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
This course is about **formal** techniques for validating software.

Formal methods allow us to **formally specify** the intended behaviour of our programs and use mathematical proof systems to **formally prove** that our programs satisfy their specification.

In this course we will focus on two techniques:

- **Hoare logic** (Lectures 1-6)
- **Model checking** (Lectures 7-12)
There are many different formal reasoning techniques of varying expressivity and level of automation.
Testing can quickly find obvious bugs:

- only trivial programs can be tested exhaustively
- the cases you do not test can still hide bugs
- coverage tools can help

Formal methods can improve assurance:

- allows us to reason about all possible executions
- can reveal hard-to-find bugs
Famous software bugs

At least 3 people were killed due to massive radiation overdoses delivered by a Therac-25 radiation therapy machine.

- the cause was a race-condition in the control software

An unmanned Ariane 5 rocket blew up on its maiden flight; the rocket and its cargo were estimated to be worth $500M.

- the cause was an unsafe floating point to integer conversion
However, formal methods are not a panacea:

- formally verified designs may still not work
- can give a false sense of security
- formal verification can be very expensive and time-consuming

Formal methods should be used in conjunction with testing, not as a replacement.
Lecture plan

Lecture 1: Informal introduction to Hoare logic
Lecture 2: Formal semantics of Hoare logic
Lecture 3: Examples, loop invariants & total correctness
Lecture 4: Mechanised program verification
Lecture 5: Separation logic
Lecture 6: Examples in separation logic
Hoare logic
Hoare logic is a formalism for relating the initial and terminal state of a program.

Hoare logic was invented in 1969 by Tony Hoare, inspired by earlier work of Robert Floyd.

Hoare logic is still an active area of research.
Hoare logic uses **partial correctness triples** for specifying and reasoning about the behaviour of programs:

\[
\{P\} \ C \ \{Q\}
\]

Here \( C \) is a command and \( P \) and \( Q \) are state predicates.

- \( P \) is called the precondition and describes the initial state
- \( Q \) is called the postcondition and describes the terminal state
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 introduce each component informally. In the coming lectures we will cover the formal details.
The WHILE language
WHILE is a prototypical imperative language. Programs consists of commands, which include branching, iteration and assignments:

\[
C ::= \text{skip} \mid C_1; C_2 \mid V := E \\
    \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid \text{while } B \text{ do } C
\]

Here \( E \) is an expression which evaluates to a natural number and \( B \) is a boolean expression, which evaluates to a boolean.

States are mappings from variables to natural numbers.
The grammar for expressions and boolean includes the usual arithmetic operations and comparison operators:

\[
E ::= N \mid V \mid E_1 + E_2 \mid \text{expressions} \\
| E_1 - E_2 \mid E_1 \times E_2 \mid \cdots
\]

\[
B ::= T \mid F \mid E_1 = E_2 \mid \text{boolean expressions} \\
| E_1 \leq E_2 \mid E_1 \geq E_2 \mid \cdots
\]

Note that expressions do not have side effects.
The assertion language
State predicates $P$ and $Q$ can refer to program variables from $C$ and will be written using standard mathematical notations together with \textbf{logical operators} like:

- $\land$ ("and"), $\lor$ ("or"), $\neg$ ("not") and $\Rightarrow$ ("implies")

For instance, the predicate $X = Y + 1 \land Y > 0$ describes states in which the variable $Y$ contains a positive value and the value of $X$ is equal to the value of $Y$ plus 1.
Partial correctness triples

The partial correctness triple $\{ P \} \ C \ {\{ Q \}}$ holds if and only if:

- whenever $C$ is executed in an initial state satisfying $P$
- and this execution terminates
- then the terminal state of the execution satisfies $Q$.

For instance,

- $\{ X = 1 \} \ X := X + 1 \ {\{ X = 2 \}}$ holds
- $\{ X = 1 \} \ X := X + 1 \ {\{ X = 3 \}}$ does not hold
Partial correctness triples are called **partial** because they only specify the intended behaviour of terminating executions.

For instance, \( \{X = 1\} \textbf{while } X > 0 \textbf{ do } X := X + 1 \{X = 0\} \) holds, because the given program never terminates when executed from an initial state where \( X \) is 1.

Hoare logic also features total correctness triples that strengthen the specification to require termination.
The total correctness triple $[P] \ C \ [Q]$ holds if and only if:

- whenever $C$ is executed in an initial state satisfying $P$
- then the execution must terminate
- and the terminal state must satisfy $Q$.

There is no standard notation for total correctness triples, but we will use $[P] \ C \ [Q]$. 
Total correctness

The following total correctness triple does not hold:

\[ [X = 1] \text{ while } X > 0 \text{ do } X := X + 1 [X = 0] \]

• the loop never terminates when executed from an initial state where \( X \) is positive

The following total correctness triple does hold:

\[ [X = 0] \text{ while } X > 0 \text{ do } X := X + 1 [X = 0] \]

• the loop always terminates immediately when executed from an initial state where \( X \) is zero
Informally: total correctness = termination + partial correctness.

It is often easier to show partial correctness and termination separately.

Termination is usually straightforward to show, but there are examples where it is not: no one knows whether the program below terminates for all values of $X$

\[
\text{while } X > 1 \text{ do } \\
\quad \text{if } \text{ODD}(X) \text{ then } X := 3 \times X + 1 \text{ else } X := X \text{ DIV } 2
\]

Microsoft’s T2 tool proves systems code terminates.
Specifications
Simple examples

\{\bot\} \ C \ \{Q\}

- this says nothing about the behaviour of \(C\), because \(\bot\) never holds for any initial state

\{\top\} \ C \ \{Q\}

- this says that whenever \(C\) halts, \(Q\) holds

\{P\} \ C \ \{T\}

- this holds for every precondition \(P\) and command \(C\), because \(T\) always holds in the terminate state
Simple examples

\[ [P] \ C \ [T] \]

- this says that \( C \) always terminates when executed from an initial state satisfying \( P \)

\[ [T] \ C \ [Q] \]

- this says that \( C \) always terminates in a state where \( Q \) holds
Consider a program $C$ that computes the maximum value of two variables $X$ and $Y$ and stores the result in a variable $Z$.

Is this a good specification for $C$?

\[
\{ \top \} C \{(X \leq Y \Rightarrow Z = Y) \land (Y \leq X \Rightarrow Z = X)\}
\]

No! Take $C$ to be $X := 0; Y := 0; Z := 0$, then $C$ satisfies the above specification. The postcondition should refer to the initial values of $X$ and $Y$.

In Hoare logic we use **auxiliary variables** which do not occur in the program to refer to the initial value of variables in postconditions.
For instance, \( \{X = x \land Y = y\} C \{X = y \land Y = x\} \), expresses that if \( C \) terminates then it exchanges the values of variables \( X \) and \( Y \).

Here \( x \) and \( y \) are auxiliary variables (or ghost variables) which are not allowed to occur in \( C \) and are only used to name the initial values of \( X \) and \( Y \).

Informal convention: program variables are uppercase and auxiliary variables are lowercase.
Formal proof system for Hoare logic
Hoare logic

We will now introduce a natural deduction proof system for partial correctness triples due to Tony Hoare.

The logic consists of a set of **axiom schemas** and **inference rule schemas** for deriving consequences from premises.

If $S$ is a statement of Hoare logic, we will write $\vdash S$ to mean that the statement $S$ is derivable.
The inference rules of Hoare logic will be specified as follows:

\[
\begin{align*}
\vdash S_1 & \quad \cdots \quad \vdash S_n \\
\hline
\vdash \ & \vdash S
\end{align*}
\]

This expresses that \( S \) may be deduced from assumptions \( S_1, \ldots, S_n \).

An axiom is an inference rule without any assumptions:

\[
\begin{align*}
\hline
\vdash S
\end{align*}
\]

In general these are schemas that may contain meta-variables.
A proof tree for $\vdash S$ in Hoare logic is a tree with $\vdash S$ at the root, constructed using the inference rules of Hoare logic with axioms at the leaves.

$\vdash S_1 \quad \vdash S_2$

$\vdash S_3 \quad \vdash S_4$

$\vdash S$

We typically write proof trees with the root at the bottom.
Formal proof system

\[ \vdash \{ P \} \textbf{skip} \{ P \} \]

\[ \vdash \{ P \} \textbf{C}_1 \{ Q \} \quad \vdash \{ Q \} \textbf{C}_2 \{ R \} \]

\[ \vdash \{ P \} \textbf{C}_1 ; \textbf{C}_2 \{ R \} \]

\[ \vdash \{ P \} \textbf{if} \ B \ \textbf{then} \ \textbf{C}_1 \ \textbf{else} \ \textbf{C}_2 \{ Q \} \]

\[ \vdash \{ P \} \textbf{while} \ B \ \textbf{do} \ \textbf{C} \{ P \} \]

\[ \vdash \{ P \} \textbf{while} \ B \ \textbf{do} \ \textbf{C} \{ P \wedge \neg B \} \]
\[\vdash P_1 \Rightarrow P_2 \quad \vdash \{P_2\} \mathcal{C} \{Q_2\} \quad \vdash Q_2 \Rightarrow Q_1\]

\[\vdash \{P_1\} \mathcal{C} \{Q_1\}\]

\[\vdash \{P_1\} \mathcal{C} \{Q\} \quad \vdash \{P_2\} \mathcal{C} \{Q\}\]

\[\vdash \{P_1 \lor P_2\} \mathcal{C} \{Q\}\]

\[\vdash \{P\} \mathcal{C} \{Q_1\} \quad \vdash \{P\} \mathcal{C} \{Q_2\}\]

\[\vdash \{P\} \mathcal{C} \{Q_1 \land Q_2\}\]
The skip rule

\[ \vdash \{ P \} \text{skip} \{ P \} \]

The skip axiom expresses that any assertion that holds before skip is executed also holds afterwards.

\( P \) is a meta-variable ranging over an arbitrary state predicate.

For instance, \( \vdash \{ X = 1 \} \text{skip} \{ X = 1 \} \).
The assignment rule

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

Here $P[E/V]$ means the assertion $P$ with the expression $E$ substituted for all occurrences of the variable $V$.

For instance,

$$\{X + 1 = 2\} \ X := X + 1 \ \{X = 2\}$$

$$\{Y + X = Y + 10\} \ X := Y + X \ \{X = Y + 10\}$$
This assignment axiom looks backwards! Why is it sound?

In the next lecture we will prove it sound, but for now, consider some plausible alternative assignment axioms:

\[
\{P\} \quad V := E \{P[E/V]\}
\]

We can instantiate this axiom to obtain the following triple which does not hold:

\[
\{X = 0\} \quad X := 1 \quad \{1 = 0\}
\]
The rule of consequence allows us to strengthen preconditions and weaken postconditions.

Note: the $\vdash P \Rightarrow Q$ hypotheses are a different kind of judgment.

For instance, from $\{X + 1 = 2\} \quad X := X + 1 \quad \{X = 2\}$
we can deduce $\{X = 1\} \quad X := X + 1 \quad \{X = 2\}$.
Sequential composition

\[
\vdash \{P\} C_1 \{Q\} \quad \vdash \{Q\} C_2 \{R\} \\
\vdash \{P\} C_1; C_2 \{R\}
\]

If the postcondition of \(C_1\) matches the precondition of \(C_2\), we can derive a specification for their sequential composition.

For example, if one has deduced:

- \(\{X = 1\} \ X := X + 1 \ \{X = 2\}\)
- \(\{X = 2\} \ X := X + 1 \ \{X = 3\}\)

we may deduce that \(\{X = 1\} \ X := X + 1; X := X + 1 \ \{X = 3\}\).
The conditional rule

\[
\frac{
\vdash \{ P \land B \} \ C_1 \ \{ Q \} \quad \vdash \{ P \land \neg B \} \ C_2 \ \{ Q \}
}{
\vdash \{ P \} \text{ if } B \text{ then } C_1 \text{ else } C_2 \ \{ Q \}
}
\]

For instance, to prove that

\[
\vdash \{ T \} \text{ if } X \geq Y \text{ then } Z := X \text{ else } Z := Y \ \{ Z = \max(X, Y) \}
\]

It suffices to prove that \( \vdash \{ T \land X \geq Y \} \ Z := X \ \{ Z = \max(X, Y) \} \)

and \( \vdash \{ T \land \neg(X \geq Y) \} \ Z := Y \ \{ Z = \max(X, Y) \} \).
The loop rule

\[
\vdash \{P \land B\} \ C \ \{P\} \\
\vdash \{P\} \ \textbf{while} \ B \ \textbf{do} \ C \ \{P \land \neg B\}
\]

The loop rule says that

- if \(P\) is an invariant of the loop body when the loop condition succeeds, then \(P\) is an invariant for the whole loop
- and if the loop terminates, then the loop condition failed

We will return to the problem of finding loop invariants.
Conjunction and disjunction rule

\[ \vdash \{P_1\} \quad C \quad \{Q\} \quad \vdash \{P_2\} \quad C \quad \{Q\} \]
\[ \vdash \{P_1 \lor P_2\} \quad C \quad \{Q\} \]
\[ \vdash \{P\} \quad C \quad \{Q_1\} \quad \vdash \{P\} \quad C \quad \{Q_2\} \]
\[ \vdash \{P\} \quad C \quad \{Q_1 \land Q_2\} \]

These rules are useful for splitting up proofs.

Any proof with these rules could be done without using them

- i.e. they are theoretically redundant (proof omitted)
- however, useful in practice
Hoare Logic is a formalism for reasoning about the behaviour of programs by relating their initial and terminal state.

It uses an assertion logic based on first-order logic to reason about program states and extends this with Hoare triples to reason about the programs.

Suggested reading: