Hoare logic

Lecture 3: Examples in Hoare logic

Jean Pichon-Pharabod
University of Cambridge

CST Part II – 2017/18

Recap

In the past lectures, we have discussed Hoare logic: we have given

- a notation for specifying the intended behaviour of programs:

$$\{P\}\ C\ \{Q\}$$

- a semantics capturing the precise meaning of this notation:

$$\models \{P\}\ C\ \{Q\}$$

- a syntactic proof system for proving that programs satisfy their intended specification:

$$\vdash \{P\}\ C\ \{Q\}$$

- a proof of soundness of that proof system:

$$\vdash \{P\}\ C\ \{Q\} \Rightarrow \models \{P\}\ C\ \{Q\}$$

Introduction

Today, we will **use** Hoare logic, and look at how to find proofs.

We will first establish derived rules that make using Hoare logic easier.

Using these, we will then verify two simple programs to exercise Hoare logic, and to illustrate how to find invariants in Hoare logic.

We will also find proof rules for total correctness.

Finding proofs

Finding proofs: backwards reasoning

Forward reasoning

The proof rules we have seen so far are best suited for **forward** (also "top down") reasoning, where a proof tree is constructed starting from the leaves, going towards the root.

For instance, consider a proof of

$$\vdash \{X = a\}\ X := X + 1\ \{X = a + 1\}$$

using the assignment rule:

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

Proof of a simple assignment using the forward reasoning

$$\frac{\dfrac{\vdots}{\vdash X = 1 \Rightarrow X + 1 = a + 1} \quad \dfrac{}{\vdash \{(X = a + 1)[X + 1/X]\} X := X + 1 \{X = a + 1\}} \quad \dfrac{\vdots}{\vdash X = a + 1 \Rightarrow X = a + 1}}{\vdash \{X = a\} X := X + 1 \{X = a + 1\}}$$

Given that $(X = a + 1)[X + 1/X] \equiv X + 1 = a + 1$.

Backwards reasoning & backwards assignment rule

It is often more natural to work **backwards** (also "bottom up"), starting from the root of the proof tree, and generating new subgoals until all the nodes have been shown to be derivable.

We can **derive** rules better suited for backwards reasoning.

For instance, we can derive this backwards assignment rule:

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

This rule does not impose that the precondition is of a given shape, but instead that it implies an assertion of the desired shape.

Backwards assignment rule

We can derive the backwards assignment rule by combining the assignment rule with the rule of consequence:

$$\frac{\vdash P \Rightarrow Q[E/V] \qquad \dfrac{}{\vdash \{Q[E/V]\}V := E\{Q\}} \qquad \dfrac{\vdots}{\vdash Q \Rightarrow Q}}{\vdash \{P\}V := E\{Q\}}$$

Backwards sequenced assignment rule

The sequence rule can already be applied bottom up, but requires us to guess an assertion R:

$$\frac{\vdash \{P\}\ C_1\ \{R\} \qquad \vdash \{R\}\ C_2\ \{Q\}}{\vdash \{P\}\ C_1; C_2\ \{Q\}}$$

In the case of a command sequenced before an assignment, we can avoid having to guess R by using the sequenced assignment rule:

$$\frac{\vdash \{P\}\ C\ \{Q[E/V]\}}{\vdash \{P\}\ C; V := E\ \{Q\}}$$

This is easily derivable using the sequencing rule and the backwards assignment rule (exercise).

Backwards loop rule

In the same way, we can derive a backwards reasoning rule for loops by building in consequence:

$$\frac{\vdash P \Rightarrow I \qquad \vdash \{I \wedge B\}\ C\ \{I\} \qquad \vdash I \wedge \neg B \Rightarrow Q}{\vdash \{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{Q\}}$$

This rule still requires us to guess I to apply it bottom-up.

Backwards skip and conditional rules

We can also derive a backwards skip rule that builds in consequence:

$$\frac{\vdash P \Rightarrow Q}{\vdash \{P\}\ \textbf{skip}\ \{Q\}}$$

The conditional rule needs not be changed:

$$\frac{\vdash \{P \wedge B\}\ C_1\ \{Q\} \qquad \vdash \{P \wedge \neg B\}\ C_2\ \{Q\}}{\vdash \{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \{Q\}}$$

Backwards reasoning proof rules

$$\frac{\vdash P \Rightarrow Q}{\vdash \{P\}\ \textbf{skip}\ \{Q\}} \qquad \frac{\vdash \{P\}\ C_1\ \{R\} \qquad \vdash \{R\}\ C_2\ \{Q\}}{\vdash \{P\}\ C_1; C_2\ \{Q\}}$$

$$\frac{\vdash P \Rightarrow Q[E/V]}{\vdash \{P\}\ V := E\ \{Q\}} \qquad \frac{\vdash \{P\}\ C\ \{Q[E/V]\}}{\vdash \{P\}\ C; V := E\ \{Q\}}$$

$$\frac{\vdash P \Rightarrow I \qquad \vdash \{I \wedge B\}\ C\ \{I\} \qquad \vdash I \wedge \neg B \Rightarrow Q}{\vdash \{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{Q\}}$$

$$\frac{\vdash \{P \wedge B\}\ C_1\ \{Q\} \qquad \vdash \{P \wedge \neg B\}\ C_2\ \{Q\}}{\vdash \{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \{Q\}}$$

There is no separate rule of consequence anymore.
These rules are still relatively complete.

Finding proofs: factorial

Finding proofs: loop invariants

Specifying a program computing factorial

We wish to verify that the following command computes the factorial of X, and stores the result in Y:

$$\textbf{while}\ X \neq 0\ \textbf{do}\ (Y := Y \times X; X := X - 1)$$

First, we need to formalise the specification:

- Factorial is only defined for non-negative numbers, so X should be non-negative in the initial state.
- The terminal state of Y should be equal to the factorial of the initial state of X.
- The implementation assumes that Y is equal to 1 initially.

A specification of a program computing factorial

This corresponds to the following partial correctness triple:

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$
$$\quad \textbf{while } X \neq 0 \textbf{ do } (Y := Y \times X; X := X - 1)$$
$$\{Y = x!\}$$

Here, '!' denotes the usual mathematical factorial function.

Note that we used an auxiliary variable x to record the initial value of X and relate the terminal value of Y with the initial value of X.

How does one find an invariant?

$$\frac{\vdash P \Rightarrow I \qquad \vdash \{I \wedge B\}\ C\ \{I\} \qquad \vdash I \wedge \neg B \Rightarrow Q}{\vdash \{P\}\ \textbf{while } B \textbf{ do } C\ \{Q\}}$$

Here, I is an invariant, meaning that it

- must hold initially;
- must be preserved by the loop body when B is true; and
- must imply the desired postcondition when B is false.

Analysing the factorial implementation

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$
$$\quad \textbf{while } X \neq 0 \textbf{ do } (Y := Y \times X; X := X - 1)$$
$$\{Y = x!\}$$

How does this program work?

Observations about the factorial implementation

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$
$$\quad \textbf{while } X \neq 0 \textbf{ do } (Y := Y \times X; X := X - 1)$$
$$\{Y = x!\}$$

iteration	Y	X
0	1	x
1	$1 \times x$	$x - 1$
2	$1 \times x \times (x - 1)$	$x - 2$
3	$1 \times x \times (x - 1) \times (x - 2)$	$x - 3$
$\vdots$	$\vdots$	$\vdots$
x	$1 \times x \times (x - 1) \times (x - 2) \times \cdots \times 1$	0

Y is the value computed so far, and $X!$ remains to be computed.

An invariant for the factorial implementation

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$
$$\quad \textbf{while } X \neq 0 \textbf{ do } (Y := Y \times X; X := X - 1)$$
$$\{Y = x!\}$$

Take I to be $Y \times X! = x! \wedge X \geq 0$.
(We need $X \geq 0$ for $X!$ to make sense.)

Backwards reasoning proof rules (recap)

$$\frac{\vdash P \Rightarrow Q}{\vdash \{P\}\ \textbf{skip}\ \{Q\}} \qquad \frac{\vdash \{P\}\ C_1\ \{R\} \qquad \vdash \{R\}\ C_2\ \{Q\}}{\vdash \{P\}\ C_1; C_2\ \{Q\}}$$

$$\frac{\vdash P \Rightarrow Q[E/V]}{\vdash \{P\}\ V := E\ \{Q\}} \qquad \frac{\vdash \{P\}\ C\ \{Q[E/V]\}}{\vdash \{P\}\ C; V := E\ \{Q\}}$$

$$\frac{\vdash P \Rightarrow I \qquad \vdash \{I \wedge B\}\ C\ \{I\} \qquad \vdash I \wedge \neg B \Rightarrow Q}{\vdash \{P\}\ \textbf{while } B \textbf{ do } C\ \{Q\}}$$

$$\frac{\vdash \{P \wedge B\}\ C_1\ \{Q\} \qquad \vdash \{P \wedge \neg B\}\ C_2\ \{Q\}}{\vdash \{P\}\ \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2\ \{Q\}}$$

Derivation tree of the verified factorial

Finding proofs: proof outlines

Proof outlines

Derivations in Hoare logic are often more readable when given as **proof outlines** instead of proof trees.

Proof outlines are code listings annotated with Hoare logic assertions between statements.

Sequences of Hoare logic assertions indicate reasoning about assertions.

Finding proofs: Fibonacci

Proof outline for the implementation of factorial

$\{X = x \wedge X \geq 0 \wedge Y = 1\}$
$\{Y \times X! = x! \wedge X \geq 0\}$
while $X \neq 0$ **do**
  $(\{Y \times X! = x! \wedge X \geq 0 \wedge X \neq 0\}$
  $\{(Y \times X) \times (X-1)! = x! \wedge (X-1) \geq 0\}$
  $Y := Y \times X;$
  $\{Y \times (X-1)! = x! \wedge (X-1) \geq 0\}$
  $X := X - 1$
  $\{Y \times X! = x! \wedge X \geq 0\})$
$\{Y \times X! = x! \wedge X \geq 0 \wedge \neg(X \neq 0)\}$
$\{Y = x!\}$

A verified Fibonacci implementation

We wish to verify that the following command computes the N-th Fibonacci number (indexed from 1), and stores the result in Y. This corresponds to the following partial correctness Hoare triple:

$\{1 \leq N \wedge N = n\}$
$X = 0;$
$Y := 1;$
$Z := 1;$
while $Z < N$ **do**
  $(Y := X + Y; X := Y - X; Z := Z + 1)$
$\{Y = \mathit{fib}(n)\}$

Recall that the Fibonacci sequence is defined by

$$\mathit{fib}(1) = 1, \quad \mathit{fib}(2) = 1, \quad \forall n > 2.\, \mathit{fib}(n) = \mathit{fib}(n-1) + \mathit{fib}(n-2)$$

Moreover, for convenience, we assume $\mathit{fib}(0) = 0$.

A verified Fibonacci implementation

Reasoning about the initial assignment of constants is easy.

How can we verify the loop?

$\{X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n\}$
while $Z < N$ **do**
$(Y := X + Y; X := Y - X; Z := Z + 1)$
$\{Y = fib(n)\}$

First, we need to understand the implementation.

Observations about the implementation of Fibonacci

$\{X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n\}$
while $Z < N$ **do**
$(Y := X + Y; X := Y - X; Z := Z + 1)$
$\{Y = fib(n)\}$

iteration	0	1	2	3	4	5	6	$\cdots$	$n-1$
Y	1	1	2	3	5	8	13	$\cdots$	$fib(n)$
X	0	1	1	2	3	5	8	$\cdots$	$fib(n-1)$
Z	1	2	3	4	5	6	7	$\cdots$	n

Analysing the implementation of Fibonacci

$\{X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n\}$
while $Z < N$ **do**
$(Y := X + Y; X := Y - X; Z := Z + 1)$
$\{Y = fib(n)\}$

Z is used to count loop iterations, and Y and X are used to compute the Fibonacci number:
Y contains the current Fibonacci number,
and X contains the previous Fibonacci number.

This suggests trying the invariant
$Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0$.
(We need $Z > 0$ for $fib(Z-1)$ to make sense.)

Trying an invariant for the Fibonacci implementation

$\{X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n\}$
while $Z < N$ **do**
$(Y := X + Y; X := Y - X; Z := Z + 1)$
$\{Y = fib(n)\}$

Take $I \equiv Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0$.
Then we have to prove:

- $(X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n) \Rightarrow I$
- $\{I \wedge (Z < N)\}\ Y := X + Y; X := Y - X; Z := Z + 1\ \{I\}$
- $(I \wedge \neg(Z < N)) \Rightarrow Y = fib(n)$

Do all these hold? Only the first two do. (Exercise.)

A better invariant for the Fibonacci implementation

$\{X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n\}$

while $Z < N$ **do**

$(Y := X + Y; X := Y - X; Z := Z + 1)$

$\{Y = fib(n)\}$

While $Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0$ **is an invariant**, it is not strong enough to establish the desired postcondition.

We need to know that when the loop terminates, then $Z = n$.
It suffices to strengthen the invariant to:

$$Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0 \wedge Z \leq N \wedge N = n$$

Proof outline for the loop of the Fibonacci implementation

$\{X = 0 \wedge Y = 1 \wedge Z = 1 \wedge 1 \leq N \wedge N = n\}$

$\{Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0 \wedge Z \leq N \wedge N = n\}$

while $Z < N$ **do**

$(\{Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0 \wedge Z \leq N \wedge N = n \wedge Z < N\}$

$\{X + Y = fib(Z+1) \wedge (X+Y) - X = fib(Z) \wedge Z + 1 > 0 \wedge Z + 1 \leq N \wedge N = n\}$

$Y := X + Y;$

$\{Y = fib(Z+1) \wedge Y - X = fib(Z) \wedge Z + 1 > 0 \wedge Z + 1 \leq N \wedge N = n\}$

$X := Y - X;$

$\{Y = fib(Z+1) \wedge X = fib(Z) \wedge Z + 1 > 0 \wedge Z + 1 \leq N \wedge N = n\}$

$\{Y = fib(Z+1) \wedge X = fib((Z+1)-1) \wedge Z + 1 > 0 \wedge Z + 1 \leq N \wedge N = n\}$

$Z := Z + 1$

$\{Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0 \wedge Z \leq N \wedge N = n\})$

$\{Y = fib(Z) \wedge X = fib(Z-1) \wedge Z > 0 \wedge Z \leq N \wedge N = n \wedge \neg(Z < N)\}$

$\{Y = fib(n)\}$

Summary of proof-finding

We have looked at how to find proofs:

- how "backwards" reasoning can help;
- how to find invariants.

Finding invariants is difficult!

Writing out full proof trees or even proof outlines by hand is tedious and error-prone, even for simple programs.

In the next lecture, we will look at using mechanisation to check our proofs and help discharge simple proof obligations.

Total correctness

Total correctness

So far, we have mainly concerned ourselves with partial correctness. What about total correctness?

Recall: the total correctness triple, $[P]\ C\ [Q]$ holds if and only if

- whenever C is executed in a state satisfying P, then C terminates, and the terminal state satisfies Q.

Total correctness

while commands are the commands that introduce non-termination.

Except for the loop rule, all the rules described so far are sound for total correctness as well as partial correctness.

Unsoundness of the loop rule for total correctness

The loop rule that we have for partial correctness is not sound for total correctness:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash (\top \wedge \mathbf{T}) \Rightarrow \top} \quad \dfrac{}{\vdash \{\top\}\ \mathbf{skip}\ \{\top\}} \quad \dfrac{\vdots}{\vdash \top \Rightarrow \top}}{\vdash \{\top \wedge \mathbf{T}\}\ \mathbf{skip}\ \{\top\}}}{\vdash \{\top\}\ \mathbf{while}\ \mathbf{T}\ \mathbf{do}\ \mathbf{skip}\ \{\top \wedge \neg \mathbf{T}\}} \quad \dfrac{\vdots}{\vdash \top \wedge \neg \mathbf{T} \Rightarrow \bot}$$
$$\vdash \{\top\}\ \mathbf{while}\ \mathbf{T}\ \mathbf{do}\ \mathbf{skip}\ \{\bot\}$$

If the loop rule were sound for total correctness, then this would show that **while T do skip** always terminates in a state satisfying $\bot$.

Loop variants

We need an alternative total correctness loop rule that ensures that the loop always terminates.

The idea is to show that some non-negative integer quantity decreases on each iteration of the loop.

If this is the case, then the loop terminates, as there would otherwise be an infinite decreasing sequence of natural numbers.

This decreasing quantity is called a variant.

Loop rule for total correctness

In the rule below, the variant is E, and the fact that it decreases is specified with an auxiliary variable n:

$$\frac{\vdash [P \wedge B \wedge (E = n)]\ C\ [P \wedge (E < n)] \qquad \vdash P \wedge B \Rightarrow E \geq 0}{\vdash [P]\ \textbf{while}\ B\ \textbf{do}\ C\ [P \wedge \neg B]}$$

The second hypothesis ensures that the variant is non-negative.

Backwards reasoning total correctness loop rule

Using the rule of consequence, we can derive the following backwards reasoning total correctness loop rule:

$$\frac{\vdash P \Rightarrow I \qquad \vdash I \wedge \neg B \Rightarrow Q \qquad \vdash I \wedge B \Rightarrow E \geq 0 \qquad \vdash [I \wedge B \wedge (E = n)]\ C\ [I \wedge (E < n)]}{\vdash [P]\ \textbf{while}\ B\ \textbf{do}\ C\ [Q]}$$

Total correctness: factorial example

Consider the factorial computation we looked at before:

$$\begin{array}{l}[X = x \wedge X \geq 0 \wedge Y = 1] \\ \quad \textbf{while}\ X \neq 0\ \textbf{do}\ (Y := Y \times X; X := X - 1) \\ [Y = x!]\end{array}$$

By assumption, X is non-negative and decreases in each iteration of the loop.

To verify that this factorial implementation terminates, we can thus take the variant E to be X.

Total correctness: factorial example

$$\begin{array}{l}[X = x \wedge X \geq 0 \wedge Y = 1] \\ \quad \textbf{while}\ X \neq 0\ \textbf{do}\ (Y := Y \times X; X := X - 1) \\ [Y = x!]\end{array}$$

Take I to be $Y \times X! = x! \wedge X \geq 0$, and E to be X.

Then we have to show that

- $X = x \wedge X \geq 0 \wedge Y = 1 \Rightarrow I$
- $[I \wedge X \neq 0 \wedge (X = n)]\ Y := Y \times X; X := X - 1\ [I \wedge (X < n)]$
- $I \wedge \neg(X \neq 0) \Rightarrow Y = x!$
- $I \wedge X \neq 0 \Rightarrow X \geq 0$

Relation between partial and total correctness

The relation between partial and total correctness is informally given by the equation

$$\text{total correctness} = \text{partial correctness} + \text{termination}$$

This is captured formally by the following properties:

- If $\vdash \{P\}\ C\ \{Q\}$ and $\vdash [P]\ C\ [\top]$, then $\vdash [P]\ C\ [Q]$.
- If $\vdash [P]\ C\ [Q]$, then $\vdash \{P\}\ C\ \{Q\}$.

Summary of total correctness

We have given rules for total correctness,
similar to those for partial correctness.

Only the loop rule differs: the premises of the loop rule require that the loop body decreases a non-negative expression.

It is even possible to do amortised, asymptotic complexity analysis in Hoare logic:

- A Fistful of Dollars, Armaël Guéneau et al., ESOP 2018

In the next lecture, we will look at using mechanisation to check our proofs and help discharge simple proof obligations.