CS720

Logical Foundations of Computer Science

Lecture 14: Program verification

Tiago Cogumbreiro

`Imp.v`

Due Thursday October 18, 11:59pm EST

`IndProp.v`

Due Friday October 19, 11:59pm EST

Equiv.v

Due Thursday October 25, 11:59pm EST

Hoare.v

Due Thursday November 1, 11:59pm EST

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

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?

Specifying Imp programs

The input and the 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.
```

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. $z \times z \wedge \neg((z+1) \times (z+1) \leq x)$ written as
   `fun st ⇒ st Z * st Z ≤ st X /\ ~ (((S (st Z)) * (S (st Z))) ≤ st X)`
5. What about `fun st ⇒ True`?
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 $c\ /\ s\ \backslash\backslash\ s'$ 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 *)
  c / st \\ 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. $\{\top\}\ x ::= 5;;\ y ::= 0\ \{x = 5\}$
2. $\{x = 2 \wedge x = 3\}\ x ::= 5\ \{x = 0\}$
3. $\{\top\}\ x ::= x + 1\ \{x = 2\}$
4. $\{\top\}\ \texttt{SKIP}\ \{\bot\}$
5. $\{x = 1\}\ \texttt{WHILE}\ !(x = 0)\ \texttt{DO}\ x ::= x + 1\ \texttt{END}\ \{x = 100\}$

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 }}.
```

- How do we mention pre-updates?

Reasoning about pre-update

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

Reasoning about pre-update

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

we are stuck here

```
H: st X = m
H0 : P st
______________________________________(1/1)
P (st & {X ⟶ aeval st a})
```

> What happens if we change our post-condition?

Second try

Let us change the post-condition to understand how it affects our goal

```
Goal forall P a m,
  {{ fun st ⇒ P st /\ st X = m }}
    X ::= a
  {{ fun st ⇒ P (st & { X ⟶ 3 }) }}.
```

> Updating the store of the post-condition *shadows* the update to `a`

```
H: st X = m
H0: P st
____________________________________(1/1)
P (st & {X ⟶ aeval st a; X ⟶ 3})
```

Second try

Let us change the post-condition to understand how it affects our goal

```
Goal forall P a m,
  {{ fun st ⇒ P st /\ st X = m }}
    X ::= a
  {{ fun st ⇒ P (st & { X ⟶ 3 }) }}.
```

> Updating the store of the post-condition *shadows* the update to `a`

```
H: st X = m
H0: P st
______________________________________(1/1)
P (st & {X ⟶ aeval st a; X ⟶ 3})
```

> What if we "cancel out" the update?

Reasoning about the post-update

```
Goal forall P a m,
  {{ fun st ⇒ P st /\ st X = m }}
    X ::= a
  {{ fun st ⇒ P (st & { X ⟶ m }) }}.
```

Reasoning about the post-update

```
Goal forall P a m,
  {{ fun st ⇒ P st /\ st X = m }}
    X ::= a
  {{ fun st ⇒ P (st & { X ⟶ m }) }}.
```

We are still not there yet. How do we derive the post-value?

Reasoning about the post-update

```
Goal forall P a m,
  {{ fun st ⇒ P st /\ st X = m }}
    X ::= a
  {{ fun st ⇒ P (st & { X ⟶ m }) }}.
```

We are still not there yet. How do we derive the post-value?

```
Theorem hoare_asgn_fwd :
  forall m a P,
  {{ fun st ⇒ P st /\ st X = m}}
    X ::= a
  {{ fun st ⇒ P (st & { X ⟶ m }) /\ st X = aeval  (st & { X ⟶ m }) a }}.
```

This would be a very difficult theorem to apply. Can we do better?

Rephrasing the assignment rule

Recall that

```
Goal forall P m a,
  {{ fun st ⇒ P st }} X ::= a {{ fun st ⇒ P st }}.
```

lead us here

```
H0 : P st
______________________________________(1/1)
P (st & {X ⟶ aeval st a})
```

> What if we update the store in the pre-condition?

Rephrasing the pre-condition

```
Goal forall P m a,
  {{ fun st ⇒ P (st & { X ⟶ 3 }) }} X ::= a {{ fun st ⇒ P st }}.
```

leads us here

```
H0 : P (st & {X ⟶ 3})
______________________________________(1/1)
P (st & {X ⟶ aeval st a})
```

> Why not just set the pre-condition to `P (st & { X ⟶ aeval st a })`?

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. 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