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

Introduction

In the past lectures we have given

- a notation for specifying the intended behaviour of programs
- a proof system for proving that programs satisfy their intended specification
- a semantics capturing the precise meaning of this notation

Now we are going to look at ways of finding proofs, including:

- derived rules & backwards reasoning
- finding invariants
- ways of annotating programs prior to proving

We are also going to look at proof rules for total correctness.

Forward and backwards reasoning

Forward & backwards reasoning

The proof rules we have seen so far are best suited for **forward** directed reasoning where a proof tree is constructed starting from axioms towards the desired specification.

For instance, consider a proof of

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

using the assignment rule:

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

Forward reasoning

It is often more natural to work **backwards**, starting from the root of the proof tree and generating new subgoals until all the leaves are axioms.

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

For instance, we can derive this backwards-assignment rule:

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

Backwards sequenced assignment rule

The sequence rule can already be applied bottom, 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 with 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 reasoning

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.

Proof rules

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

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

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

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

Finding loop invariants

A verified factorial implementation

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

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

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 verified factorial implementation

This corresponds to the following partial correctness Hoare triple:

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

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?

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

Here I is an invariant that

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

How does one find an invariant?

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

The invariant I should express

- what **has been done so far** and what **remains to be done**
- that nothing has been done initially
- that nothing remains to be done when B is false

A verified factorial implementation

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

Take I to be $Y * X! = x! \wedge X \geq 0$, then we must prove:

- $X = x \wedge X \geq 0 \wedge Y = 1 \Rightarrow I$
- $\{I \wedge X \neq 0\}\ Y := Y * X; X := X - 1\ \{I\}$
- $I \wedge X = 0 \Rightarrow Y = x!$

The first and last proof obligation follow by basic arithmetic.

Proof rules

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

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

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

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

Proof outlines

In the literature, hand-written proofs in Hoare logic are often written as informal **proof outlines** instead of proof trees.

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

Proof outlines

Here is an example of a proof outline for the second proof obligation for the factorial function:

$$
\begin{array}{l}
\{Y * X! = x! \land X \geq 0 \land X \neq 0\} \\
\{(Y * X) * (X - 1)! = x! \land (X - 1) \geq 0\} \\
\quad Y := Y * X; \\
\{Y * (X - 1)! = x! \land (X - 1) \geq 0\} \\
\quad X := X - 1 \\
\{Y * X! = x! \land X \geq 0\}
\end{array}
$$

Proof outlines

Writing out full proof trees or 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 trivial proof obligations.

A verified fibonacci implementation

Imagine we want to prove the following fibonacci implementation satisfies the given specification.

$$
\begin{array}{l}
\{X = 0 \land Y = 1 \land Z = 1 \land 1 \leq N \land N = n\} \\
\textbf{while } (Z < N) \textbf{ do} \\
\quad (Y := X + Y; X := Y - X; Z := Z + 1) \\
\{Y = \mathit{fib}(n)\}
\end{array}
$$

First we need to understand the implementation:

- the Z variable is used to count loop iterations
- and Y and X are used to compute the fibonacci number

A verified fibonacci implementation

$$
\begin{array}{l}
\{X = 0 \land Y = 1 \land Z = 1 \land 1 \leq N \land N = n\} \\
\textbf{while } (Z < N) \textbf{ do} \\
\quad (Y := X + Y; X := Y - X; Z := Z + 1) \\
\{Y = \mathit{fib}(n)\}
\end{array}
$$

Take $I \equiv Y = \mathit{fib}(Z) \land X = \mathit{fib}(Z - 1)$, then we have to prove:

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

Do all these hold?

A verified fibonacci implementation

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

Take $I \equiv Y = fib(Z) \wedge X = fib(Z-1)$, 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? The first two do (Exercise!)

A verified fibonacci implementation

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

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

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

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

Total correctness

Total correctness

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

Recall, total correctness = partial correctness + termination.

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 only commands that might not terminate.

Except for the WHILE-rule, all the axioms and rules described so far are sound for total correctness as well as partial correctness.

Total correctness

The WHILE-rule is not sound for total correctness

$$\dfrac{\dfrac{\dfrac{}{\vdash \{\top\}\ X := X\ \{\top\}}}{\vdash \{\top \land \top\}\ X := X\ \{\top\}}}{\vdash \{\top\}\ \textbf{while true do}\ X := X\ \{\top \land \neg\top\}} \qquad \vdash \top \land \neg\top \Rightarrow \bot$$

$$\overline{\vdash \{\top\}\ \textbf{while true do}\ X := X\ \{\bot\}}$$

If the WHILE-rule was sound for total correctness, then this would show that **while true do** $X := X$ always terminates in a state satisfying $\bot$.

Total correctness

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

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

This decreasing quantity is called a variant.

Total correctness

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

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

The second hypothesis ensures the variant is non-negative.

Total correctness

Using the rule-of-consequence we can derive the following backwards-reasoning total correctness WHILE rule

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

Total correctness: Factorial example

Consider the factorial computation we looked at before

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

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

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

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

Then we have to show that

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

Total correctness

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

Total correctness = partial correctness + termination

This is captured formally by the following inference rules

$$\frac{\vdash \{P\}\ C\ \{Q\} \qquad \vdash [P]\ C\ [\top]}{\vdash [P]\ C\ [Q]} \qquad \frac{\vdash [P]\ C\ [Q]}{\vdash \{P\}\ C\ \{Q\}}$$

Summary: Total correctness

We have given rules for total correctness.

They are similar to those for partial correctness

The main difference is in the WHILE-rule

- WHILE commands are the only ones that can fail to terminate
- for WHILE commands we must prove that a non-negative expression is decreased by the loop body