C02 – Hoare Logic

Program Verification

FMI · Denisa Diaconescu · Spring 2025

Overview

What problem are we trying to solve?

Hoare Logic – The Calculus

What problem are we trying to solve?

Verification for Imperative Languages

- Imperative languages are built around a program state (data stored in memory).
- Imperative programs are sequences of commands that modify that state.
- To prove properties of imperative programs, we need
  1. A way of expressing assertions about program states.
  2. Rules for manipulating and proving those assertions.
- These will be provided by Hoare Logic.

Verification for programming languages

The formalisms we will see can be extended to verify

- Recursive Programs
- Object Oriented Programs
- Parallel Programs
- Distributed Programs

Tools

- Dafny – Microsoft
- Infer – Facebook
- VeriFast
- SmallFoot
- ...

C.A.R. (Tony) Hoare

Hoare Logic was introduced by Tony Hoare.

He also invented Quicksort algorithm in 1960 (when he was 26).

Hoare triples

A Hoare triple $\{P\}\ \mathbb{C}\ \{Q\}$ has three components:

P a precondition

$\mathbb{C}$ a code fragment

Q a postcondition

The precondition is an assertion saying something of interest about the state before the code is executed.

The postcondition is an assertion saying something of interest about the state after the code is executed.

Syntax

The precondition and postcondition will be built from program variables, numbers, basic arithmetics relations, and use propositional logic to combine simple assertions.

Example

- $x = 3$
- $x \neq y$
- $x > 0$
- $x = 4 \wedge y = 2$
- $(x > y) \rightarrow (x = 2 * y)$
- $\top$
- $\bot$

Semantics

A state is determined by the values given to the program variables.

In our little language all our variables will store numbers only!

Hoare Triple: $\{P\}\ \mathbb{C}\ \{Q\}$

- if the pre-state satisfies P
- and the program $\mathbb{C}$ terminates
- then the post-state satisfies Q

Partial Correctness

Hoare logic expresses partial correctness!

- A program is partially correct if it gives the right answer whenever it terminates.
- It never gives a wrong answer, but it may give no answer at all.

Example

$$\{x = 1\}\ \texttt{while x = 1 do y := 2}\ \{x = 3\}$$

is true in the Hoare logic semantics.

- if pre-state satisfies $x = 1$ and the while loop terminates then the post-state satisfies $x = 3$. But the while loop does not terminate!

Partial Correctness is OK

Why not insist on termination?

- It simplifies the logic.
- If necessary, we can prove termination separately.

We will come back to termination with the Weakest Precondition Calculus.

Dafny

- Created by Rustan Leino at Microsoft Research
- Is open source
- Supports formal specification through preconditions, postconditions, loop invariants.

- Proves that there are no runtime errors, such as index out of bounds, null dereferences, division by zero, etc
- Also proves the termination of code.

Dafny

- Programming language designed for reasoning
- Program verifier
- Language features drawn from:
  - Imperative programming
    `if, while, :=, class,...`
  - Functional programming
    `function, datatype,...`
  - Proof authoring
    `Lemma, calc, refines, inductive predicate,...`

Dafny

- Dafny's homepage
- The Github page which includes sources and binary downloads for Windows, Mac, GNU/Linux
- Dafny mode in Emacs
- Dafny IDE in VS Code

Hoare Logic – The Calculus

A simple imperative programming language

To prove things about programs, we first need to fix a programming language.

We will define a little language with four different kinds of statement.

- Assignment – $\texttt{x := } \mathbb{E}$
  - $\texttt{x}$ is a variable and $\mathbb{E}$ is an expression built from variables and arithmetics that returns a number, e.g., $\texttt{2+3, x*y+1}$, ...
- Sequencing – $\mathbb{C}_1 \texttt{ ; } \mathbb{C}_2$
- Conditional – $\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}_1 \texttt{ else } \mathbb{C}_2$
  - $\mathbb{B}$ is an expression built from variables, arithmetics, and logic that returns a boolean value, e.g., $\texttt{y < 0, x} \neq \texttt{y} \wedge \texttt{z} = 0$, ...
- While – $\texttt{while } \mathbb{B} \texttt{ do } \mathbb{C}$

The Calculus

Example

Hoare logic will allow us to make claims such as:

$$\{x > 0\}\ \texttt{y := 0 - x}\ \{y < 0 \wedge x \neq y\}$$

If $(x > 0)$ is true before `y := 0 - x` is executed then $(y < 0 \wedge x \neq y)$ is true afterwards.

This particular reasoning is intuitively true. How to prove this?

We need a calculus – a collection of rules for (formally) manipulating the triples. We will have one rule for each of our four kinds of statement (plus two other rules).

The Assignment Axiom (Rule 1/6)

Assignments change the state so we expect Hoare triples for assignments to reflect that change.

The assignment axiom:

$$\boxed{\{Q[x/\mathbb{E}]\}\ \mathtt{x\ :=\ }\mathbb{E}\ \{Q\}}$$

(Q is an assertion involving a variable x and $Q[x/\mathbb{E}]$ indicates the same assertion with all occurrences of x replaced by the expression $\mathbb{E}$)

If we want x to have some property Q after the assignment, then that property must hold for the value $\mathbb{E}$ assigned to x before the assignment is executed.

The Assignment Axiom

Example

The backwards rule is false: $\{Q\}$ `x := `$\mathbb{E}$ $\{Q[x/\mathbb{E}]\}$

If we want to apply this wrong "axiom" to the precondition $x = 0$ and code fragment `x := 1` we would get

$$\{x = 0\} \ \texttt{x := 1} \ \{1 = 0\}$$

which says "if $x = 0$ initially and `x := 1` terminates then $1 = 0$ finally".

Work from the Goal backwards

It may seem natural to start at the precondition and reason towards the postcondition, but this is not the best way to do Hoare logic.

Instead you start with the goal (postcondition) and go "backwards".

Example

To apply the assignment axiom

$$\{Q[x/\mathbb{E}]\}\ \texttt{x := } \mathbb{E}\ \{Q\}$$

take the postcondition, copy it across the precondition and then replace all occurrences of x with $\mathbb{E}$.

Note that the postcondition may have no, one, or many occurrences of x.

All get replaced by $\mathbb{E}$ in the precondition!

Proof rule for The Assignment

Assignment Axiom: $\{Q[x/\mathbb{E}]\}$ `x := e` $\{Q\}$

Example

Suppose the code fragment is `x := 2` and suppose the desired postcondition is $y = x$.

An instance of the assignment axiom:

$$\{y = 2\}\ \texttt{x := 2}\ \{y = x\}$$

Proof rule for The Assignment

You can always replace predicates by equivalent predicates; just label your proof step with "precondition equivalence" or "postcondition equivalence".

Example

How should we prove

$$\{y > 0\}\ \texttt{x := y+3}\ \{x > 3\}?$$

Start with the postcondition $x > 3$ and apply the assignment axiom:

$$\{y + 3 > 3\}\ \texttt{x := y+3}\ \{x > 3\}$$

Then use the fact that $y + 3 > 3$ is equivalent with $y > 0$ to get the result.

Proof rule for The Assignment

Example

What if we want to prove

$$\{y = 2\}\ \texttt{x := y}\ \{x > 0\}?$$

This is clearly true. But the assignment axiom gives us:

$$\{y > 0\}\ \texttt{x := y}\ \{x > 0\}$$

We cannot just replace $y > 0$ with $y = 2$ – they are not equivalent!

Weak and strong predicates

A predicate P is stronger than Q if P implies Q.

If P is stronger than Q, then whenever P is true then Q is true as well.

Example

A politician's example:

- *I will keep unemployment below* 3% is stronger than
- *I will keep unemployment below* 15%

The strongest possible assertion is $\perp$.

The weakest possible assertion is $\top$.

Strong postconditions

Example

The Hoare triple $\{x = 5\}$ `x := x+1` $\{x = 6\}$ says more about the code than does $\{x = 5\}$ `x := x+1` $\{x > 0\}$.

If a postcondition Q_1 is stronger than Q_2, then $\{P\}\ \mathbb{C}\ \{Q_1\}$ is a stronger statement than $\{P\}\ \mathbb{C}\ \{Q_2\}$.

Example

Since the postcondition $x = 6$ is stronger than $x > 0$ (as $x = 6 \to x > 0$), then $\{x = 5\}$ `x := x+1` $\{x = 6\}$ is a stronger statement than $\{x = 5\}$ `x := x+1` $\{x > 0\}$.

Weak preconditions

Example

The Hoare triple $\{x > 0\}$ `x := x+1` $\{x > 1\}$ says more about the code than does $\{x = 5\}$ `x := x+1` $\{x > 1\}$.

If a precondition P_1 is weaker than P_2, then $\{P_1\}\ \mathbb{C}\ \{Q\}$ is a stronger statement than $\{P_2\}\ \mathbb{C}\ \{Q\}$.

Example

Since the precondition $x > 0$ is weaker than $x = 5$, then $\{x > 0\}$ `x := x+1` $\{x > 1\}$ is a stronger statement than $\{x = 5\}$ `x := x+1` $\{x > 1\}$.

Proof rule for Strengthening Preconditions (Rule 2/6)

It is safe (sound) to make a precondition more *specific* (stronger).

Precondition Strengthening rule:

$$\boxed{\dfrac{P_s \rightarrow P_w \qquad \{P_w\}\ \mathbb{C}\ \{Q\}}{\{P_s\}\ \mathbb{C}\ \{Q\}}}$$

Example

$$\dfrac{y = 2 \rightarrow y > 0 \qquad \{y > 0\}\ \mathtt{x := y}\ \{x > 0\}}{\{y = 2\}\ \mathtt{x := y}\ \{x > 0\}}$$

Proof rule for Weakening Postconditions (Rule 3/6)

It is safe (sound) to make a postcondition less *specific* (weaker).

Postcondition Weakening rule:

$$\boxed{\dfrac{\{P\}\ \mathbb{C}\ \{Q_s\} \qquad Q_s \rightarrow Q_w}{\{P\}\ \mathbb{C}\ \{Q_w\}}}$$

Example

$$\dfrac{\{x > 2\}\ \mathtt{x := x + 1}\ \{x > 3\} \qquad x > 3 \rightarrow x > 1}{\{x > 2\}\ \mathtt{x := x + 1}\ \{x > 1\}}$$

Proof rule for Sequencing (Rule 4/6)

Imperative programs consist of a sequence of statements, affecting the state one after the other.

Sequencing rule:

$$\boxed{\frac{\{P\}\mathbb{C}_1\{Q\} \qquad \{Q\}\mathbb{C}_2\{R\}}{\{P\}\mathbb{C}_1;\mathbb{C}_2\{R\}}}$$

Example

$$\frac{\{x > 2\}\texttt{x} := \texttt{x} + 1\{x > 3\} \qquad \{x > 3\}\texttt{x} := \texttt{x} + 2\{x > 5\}}{\{x > 2\}\texttt{x} := \texttt{x} + 1\ ;\ \texttt{x} := \texttt{x} + 2\{x > 5\}}$$

How do we get the intermediate condition?

In the rule

$$\boxed{\frac{\{P\}\mathbb{C}_1\{Q\} \qquad \{Q\}\mathbb{C}_2\{R\}}{\{P\}\mathbb{C}_1;\mathbb{C}_2\{R\}}}$$

our precondition P and postcondition R will be given to us, but how do we come up with Q?

By starting with our goal R and working backwards!

$$\frac{\{x > 2\}\texttt{x} := \texttt{x} + 1\{Q\} \qquad \{Q\}\texttt{x} := \texttt{x} + 2\{x > 5\}}{\{x > 2\}\texttt{x} := \texttt{x} + 1; \texttt{x} := \texttt{x} + 2\{x > 5\}}$$

Laying out a proof

Example
Suppose we want to prove

$$\{x = 3\}\ \texttt{x := x+1; x := x+2}\ \{x > 5\}.$$

Note the numbered proof steps and justifications!

1. $\{x + 2 > 5\}\ \texttt{x := x+2}\ \{x > 5\}$ (Assignment)
2. $\{x > 3\}\ \texttt{x := x+2}\ \{x > 5\}$ (1, Precondition Equivalence)
3. $\{x + 1 > 3\}\ \texttt{x := x+1}\ \{x > 3\}$ (Assignment)
4. $\{x > 2\}\ \texttt{x := x+1}\ \{x > 3\}$ (3, Precondition Equivalence)
5. $\{x > 2\}\ \texttt{x := x+1; x := x+2}\ \{x > 5\}$ (2,4, Sequencing)
6. $x = 3 \rightarrow x > 2$ (Basic arithmetics)
7. $\{x = 3\}\ \texttt{x := x+1; x := x+2}\ \{x > 5\}$. (5,6, Precondition Strength)

Proof rule for Conditionals (Rule 5/6)

Conditional rule:

$$\boxed{\frac{\{P \wedge \mathbb{B}\}\ \mathbb{C}_1\ \{Q\} \qquad \{P \wedge \neg\mathbb{B}\}\ \mathbb{C}_2\ \{Q\}}{\{P\}\ \texttt{if}\ \mathbb{B}\ \texttt{then}\ \mathbb{C}_1\ \texttt{else}\ \mathbb{C}_2\ \{Q\}}}$$

- When a conditional is executed, either $\mathbb{C}_1$ or $\mathbb{C}_2$ is executed.
- Therefore, if the conditional is to establish Q, then both $\mathbb{C}_1$ and $\mathbb{C}_2$ must establish Q.
- Similarly, if the precondition for the conditional is P, then it must also be a precondition for the both branches $\mathbb{C}_1$ and $\mathbb{C}_2$.
- The choice between $\mathbb{C}_1$ and $\mathbb{C}_2$ depends on evaluating $\mathbb{B}$ in the initial state, so we can also assume $\mathbb{B}$ to be a precondition for $\mathbb{C}_1$ and $\neg\mathbb{B}$ to be a precondition for $\mathbb{C}_2$.

Proof rule for Conditionals

Conditional rule:

$$\frac{\{P \wedge \mathbb{B}\}\ \mathbb{C}_1\ \{Q\} \qquad \{P \wedge \neg\mathbb{B}\}\ \mathbb{C}_2\ \{Q\}}{\{P\}\ \texttt{if}\ \mathbb{B}\ \texttt{then}\ \mathbb{C}_1\ \texttt{else}\ \mathbb{C}_2\ \{Q\}}$$

Example
Suppose we wish to prove

$$\{x > 2\}\ \texttt{if x > 2 then y := 1 else y := -1}\ \{y > 0\}$$

The proof rule for conditionals suggests we prove:

$$\{(x > 2) \wedge (x > 2)\}\ \texttt{y := 1}\ \{y > 0\} \qquad \{(x > 2) \wedge \neg(x > 2)\}\ \texttt{y := -1}\ \{y > 0\}$$

Simplifying the preconditions, we get

1. $\{x > 2\}\ \texttt{y := 1}\ \{y > 0\}$
2. $\{\bot\}\ \texttt{y := -1}\ \{y > 0\}$

Proof rule for Conditionals

Example (cont.)

For the subgoal 1. $\{x > 2\}$ `y:=1` $\{y > 0\}$ we have the following proof

3. $\{1 > 0\}$ `y := 1` $\{y > 0\}$ (Assignment rule)
4. $1 > 0 \leftrightarrow \top$ (Propositional logic)
5. $\{\top\}$ `y := 1` $\{y > 0\}$ (Precondition equivalence)
6. $x > 2 \rightarrow \top$ (Propositional logic)
7. $\{x > 2\}$ `y := 1` $\{y > 0\}$ (Precondition Strenthening)

For the subgoal 2. $\{\bot\}$ `y:=-1` $\{y > 0\}$ we have the following proof

8. $\{-1 > 0\}$ `y := -1` $\{y > 0\}$ (Assignment rule)
9. $-1 > 0 \leftrightarrow \bot$ (Propositional logic)
10. $\{\bot\}$ `y := -1` $\{y > 0\}$ (Precondition equivalence)

Proof rule for Conditionals

Exercise:

How would you derive a rule for a conditional statement without else?

$$\texttt{if } \mathbb{B} \texttt{ then } \mathbb{C}$$

Quiz time!

https://tinyurl.com/FMI-PV2023-Quiz2

References

- Lecture Notes on "Formal Methods for Software Engineering", Australian National University, Rajeev Goré.
- Mike Gordon, "Specification and Verification I", chapters 1 and 2.
- Michael Huth, Mark Ryan, "Logic in Computer Science: Modeling and Reasoning about Systems", 2nd edition, Cambridge University Press, 2004.
- Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog, "Verification of Sequential and Concurrent Programs", 3rd edition, Springer.