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 \texttt{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:

\[
\text{Definition } \text{Assertion} ::= \text{state} \rightarrow \text{Prop}.
\]
Specifying Imp programs

Example assertions

1. $\{ x = 3 \}$ written as fun st $\Rightarrow$ st $X = 3$
Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as $\text{fun st } \Rightarrow \text{st } X = 3$

2. $\{x \leq y\}$ written as $\text{fun st } \Rightarrow \text{st } X \leq \text{st } Y$
Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as $\text{fun } st \Rightarrow st \ X = 3$
2. $\{x \leq y\}$ written as $\text{fun } st \Rightarrow st \ X \leq st \ Y$
3. $\{x = 3 \lor x \leq y\}$ written as $\text{fun } st \Rightarrow st \ X = 3 \lor st \ X \leq st \ Y$
Specifying Imp programs

Example assertions

1. \{x = 3\} written as \(\text{fun } st \Rightarrow st \ X = 3\)
2. \{x \leq y\} written as \(\text{fun } st \Rightarrow st \ X \leq st \ Y\)
3. \{x = 3 \lor x \leq y\} written as \(\text{fun } st \Rightarrow st \ X = 3 \lor st \ X \leq st \ Y\)
4. \(z \times z \leq x \land \neg((z + 1) \times (z + 1) \leq x)\) written as
   \(\text{fun } st \Rightarrow st \ Z \times st \ Z \leq st \ X \lor \neg (((S (st \ Z)) \times (S (st \ Z))) \leq st \ X)\)
Specifying Imp programs

Example assertions

1. $\{x = 3\}$ written as $\text{fun st \Rightarrow st X = 3}$
2. $\{x \leq y\}$ written as $\text{fun st \Rightarrow st X \leq st Y}$
3. $\{x = 3 \lor x \leq y\}$ written as $\text{fun st \Rightarrow st X = 3 \lor/ st X \leq st Y}$
4. $z \times z \leq x \land \neg((z + 1) \times (z + 1) \leq x)$ written as
   $\text{fun st \Rightarrow st Z \times st Z \leq st X \lor \neg(((S\ (st\ Z)) \times (S\ (st\ Z))) \leq st\ X)}$
5. What about $\text{fun st \Rightarrow True}$?
Specifying Imp programs

Example assertions

1. $\{ x = 3 \}$ written as $\text{fun } st \Rightarrow st X = 3$
2. $\{ x \leq y \}$ written as $\text{fun } st \Rightarrow st X \leq st Y$
3. $\{ x = 3 \lor x \leq y \}$ written as $\text{fun } st \Rightarrow st X = 3 \lor st X \leq st Y$
4. $z \times z \leq x \land \neg((z + 1) \times (z + 1) \leq x)$ written as
   $\text{fun } st \Rightarrow st Z \times st Z \leq st X \lor \sim (((S (st Z)) \times (S (st Z))) \leq st X)$
5. What about $\text{fun } st \Rightarrow True$?
6. What about $\text{fun } st \Rightarrow False$?
A Hoare Triple

Combining assertions with commands

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

Definition hoare_triple (P:Assertion) (c:com) (Q:Assertion) : Prop :=

\[
\forall \ st \ st', \quad
P \ st \rightarrow \quad \text{(* If } [P \ st] \text{ holds *)} \\
\text{ceval } st \ c \ st' \rightarrow \quad \text{(* And } [c] \text{ runs with an input state } [st] \text{ yielding a state } [st'] \text{ *)} \\
Q \ st'. \quad \text{(* Then } [Q \ st'] \text{ holds *)}
\]
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\} \text{Provable}
Exercise

Which of these programs are provable?

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

Which of these programs are provable?

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

5. \(\{x = 1\} \ \text{while} \ x \neq 0 \ \text{do} \ x := x + 1 \ \text{end} \ \{x = 100\}\)
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. $\{\top\}$ skip $\{\bot\}$ Improvable, because the conclusion is not provable.
5. $\{x = 1\}$ while $x \neq 0$ do $x := x + 1$ 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. \(\{\top\} \ \text{skip} \ \{\bot\}\) Improvable, because the conclusion is not provable.
5. \(\{x = 1\} \ \text{while} \ x \neq 0 \ \text{do} \ x := x + 1 \ \text{end} \ \{x = 100\}\) Provable, because the loop is not provable, so we can reach a contradiction.
6. \(\{x = 1\} \ \text{skip} \ \{x \geq 1\}\)
Exercise

Which of these programs are provable?

1. \( \{ \top \} \ x := 5; y := 0 \ \{ x = 5 \} \) \textbf{Provable}
2. \( \{ x = 2 \land x = 3 \} \ x := 5 \ \{ x = 0 \} \) \textbf{Provable, because the pre-condition is false}
3. \( \{ \top \} \ x := x + 1 \ \{ x = 2 \} \) \textbf{Improvable, because there's not enough information to assume} \( x = 1 \)
4. \( \{ \top \} \ \text{skip} \ \{ \bot \} \) \textbf{Improvable, because the conclusion is not provable.}
5. \( \{ x = 1 \} \ \text{while} \ x \neq 0 \ \text{do} \ x := x + 1 \ \text{end} \ \{ x = 100 \} \) \textbf{Provable, because the loop is not provable, so we can reach a contradiction.}
6. \( \{ x = 1 \} \ \text{skip} \ \{ x \geq 1 \} \) \textbf{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
Theorem (H-seq): If \( \{P\} c_1 \{Q\} \) and \( \{Q\} c_2 \{R\} \), then \( \{P\} c_1; c_2 \{R\} \).

\[
\text{Theorem } \text{hoare_seq : forall } P \ Q \ R \ c_1 \ c_2, \\
\{\{P\}\} \ c_1 \ \{\{Q\}\} \rightarrow \\
\{\{Q\}\} \ c_2 \ \{\{R\}\} \rightarrow \\
\{\{P\}\} \ c_1; c_2 \ \{\{R\}\}.
\]
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**
\[
\forall P \ a, \ \{\{ \ \text{fun} \ st \Rightarrow P \ st \} \} \ X := a \ \{\{ \ \text{fun} \ st \Rightarrow P \ st \ \land \ False \} \}.
\]

**Proof.**
- intros.
- intros s_in s_out Ha Hb.
- invc Ha.

Yields

\[
Hb : P \ s_\text{in} \ \Rightarrow \ aeval \ s_\text{in} \ a ; \ s_\text{in} \ \land \ False
\]

\(1/1\)
Deriving the rule for the assignment

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

Hb : P s_in

\[ P (X \rightarrow \text{aeval s_in a; s_in}) \land \text{False} \]
Deriving the rule for assignment

The following result should be provable.

Goal for all $P, a,$

\[
\{\{ \text{fun } st \Rightarrow P\ st \land st\ X = \text{aeval}\ st\ a \}\}\ns\ X := a
\{\{ \text{fun } st \Rightarrow P\ st \}\}.\]
Deriving the rule for assignment

The following result should be provable.

**Goal**
\[
\forall P, \ a, \\
\{\{ \text{fun } st \Rightarrow P st \land st X = \text{aeval } st \ a \}\} \\
X := a \\
\{\{ \text{fun } st \Rightarrow P st \}\}.
\]

**Proof.**
- intros.
- intros s_in s_out Ha [Hb Hc].
- invc Ha.
- rewrite \(\Leftarrow\) Hc.
- rewrite t_update_same.
- assumption.

Qed.
Deriving the rule for assignment

Making the code read more like the paper

\[
\{\{ \text{fun } \text{st } \Rightarrow \text{P} \text{ st } /\ \text{st } X = \text{aeval} \text{ st } a \}\} \ X := a \ \{\{ \text{fun } \text{st } \Rightarrow \text{P} \text{ st } \}\} \\
\]

becomes

\[
\{\{ \text{P } [X \mapsto a] \}\} \ X := a \ \{\{ \text{P} \}\} \\
\]
Abstracting a state update with evaluation

Another level of indirection

Read $P \left[ X \mapsto a \right]$ as:

- assertion $P$ where $X$ is assigned to the value of expression $a$

**Definition**

\[
\text{assn_sub } X \ a \ (P:Assertion) : Assertion := \\
\quad \text{fun (st : state) } \Rightarrow \\
\quad \quad P \ (X \mapsto \text{aeval st a ; st}).
\]

**Notation**

$"P \left[ X \mapsto a \right]" := (\text{assn_sub } X \ a \ P)$

(at level 10, $X$ at next level, a custom com).
Understanding the notation

\( (X \leq 5) [X \mapsto 3] \)

\[
\begin{align*}
P &= (\text{fun } st' \Rightarrow st' X \leq 5) \\
&= P \ [\ X \mapsto 3 \ ] \\
&= \text{assn_sub} \ X \ 3 \ P \\
&= \text{fun } st \Rightarrow \\
&\quad P (X \mapsto \text{aeval st 3}; st) \\
&= \text{fun } st \Rightarrow \\
&\quad P (X \mapsto 3; st) \\
&= \text{fun } st \Rightarrow \\
&\quad (\text{fun } st' \Rightarrow 0 \leftarrow st' X \leq 5) (X \mapsto 3; st) \\
&= \text{fun } st \Rightarrow \\
&\quad (X \mapsto 3; st) X \leq 5 \\
&= \text{fun } st \Rightarrow \\
&\quad 3 \leq 5
\end{align*}
\]
Backward style assignment rule

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

**Theorem (hoare_asgn):**
\[
\forall a \ P, \\
\{ \text{fun } st \Rightarrow P \ (st ; \{ X \mapsto \text{aeval } st \ a \}) \} \\
X := a \\
\{ \text{fun } st \Rightarrow P \ st \}. \\
\]
Exercise

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

Goal \( \{\ \{ \text{(fun st : state} \Rightarrow \text{st} \ X = 2\} \ [X \mapsto X + 1] \ [X \mapsto 1] \}\} \)

\[ X := 1; X := X + 1 \]

\( \{\ \{ \text{fun st} \Rightarrow \text{st} \ X = 2 \}\}. \)
Exercise

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

**Goal**

\[
\{ \text{(fun st : state} \Rightarrow \text{st } X = 2) \ [X \mapsto X + 1] \ [X \mapsto 1] \} \\
X := 1; X := X + 1 \\
\{ \text{fun st} \Rightarrow \text{st } X = 2 \} .
\]

**Yes.**
Exercise

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

Goal \( \{\{ \text{fun st : state } \Rightarrow \ st \ X = 2 \} \ [X \ |\rightarrow X + 1] \ [ X \ |\rightarrow 1 \} \} \)
\( X := 1; X := X + 1 \)
\( \{\{ \text{fun st } \Rightarrow \ st \ X = 2 \} \} \).

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

Goal \( \{\{ \text{fun st } \Rightarrow \ True \} \} \)
\( X := 1; X := X + 1 \)
\( \{\{ \text{fun st } \Rightarrow \ st \ X = 2 \} \} \).
Exercise

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

**Goal** \( \{ \) (fun \( st : \text{state} \Rightarrow \text{st} \ X = 2 \) [X \( \mapsto \) X + 1] [ X \( \mapsto \) 1] )\( \} \)
\( X := 1; X := X + 1 \)
\( \{ \) fun \( st \Rightarrow \text{st} \ X = 2 \) \( \} \).

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

**Goal** \( \{ \) fun \( st \Rightarrow \text{True} \) \( \} \) \( X := 1; X := X + 1 \)
\( \{ \) fun \( st \Rightarrow \text{st} \ X = 2 \) \( \} \).

No. The pre-condition has to match what we stated H-assign. 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:

\[
\{P\} \text{ SKIP } \{P\} \quad (\text{H-skip})
\]

\[
\frac{\{P\} \; c_1 \; \{Q\} \quad \{Q\} \; c_2 \; \{R\}}{\{P\} \; c_1 \; ; \; c_2 \; \{R\}} \quad (\text{H-seq})
\]

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