CS720

Logical Foundations of Computer Science

Lecture 14: Program verification

Tiago Cogumbreiro

Summary

- Learn how to design a framework to prove properties about programs
  (We will develop the Floyd-Hoare Logic.)
  - Assigning Meanings to Programs. Robert W. Floyd. 1967
  - An axiomatic basis for computer programming. C. A. R. Hoare. 1969
- Introduce pre and post-conditions on commands

How do we **specify** an algorithm?

How do we **specify** an algorithm?

A formal specification describes ***what*** a system does

(and not ***how*** a system does it)

How do we **observe**

what an `Imp` program does?

What are its inputs and outputs?

We **observe** an Imp program

via its input/output state

Specifying Imp programs

How do we reason about the inputs/outputs?

- Input/output of an Imp program is a ***state***.
- - Let us call the formalize reasoning about an `Imp` state as an **assertion**, notation $\{P\}$, for some proposition P that accesses an implicit state:

```
Definition Assertion := state → Prop.
```

Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as `fun st ⇒ st X = 3`

Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as `fun st ⇒ st X = 3`
2. $\{x \leq y\}$ written as `fun st ⇒ st X ≤ st Y`

Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as `fun st ⇒ st X = 3`
2. $\{x \leq y\}$ written as `fun st ⇒ st X ≤ st Y`
3. $\{x = 3 \vee x \leq y\}$ written as `fun st ⇒ st X = 3 \/ st X ≤ st Y`

Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as `fun st ⇒ st X = 3`
2. $\{x \leq y\}$ written as `fun st ⇒ st X ≤ st Y`
3. $\{x = 3 \vee x \leq y\}$ written as `fun st ⇒ st X = 3 \/ st X ≤ st Y`
4. $z \times z \leq x \wedge \neg((z+1) \times (z+1) \leq x)$ written as
   `fun st ⇒ st Z * st Z ≤ st X /\ ~ (((S (st Z)) * (S (st Z))) ≤ st X)`

Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as `fun st ⇒ st X = 3`
2. $\{x \leq y\}$ written as `fun st ⇒ st X ≤ st Y`
3. $\{x = 3 \vee x \leq y\}$ written as `fun st ⇒ st X = 3 \/ st X ≤ st Y`
4. $z \times z \leq x \wedge \neg((z+1) \times (z+1) \leq x)$ written as
   `fun st ⇒ st Z * st Z ≤ st X /\ ~ (((S (st Z)) * (S (st Z))) ≤ st X)`
5. What about `fun st ⇒ True`?

Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as `fun st ⇒ st X = 3`
2. $\{x \leq y\}$ written as `fun st ⇒ st X ≤ st Y`
3. $\{x = 3 \lor x \leq y\}$ written as `fun st ⇒ st X = 3 \/ st X ≤ st Y`
4. $z \times z \leq x \land \neg((z + 1) \times (z + 1) \leq x)$ written as
   `fun st ⇒ st Z * st Z ≤ st X /\ ~ (((S (st Z)) * (S (st Z))) ≤ st X)`
5. What about `fun st ⇒ True`?
6. What about `fun st ⇒ False`?

A Hoare Triple

Combining assertions with commands

A **Hoare triple**, notation $\{P\}\ c\ \{Q\}$, holds if, and only if, from $P(s)$ and `ceval` $s\ c\ s$ we can obtain $Q(s')$ for any states s and s'.

```
Definition hoare_triple (P:Assertion) (c:com) (Q:Assertion) : Prop :=
  forall st st',
  P st →              (* If [P st] holds *)
  ceval st c st' →   (* And [c] runs with an input state [st] yielding a state [st'] *)
  Q st'.              (* Then [Q st'] holds *)
```

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \wedge x = 3\}\ x := 5\ \{x = 0\}$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \wedge x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \land x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \land x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume** $x = 1$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \wedge x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume $x = 1$**
4. $\{\top\}\ \texttt{skip}\ \{\bot\}$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \land x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume $x = 1$**
4. $\{\top\}\ \texttt{skip}\ \{\bot\}$ **Improvable, because the conclusion is not provable.**

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \land x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume** $x = 1$
4. $\{\top\}\ \texttt{skip}\ \{\bot\}$ **Improvable, because the conclusion is not provable.**
5. $\{x = 1\}\ \texttt{while}\ x \neq 0\ \texttt{do}\ x := x + 1\ \texttt{end}\ \{x = 100\}$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \wedge x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume** $x = 1$
4. $\{\top\}\ \texttt{skip}\ \{\bot\}$ **Improvable, because the conclusion is not provable.**
5. $\{x = 1\}\ \texttt{while}\ x \neq 0\ \texttt{do}\ x := x + 1\ \texttt{end}\ \{x = 100\}$ **Provable, because the loop is not provable, so we can reach a contradiction.**

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \land x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume** $x = 1$
4. $\{\top\}\ \texttt{skip}\ \{\bot\}$ **Improvable, because the conclusion is not provable.**
5. $\{x = 1\}\ \texttt{while}\ x \neq 0\ \texttt{do}\ x := x + 1\ \texttt{end}\ \{x = 100\}$ **Provable, because the loop is not provable, so we can reach a contradiction.**
6. $\{x = 1\}\ \texttt{skip}\ \{x \geq 1\}$

Exercise

Which of these programs are provable?

1. $\{\top\}\ x := 5; y := 0\ \{x = 5\}$ **Provable**
2. $\{x = 2 \land x = 3\}\ x := 5\ \{x = 0\}$ **Provable, because the pre-condition is false**
3. $\{\top\}\ x := x + 1\ \{x = 2\}$ **Improvable, because there's not enough information to assume** $x = 1$
4. $\{\top\}\ \texttt{skip}\ \{\bot\}$ **Improvable, because the conclusion is not provable.**
5. $\{x = 1\}\ \texttt{while}\ x \neq 0\ \texttt{do}\ x := x + 1\ \texttt{end}\ \{x = 100\}$ **Provable, because the loop is not provable, so we can reach a contradiction.**
6. $\{x = 1\}\ \texttt{skip}\ \{x \geq 1\}$ **Provable, the state is unchanged, but we can conclude.**

Let us build a theory on Hoare triples over Imp

(That is, define theorems to help us prove results on Hoare triples.)

Skip

Theorem (H-skip): for any proposition P we have that $\{P\}$ **`skip`** $\{P\}$.

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

Sequence

Theorem (H-seq): If $\{P\}\ c_1\ \{Q\}$ and $\{Q\}\ c_2\ \{R\}$, then

Sequence

Theorem (H-seq): If $\{P\}\ c_1\ \{Q\}$ and $\{Q\}\ c_2\ \{R\}$, then $\{P\}\ c_1; c_2\ \{R\}$.

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

We have seen how to derive theorems for some commands,

Let us derive a theorem over the assignment

Assignment

How do we derive a general-enough theorem over the assignment?

> ***Idea:*** try to prove `False` and simplify the hypothesis.

```
Goal forall P a,
  {{ fun st ⇒ P st }} X := a {{ fun st ⇒ P st /\ False }}.
Proof.
  intros.
  intros s_in s_out Ha Hb.
  invc Ha.
```

Yields

```
Hb : P s_in
______________________________________(1/1)
P (X !-> aeval s_in a; s_in) /\ False
```

Deriving the rule for the assignment

The proof state tells us that the pre-condition does not have enough information.

```
Hb : P s_in
______________________________________(1/1)
P (X !-> aeval s_in a; s_in) /\ False
```

Deriving the rule for assignment

The following result should is provable.

```
Goal forall P a,
  {{ fun st ⇒ P st /\ st X = aeval st a }}
  X := a
  {{ fun st ⇒ P st }}.
```

Deriving the rule for assignment

The following result should is provable.

```
Goal forall P a,
  {{ fun st ⇒ P st /\ st X = aeval st a }}
  X := a
  {{ fun st ⇒ P st }}.

Proof.
  intros.
  intros s_in s_out Ha [Hb Hc].
  invc Ha.
  rewrite ← Hc.
  rewrite t_update_same.
  assumption.
Qed.
```

Deriving the rule for assignment

Making the code read more like the paper

```
{{ fun st ⇒ P st /\ st X = aeval st a }} X:= a {{ fun st ⇒ P st }}
```

becomes

```
{{P [X |-> a]}} X := a {{P}}
```

Abstracting a state update with evaluation

Another level of indirection

Read `P [ X |-> a ]` as:

> assertion `P` where `X` is assigned to the ***value*** of expression `a`

```
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).
```

Understanding the notation

```
  (X ≤ 5) [X |-> 3]
  \------/
     P = (fun st' ⇒ st' X ≤ 5)

= P [ X |-> 3 ]                                          (1. unfold notation)
= assn_sub X 3 P                                         (2. apply assn_sub to args)
= fun st ⇒
    P (X !-> aeval st 3; st)                             (3. apply aeval to args)
= fun st ⇒
    P (X !-> 3; st)                                      (4. unfold P)
= fun st ⇒
    (fun st' ⇒ 0 ← st' X ≤ 5) (X !-> 3; st) (5. apply function to arg)
= fun st ⇒
    (X !-> 3; st) X ≤ 5                                  (6. apply function to arg)
= fun st ⇒
    3 ≤ 5
```

Backward style assignment rule

Theorem (H-asgn): $\{P[x \mapsto a]\}\ x := a\ \{P\}$.

```
Theorem hoare_asgn: forall a P,
  {{ fun st ⇒ P (st ; { X ⟶ aeval st a }) }}
    X := a
  {{ fun st ⇒ P st }}.
```

Exercise

Does $\{x = 2[x \mapsto x + 1][x \mapsto 1]\}\ x := 1; x := x + 1\ \{x = 2\}$ hold?

```
Goal {{ (fun st : state ⇒ st X = 2) [X |-> X + 1] [ X |-> 1] }}
       X := 1; X := X + 1
     {{ fun st ⇒ st X = 2 }}.
```

Exercise

Does $\{x = 2[x \mapsto x + 1][x \mapsto 1]\}\ x := 1; x := x + 1\ \{x = 2\}$ hold?

```
Goal {{ (fun st : state => st X = 2) [X |-> X + 1] [ X |-> 1] }}
       X := 1; X := X + 1
     {{ fun st => st X = 2 }}.
```

Yes.

Exercise

Does $\{x = 2[x \mapsto x+1][x \mapsto 1]\}\ x := 1; x := x + 1\ \{x = 2\}$ hold?

```
Goal {{ (fun st : state => st X = 2) [X |-> X + 1] [ X |-> 1] }}
       X := 1; X := X + 1
     {{ fun st => st X = 2 }}.
```

Yes. Does $\{\top\}\ x := 1;; x := x + 1\ \{x = 2\}$ hold? And, can we prove it T-seq and T-asgn?

```
Goal {{ fun st => True }}   X := 1; X := X + 1   {{ fun st => st X = 2 }}.
```

Exercise

Does $\{x = 2[x \mapsto x+1][x \mapsto 1]\}\ x := 1; x := x+1\ \{x = 2\}$ hold?

```
Goal {{ (fun st : state => st X = 2) [X |-> X + 1] [ X |-> 1] }}
       X := 1; X := X + 1
     {{ fun st => st X = 2 }}.
```

Yes. Does $\{\top\}\ x := 1; ; x := x+1\ \{x = 2\}$ hold? And, can we prove it T-seq and T-asgn?

```
Goal {{ fun st => True }}   X := 1; X := X + 1   {{ fun st => st X = 2 }}.
```

No. The pre-condition has to match what we stated H-asgn. But we know that the above statement holds. Let us write a new theorem that handles such cases.

Summary

Here are theorems we've proved today:

$$\{P\}\ \texttt{SKIP}\ \{P\} \qquad \text{(H-skip)}$$

$$\frac{\{P\}\ c_1\ \{Q\} \qquad \{Q\}\ c_2\ \{R\}}{\{P\}\ c_1; c_2\ \{R\}} \qquad \text{(H-seq)}$$

$$\{P[x \mapsto a]\}\ x := a\ \{P\} \qquad \text{(H-asgn)}$$

Summary

- Learn how to design a framework to prove properties about programs
  (We will develop the Floyd-Hoare Logic.)
- Introduce pre and post-conditions on commands
- Notations keep the formalism close to the mathematical intuition
- While doing the proofs you need to know ***every*** level of the notations