R
  | H_If : ∀ P Q b c1 c2,
    hoare_proof (fun st ⇒ P st ∧ bassn b st) c1 Q →
    hoare_proof (fun st ⇒ P st ∧ ~(bassn b st)) c2 Q →
    hoare_proof P (IFB b THEN c1 ELSE c2 FI) Q
  | H_While : ∀ P b c,
```

```
      hoare_proof (fun st ⇒ P st ∧ bassn b st) c P →
      hoare_proof P (WHILE b DO c END) (fun st ⇒ P st ∧ ¬ (bassn b
st))
    | H_Consequence : ∀ (P Q P' Q' : Assertion) c,
      hoare_proof P' c Q' →
      (∀ st, P st → P' st) →
      (∀ st, Q' st → Q st) →
      hoare_proof P c Q.
```

We don't need to include axioms corresponding to `hoare_consequence_pre` or `hoare_consequence_post`, because these can be proven easily from `H_Consequence`.

```
Lemma H_Consequence_pre : ∀ (P Q P': Assertion) c,
    hoare_proof P' c Q →
    (∀ st, P st → P' st) →
    hoare_proof P c Q.
 +

Lemma H_Consequence_post : ∀ (P Q Q' : Assertion) c,
    hoare_proof P c Q' →
    (∀ st, Q' st → Q st) →
    hoare_proof P c Q.
 +
```

As an example, let's construct a proof object representing a derivation for the hoare triple

```
      {{(X=3) [X |-> X + 2] [X |-> X + 1]}}
      X::=X+1 ;; X::=X+2
      {{X=3}}.
```

We can use Coq's tactics to help us construct the proof object.

```
Example sample_proof :
  hoare_proof
    ((fun st:state ⇒ st X = 3) [X |-> X + 2] [X |-> X + 1])
    (X ::= X + 1;; X ::= X + 2)
    (fun st:state ⇒ st X = 3).
Proof.
  eapply H_Seq; apply H_Asgn.
Qed.

(*
Print sample_proof.

====>
  H_Seq
  (((fun st : state => st X = 3) X |-> X + 2) X |-> X + 1)
  (X ::= X + 1)
  ((fun st : state => st X = 3) X |-> X + 2)
  (X ::= X + 2)
  (fun st : state => st X = 3)
  (H_Asgn
     ((fun st : state => st X = 3) X |-> X + 2)
```

```
      X (X + 1))
   (H_Asgn
      (fun st : state => st X = 3)
      X (X + 2))
*)
```

Properties

Exercise: 2 stars (hoare_proof_sound)

Prove that such proof objects represent true claims.

```
Theorem hoare_proof_sound : ∀ P c Q,
  hoare_proof P c Q → {{P}} c {{Q}}.
Proof.
  (* FILL IN HERE *) Admitted.
```

□

We can also use Coq's reasoning facilities to prove metatheorems about Hoare Logic. For example, here are the analogs of two theorems we saw in chapter Hoare — this time expressed in terms of the syntax of Hoare Logic derivations (provability) rather than directly in terms of the semantics of Hoare triples.

The first one says that, for every `P` and `c`, the assertion `{{P}} c {{True}}` is *provable* in Hoare Logic. Note that the proof is more complex than the semantic proof in `Hoare`: we actually need to perform an induction over the structure of the command `c`.

```
Theorem H_Post_True_deriv:
  ∀ c P, hoare_proof P c (fun _ ⇒ True).
Proof.
  intro c.
  induction c; intro P.
  - (* SKIP *)
    eapply H_Consequence.
    apply H_Skip.
    intros. apply H.
    (* Proof of True *)
    intros. apply I.
  - (* ::= *)
    eapply H_Consequence_pre.
    apply H_Asgn.
    intros. apply I.
  - (* ;; *)
    eapply H_Consequence_pre.
    eapply H_Seq.
    apply (IHc1 (fun _ ⇒ True)).
    apply IHc2.
    intros. apply I.
  - (* IFB *)
    apply H_Consequence_pre with (fun _ ⇒ True).
    apply H_If.
    apply IHc1.
    apply IHc2.
    intros. apply I.
```

```
    - (* WHILE *)
      eapply H_Consequence.
      eapply H_While.
      eapply IHc.
      intros; apply I.
      intros; apply I.
Qed.
```

Similarly, we can show that `{{False}} c {{Q}}` is provable for any `c` and `Q`.

```
Lemma False_and_P_imp: ∀ P Q,
  False ∧ P → Q.
Proof.
  intros P Q [CONTRA HP].
  destruct CONTRA.
Qed.

Tactic Notation "pre_false_helper" constr(CONSTR) :=
  eapply H_Consequence_pre;
    [eapply CONSTR | intros ? CONTRA; destruct CONTRA].

Theorem H_Pre_False_deriv:
  ∀ c Q, hoare_proof (fun _ ⇒ False) c Q.
Proof.
  intros c.
  induction c; intro Q.
  - (* SKIP *) pre_false_helper H_Skip.
  - (* ::= *) pre_false_helper H_Asgn.
  - (* ;; *) pre_false_helper H_Seq. apply IHc1. apply IHc2.
  - (* IFB *)
    apply H_If; eapply H_Consequence_pre.
    apply IHc1. intro. eapply False_and_P_imp.
    apply IHc2. intro. eapply False_and_P_imp.
  - (* WHILE *)
    eapply H_Consequence_post.
    eapply H_While.
    eapply H_Consequence_pre.
      apply IHc.
      intro. eapply False_and_P_imp.
    intro. simpl. eapply False_and_P_imp.
Qed.
```

As a last step, we can show that the set of `hoare_proof` axioms is sufficient to prove any true fact about (partial) correctness. More precisely, any semantic Hoare triple that we can prove can also be proved from these axioms. Such a set of axioms is said to be *relatively complete*. Our proof is inspired by this one:

http://www.ps.uni-saarland.de/courses/sem-ws11/script/Hoare.html

To carry out the proof, we need to invent some intermediate assertions using a technical device known as *weakest preconditions*. Given a command `c` and a desired postcondition assertion `Q`, the weakest precondition `wp c Q` is an assertion `P` such that `{{P}} c {{Q}}` holds, and moreover, for any other assertion `P'`, if `{{P'}} c {{Q}}` holds then `P' → P`. We can more directly define this as follows:

```
Definition wp (c:com) (Q:Assertion) : Assertion :=
  fun s ⇒ ∀ s', c / s \\ s' → Q s'.
```

Exercise: 1 star (wp_is_precondition)

```
Lemma wp_is_precondition: ∀ c Q,
  {{wp c Q}} c {{Q}}.
(* FILL IN HERE *) Admitted.
```

☐

Exercise: 1 star (wp_is_weakest)

```
Lemma wp_is_weakest: ∀ c Q P',
   {{P'}} c {{Q}} → ∀ st, P' st → wp c Q st.
(* FILL IN HERE *) Admitted.
```

The following utility lemma will also be useful.

```
Lemma bassn_eval_false : ∀ b st, ¬ bassn b st → beval st b =
false.
Proof.
  intros b st H. unfold bassn in H. destruct (beval st b).
    exfalso. apply H. reflexivity.
    reflexivity.
Qed.
```

☐

Exercise: 5 stars (hoare_proof_complete)

Complete the proof of the theorem.

```
Theorem hoare_proof_complete: ∀ P c Q,
  {{P}} c {{Q}} → hoare_proof P c Q.
Proof.
  intros P c. generalize dependent P.
  induction c; intros P Q HT.
  - (* SKIP *)
    eapply H_Consequence.
     eapply H_Skip.
      intros. eassumption.
      intro st. apply HT. apply E_Skip.
  - (* ::= *)
    eapply H_Consequence.
      eapply H_Asgn.
      intro st. apply HT. econstructor. reflexivity.
      intros; assumption.
  - (* ;; *)
    apply H_Seq with (wp c₂ Q).
     eapply IHc1.
       intros st st' E₁ H. unfold wp. intros st'' E₂.
         eapply HT. econstructor; eassumption. assumption.
     eapply IHc2. intros st st' E₁ H. apply H; assumption.
  (* FILL IN HERE *) Admitted.
```

☐

Finally, we might hope that our axiomatic Hoare logic is *decidable*; that is, that there is an (terminating) algorithm (a *decision procedure*) that can determine whether or not a given Hoare triple is valid (derivable). But such a decision procedure cannot exist!

Consider the triple `{{True}} c {{False}}`. This triple is valid if and only if `c` is non-terminating. So any algorithm that could determine validity of arbitrary triples could solve the Halting Problem.

Similarly, the triple `{{True}} SKIP {{P}}` is valid if and only if `∀s, P s` is valid, where `P` is an arbitrary assertion of Coq's logic. But it is known that there can be no decision procedure for this logic.

Overall, this axiomatic style of presentation gives a clearer picture of what it means to "give a proof in Hoare logic." However, it is not entirely satisfactory from the point of view of writing down such proofs in practice: it is quite verbose. The section of chapter Hoare2 on formalizing decorated programs shows how we can do even better.