Hoare Logic and Model Checking

Kasper Svendsen
University of Cambridge

CST Part II – 2016/17

Acknowledgement: slides heavily based on previous versions by Mike Gordon and Alan Mycroft

Semantics of Hoare Logic

Semantics of Hoare Logic

Recall, to define a Hoare Logic we need three main components:

- the programming language that we want to reason about, along with its operational semantics
- an assertion language for defining state predicates, along with a semantics
- a formal interpretation of Hoare triples, together with a (sound) formal proof system for deriving Hoare triples

This lecture will define defines a formal semantics of Hoare Logic and introduces some meta-theoretic results about Hoare Logic (soundness & completeness).

Operational semantics for WHILE

Operational semantics of WHILE

The operational semantics of WHILE will be defined as a transition system that consists of

- a set of stores, *stores*, and
- a reduction relation, $\Downarrow \in \mathcal{P}(\mathit{Cmd} \times \mathit{Store} \times \mathit{Store})$.

The reduction relation, written $\langle C, s\rangle \Downarrow s'$, expresses that the command C reduces to the terminal state s' when executed from initial state s.

Operational semantics of WHILE

Stores are functions from variables to integers:

$$\mathit{Store} \stackrel{\mathit{def}}{=} \mathit{Var} \to \mathbb{Z}$$

These are **total** functions and define the current value of every program and auxiliary variable.

This models WHILE with arbitrary precision integer arithmetic. A more realistic model might use 32-bit integers are require reasoning about overflow, etc.

Operational semantics of WHILE

The reduction relation is defined inductively by a set of rules.

To reduce an assignment we first evaluate the expression E using the current store and update the store with the value of E.

$$\frac{\mathcal{E}[\![E]\!](s) = n}{\langle X := E, s\rangle \Downarrow s[X \mapsto n]}$$

We use functions $\mathcal{E}[\![E]\!](s)$ and $\mathcal{B}[\![B]\!](s)$ to evaluate expressions and boolean expressions in a given store s.

Semantics of expressions

$\mathcal{E}[\![E]\!](s)$ evaluates expression E to an integer in store s:

$$\mathcal{E}[\![-]\!](=) : \mathit{Exp} \times \mathit{Store} \to \mathbb{Z}$$

$$\begin{aligned}
\mathcal{E}[\![N]\!](s) &= N \\
\mathcal{E}[\![V]\!](s) &= s(V) \\
\mathcal{E}[\![E_1 + E_2]\!](s) &= \mathcal{E}[\![E_1]\!](s) + \mathcal{E}[\![E_2]\!](s) \\
&\vdots
\end{aligned}$$

This semantics is too simple to handle operations such as division, which fails to evaluate to an integer on some inputs.

Semantics of boolean expressions

$\mathcal{B}[\![B]\!](s)$ evaluates boolean expression B to an boolean in store s:

$$\mathcal{B}[\![-]\!](=) : \mathit{BExp} \times \mathit{Store} \to \mathbb{B}$$

$$\mathcal{E}[\![T]\!](s) = \top$$

$$\mathcal{E}[\![F]\!](s) = \bot$$

$$\mathcal{E}[\![E_1 \leq E_2]\!](s) = \begin{cases} \top & \text{if } \mathcal{E}[\![E_1]\!](s) \leq \mathcal{E}[\![E_2]\!](s) \\ \bot & \text{otherwise} \end{cases}$$

$$\vdots$$

Operational semantics of WHILE

$$\frac{\mathcal{E}[\![E]\!](s) = n}{\langle X := E, s\rangle \Downarrow s[X \mapsto n]} \qquad \frac{\langle C_1, s\rangle \Downarrow s' \qquad \langle C_2, s'\rangle \Downarrow s''}{\langle C_1; C_2, s\rangle \Downarrow s''}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \top \qquad \langle C_1, s\rangle \Downarrow s'}{\langle \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2, s\rangle \Downarrow s'} \qquad \frac{\mathcal{B}[\![B]\!](s) = \bot \qquad \langle C_2, s\rangle \Downarrow s'}{\langle \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2, s\rangle \Downarrow s'}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \top \qquad \langle C, s\rangle \Downarrow s' \qquad \langle \textbf{while } B \textbf{ do } C, s'\rangle \Downarrow s''}{\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow s''}$$

$$\frac{\mathcal{B}[\![B]\!](s) = \bot}{\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow s} \qquad \frac{}{\langle \textbf{skip}, s\rangle \Downarrow s}$$

Meta-theory

Note that the operational semantics of WHILE is deterministic:

$$\langle C, s\rangle \Downarrow s' \wedge \langle C, s\rangle \Downarrow s'' \Rightarrow s' = s''$$

We have already implicitly used this in the definition of total correctness triples.

Without this property, we would have to specify whether all reductions or just some reductions were required to terminate.

Meta-theory

We will need the following expression substitution property later to prove soundness of the Hoare assignment axiom:

$$\mathcal{E}[\![E_1[E_2/V]]\!](s) = \mathcal{E}[\![E_1]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])$$

The expression substitution property follows by induction on E_1.

Case $E_1 \equiv N$:

$$\mathcal{E}[\![N[E_2/V]]\!](s) = N = \mathcal{E}[\![N]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])$$

Meta-theory

$$\mathcal{E}[\![E_1[E_2/V]]\!](s) = \mathcal{E}[\![E_1]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])$$

Case $E_1 \equiv V'$:

$$\begin{aligned}\mathcal{E}[\![V'[E_2/V]]\!](s) &= \begin{cases}\mathcal{E}[\![E_2]\!](s) & \text{if } V = V' \\ s(V') & \text{if } V \neq V'\end{cases} \\ &= \mathcal{E}[\![V']\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])\end{aligned}$$

Meta-theory

$$\mathcal{E}[\![E_1[E_2/V]]\!](s) = \mathcal{E}[\![E_1]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])$$

Case $E_1 \equiv E_a + E_b$:

$$\begin{aligned}
&\mathcal{E}[\![(E_a + E_b)[E_2/V]]\!](s) \\
&\quad = \mathcal{E}[\![E_a[E_2/V]]\!](s) + \mathcal{E}[\![E_b[E_2/V]]\!](s) \\
&\quad = \mathcal{E}[\![E_a]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)]) + \mathcal{E}[\![E_b]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)]) \\
&\quad = \mathcal{E}[\![E_a + E_b]\!](s[V \mapsto \mathcal{E}[\![E_2]\!](s)])
\end{aligned}$$

Semantics of assertions

The language of assertions

Now we have formally defined the semantics of the WHILE language that we wish to reason about.

The next step is to formalise the assertion language that we will use to reason about states of WHILE programs.

We take the language of assertions to be an instance of (single-sorted) first-order logic with equality.

Knowledge of first-order logic is assumed. We will review some basic concepts now.

Review of first-order logic

Recall that in first-order logic there are two syntactic classes:

- Terms: which denote values (e.g., numbers)
- Assertions: describe properties that may be true or false

Assertions are built out of terms, predicates and logical connectives ($\wedge$, $\vee$, etc.).

Since we are reasoning about WHILE states, our assertions will describe properties of WHILE states.

Review of first-order logic: Terms

Terms may contain variables like x, X, y, X, z, Z etc.

Terms, like 1 and $4 + 5$, that do not contain any free variables are called ground terms.

We use conventional notation, e.g. here are some terms:

$$X, \quad y, \quad Z,$$

$$1, \quad 2, \quad 325,$$

$$-X, \quad -(X+1), \quad (x \cdot y) + Z,$$

$$\sqrt{(1+x^2)}, \quad X!, \quad sin(x), \quad rem(X, Y)$$

Review of first-order logic: Atomic assertions

Examples of atomic assertions are:

$$\bot, \qquad \top, \qquad X = 1, \qquad R < Y, \qquad X = R + (Y \cdot Q)$$

$\top$ and $\bot$ are atomic assertions that are always true and false.

Other atomic assertions are built from terms using predicates, e.g.

$$\mathit{ODD}(X), \qquad \mathit{PRIME}(3), \qquad X = 1, \qquad (X + 1)^2 \geq x^2$$

Here ODD, PRIME, and $\geq$ are examples of predicates ($\geq$ is written using infix notation) and X, 1, 3, $X + 1$, $(X + 1)^2$ and x^2 are terms in above atomic assertions.

Review of first-order logic: Atomic assertions

In general, first-order logic is parameterised over a signature that defines non-logical function symbols ($+, -, \cdot$, ...) and predicate symbols (*ODD*, *PRIME*, etc.).

We will be using a particular instance with a signature that includes the usual functions and predicates on integers.

Review of first-order logic: Compound assertions

Compound assertions are built up from atomic assertions using the usual logical connectives:

$$\wedge \text{ (}\textit{conjunction}\text{)}, \vee \text{ (}\textit{disjunction}\text{)}, \Rightarrow \text{ (}\textit{implication}\text{)}$$

and quantification:

$$\forall \text{ (}\textit{universal}\text{)}, \exists \text{ (}\textit{existential}\text{)}$$

Negation, $\neg P$, is a shorthand for $P \Rightarrow \bot$.

The assertion language

The formal syntax of the assertion language is given below.

$$
\begin{array}{rcll}
P, Q & ::= & \bot \mid \top \mid B \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q & \textit{assertions} \\
& \mid & \forall x.\, P \mid \exists x.\, P \mid t_1 = t_2 \mid p(t_1, ..., t_n) & \\
\\
t & ::= & E \mid f(t_1, ..., t_n) & \textit{terms}
\end{array}
$$

Note that assertions quantify over logical variables.

Here p and f range over an unspecified set of predicates and functions, respectively, that includes the usual mathematical operations on integers.

Semantics of terms

$[\![t]\!]$ defines the meaning of a term t.

$$[\![-]\!](=) : \mathit{Term} \times \mathit{Store} \to \mathbb{Z}$$

$$[\![E]\!](s) \stackrel{def}{=} \mathcal{E}[\![E]\!](s)$$

$$[\![f(t_1, ..., t_n)]\!](s) \stackrel{def}{=} [\![f]\!]([\![t_1]\!](s), ..., [\![t_n]\!](s))$$

We assume $[\![f]\!]$ is given by the implicit signature.

Semantics of assertions

$[\![P]\!]$ defines the set of stores that satisfy the assertion P.

$$[\![-]\!] : \mathit{Assertion} \to \mathcal{P}(\mathit{Store})$$

$$[\![\bot]\!] = \emptyset$$

$$[\![\top]\!] = \mathit{Store}$$

$$[\![B]\!] = \{s \mid \mathcal{B}[\![B]\!](s) = \top\}$$

$$[\![P \vee Q]\!] = [\![P]\!] \cup [\![Q]\!]$$

$$[\![P \wedge Q]\!] = [\![P]\!] \cap [\![Q]\!]$$

$$[\![P \Rightarrow Q]\!] = \{s \mid s \in [\![P]\!] \Rightarrow s \in [\![Q]\!]\}$$

Semantics of assertions (continued)

$$[\![\forall x.\, P]\!] = \{s \mid \forall v.\, s[x \mapsto v] \in [\![P]\!]\}$$
$$[\![\exists x.\, P]\!] = \{s \mid \exists v.\, s[x \mapsto v] \in [\![P]\!]\}$$
$$[\![t_1 = t_2]\!] = \{s \mid [\![t_1]\!](s) = [\![t_2]\!](s)\}$$
$$[\![p(t_1, ..., t_n)]\!] = \{s \mid [\![p]\!]([\![t_1]\!](s), ..., [\![t_2]\!](s))\}$$

We assume $[\![p]\!]$ is given by the implicit signature.

This interpretation is related to the forcing relation you used in "Proof and Logic":

$$s \in [\![P]\!] \Leftrightarrow s \models P$$

Substitutions

We use $t[E/V]$ and $P[E/V]$ to denote t and P with E substituted for every occurrence of program variable V, respectively.

Since our quantifiers bind logical variables and all free variables in E are program variables, there is no issue with variable capture.

Substitution property

The term and assertion semantics satisfy a similar substitution property to the expression semantics:

- $[\![t[E/V]]\!](s) = [\![t]\!](s[V \mapsto \mathcal{E}[\![E]\!](s)])$
- $s \in [\![P[E/V]]\!] \Leftrightarrow s[V \mapsto \mathcal{E}[\![E]\!](s)] \in [\![P]\!]$

They are easily provable by induction on t and P, respectively. (Exercise)

Semantics of Hoare Logic

Semantics of partial correctness triples

Now that we have formally defined the operational semantics of WHILE and our assertion language, we can define the formal meaning of our triples.

Partial correctness triples assert that if the given command terminates when executed from an initial state that satisfies the precondition than the terminal state must satisfy the postcondition:

$$\models \{P\}\ C\ \{Q\} \stackrel{def}{=} \forall s, s'.\, s \in [\![P]\!] \wedge \langle C, s\rangle \Downarrow s' \Rightarrow s' \in [\![Q]\!]$$

Semantics of total correctness triples

Total correctness triples assert that when the given command is executed from an initial state that satisfies the precondition, then it must terminate in a terminal state that satisfies the postcondition:

$$\models [P]\ C\ [Q] \stackrel{def}{=} \forall s.\, s \in [\![P]\!] \Rightarrow \exists s'.\, \langle C, s\rangle \Downarrow s' \wedge s' \in [\![Q]\!]$$

Since WHILE is deterministic, if one terminating execution satisfies the postcondition then all terminating executions satisfy the postcondition.

Meta-theory of Hoare Logic

Now we have a syntactic proof system for deriving Hoare triples, $\vdash \{P\}\ C\ \{Q\}$, and a formal definition of the meaning of our Hoare triples, $\models \{P\}\ C\ \{Q\}$.

How are these related?

We might hope that any triple that can be derived syntactically holds semantically (soundness) and that any triple that holds semantically is syntactically derivable (completeness).

This is **not** the case: Hoare Logic is sound but not complete.

Soundness of Hoare Logic

Theorem (Soundness)
If $\vdash \{P\}\ C\ \{Q\}$ *then* $\models \{P\}\ C\ \{Q\}$.

Soundness expresses that any triple derivable using the syntactic proof system holds semantically.

Soundness is proven by induction on the $\vdash \{P\}\ C\ \{Q\}$ derivation:

- we have to show that all Hoare axioms hold semantically, and
- for each inference rule, that if each hypothesis holds semantically, then the conclusion holds semantically

Soundness of the assignment axiom

$$\models \{P[E/V]\}\ V := E\ \{P\}$$

Assume $s \in [\![P[E/V]]\!]$ and $\langle V := E, s\rangle \Downarrow s'$.

From the substitution property it follows that

$$s[V \mapsto \mathcal{E}[\![E]\!](s)] \in [\![P]\!]$$

and from the reduction relation it follows that
$s' = s[V \mapsto \mathcal{E}[\![E]\!](s)]$. Hence, $s' \in [\![P]\!]$.

Soundness of the loop inference rule

If $\models \{P \wedge B\}\ C\ \{P\}$ then $\models \{P\}$ **while** B **do** $C\ \{P \wedge \neg B\}$

Assume $\models \{P \wedge B\}\ C\ \{P\}$.

We will prove $\models \{P\}$ **while** B **do** $C\ \{P \wedge \neg B\}$ by proving the following stronger property by induction on n:

$$\forall n.\, \forall s, s'.\, s \in [\![P]\!] \wedge \langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^n s' \Rightarrow s' \in [\![P \wedge \neg B]\!]$$

Here $\langle C, s\rangle \Downarrow^n s'$ indicates a reduction in n steps.

Soundness of the loop inference rule

Case $n = 1$: assume $s \in [\![P]\!]$ and $\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^1 s'$. Since the loop reduced in one step, B must have evaluated to false: $\mathcal{B}[\![B]\!](s) = \bot$ and $s' = s$. Hence, $s' = s \in [\![P \wedge \neg B]\!]$.

Case $n > 1$: assume $s \in [\![P]\!]$ and $\langle \textbf{while } B \textbf{ do } C, s\rangle \Downarrow^n s'$. Since the loop reduced in more than one step, B must have evaluated to true: $\mathcal{B}[\![B]\!](s) = \top$ and there exists an s'', n_1 and n_2 such that $\langle C, s\rangle \Downarrow^{n_1} s''$, $\langle \textbf{while } B \textbf{ do } C, s''\rangle \Downarrow^{n_2} s'$ with $n = n_1 + n_2 + 1$.

From the $\models \{P \wedge B\}\ C\ \{P\}$ assumption it follows that $s'' \in [\![P]\!]$ and by the induction hypothesis, $s' \in [\![P \wedge \neg B]\!]$.

Completeness

Completeness is the converse property of soundness:
If $\models \{P\}\ C\ \{Q\}$ then $\vdash \{P\}\ C\ \{Q\}$.

Hoare Logic inherits the incompleteness of first-order logic and is therefore **not** complete.

Completeness

To see why, consider the triple $\{T\}$ **skip** $\{P\}$.

By unfolding the meaning of this triple, we get:

$$\models \{T\} \textbf{ skip } \{P\} \Leftrightarrow \forall s.\, s \in [\![P]\!]$$

If could deduce any true triple using Hoare Logic, we would be able to deduce any true statement of the assertion logic using Hoare Logic.

Since the assertion logic (first-order logic) is **not** complete this is not the case.

Relative completeness

The previous argument showed that because the assertion logic is not complete, then neither is Hoare Logic.

However, Hoare logic is **relatively complete** for our simple language:

- Relative completeness expresses that any failure to prove $\vdash \{P\}\ C\ \{Q\}$, for a valid statement $\models \{P\}\ C\ \{Q\}$, can be traced back to a failure to prove $\vdash \phi$ for some valid arithmetic statement ϕ.

Decidability

Finally, Hoare logic is not decidable.

The triple $\{T\}\ C\ \{F\}$ holds if and only if C does not terminate.
Hence, since the Halting problem is undecidable so is Hoare Logic.

Summary

We have defined an operational semantics for the WHILE language and a formal semantics for Hoare logic for WHILE.

We have shown that the formal Hoare logic proof system from the last lecture is sound with respect to this semantics, but not complete.

Supplementary reading on soundness and completeness:

- Glynn Winskel. The Formal Semantics of Programming Languages: An Introduction. Chapters 6–7.