# Hoare logic

## Lecture 2: Examples in Hoare logic

---

**Christopher Pulte** cp526
University of Cambridge

CST Part II – 2024/2025

# Recap

In the previous lecture, we introduced Hoare logic, which uses Hoare triples to specify the behaviour of imperative programs by relating the initial state of a program with its terminate state.

Today, we will use Hoare logic to specify and verify some simple programs.

# Proof outlines

# Proof outlines

Derivations in Hoare logic are often more readable when given as **proof outlines** instead of proof trees. A proof outline of a command is an annotation of the command with the pre- and postcondition of each sub-command...

Instead of writing

$$\dfrac{\vdash \{(X+1)\times 2 = 4\}\ X := X+1\ \{X\times 2 = 4\} \qquad \vdash \{X\times 2 = 4\}\ X := X\times 2\ \{X = 4\}}{\vdash \{(X+1)\times 2 = 4\}\ X := X+1; X := X\times 2\ \{X = 4\}}$$

we can write

$$\begin{array}{l}\{(X+1)\times 2 = 4\}\\ X := X+1;\\ \{X\times 2 = 4\}\\ X := X\times 2\\ \{X = 4\}\end{array}$$

# Proof outlines

. . .and where sequences of assertions indicate uses of the rule of consequence. We elide sides of the rule of consequence that do not change the assertion. We also elide (but need to check!) the derivations of implications between assertions.

Instead of writing

$$\dfrac{\dfrac{\vdots}{\vdash_{FOL} X = 1 \Rightarrow X + 1 = 2} \qquad \dfrac{}{\vdash \{X + 1 = 2\}\ X := X + 1\ \{X = 2\}} \qquad \dfrac{\vdots}{\vdash_{FOL} X = 2 \Rightarrow X = 2}}{\vdash \{X = 1\}\ X := X + 1\ \{X = 2\}}$$

we can write

$$\begin{array}{l} \{X = 1\} \\ \{X + 1 = 2\} \\ X := X + 1 \\ \{X = 2\} \end{array}$$

# Proof outline for the integer square root

$\{X = x \wedge x \geq 0\}$

$\{X = x \wedge x \geq 0 \wedge 0 \times 0 \leq x\}$

$S := 0;$

$\{X = x \wedge x \geq 0 \wedge S \times S \leq x\}$

**while** $(S + 1) \times (S + 1) \leq X$ **do** (

$\{X = x \wedge x \geq 0 \wedge S \times S \leq x \wedge (S + 1) \times (S + 1) \leq X\}$

$\{X = x \wedge x \geq 0 \wedge (S + 1) \times (S + 1) \leq x\}$

$S := S + 1$

$\{X = x \wedge x \geq 0 \wedge S \times S \leq x\}$

)$\{X = x \wedge x \geq 0 \wedge S \times S \leq x \wedge \neg((S + 1) \times (S + 1) \leq X)\}$

$\{X = x \wedge S \times S \leq x \wedge x < (S + 1) \times (S + 1)\}$

# Factorial

# 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:

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

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 a good invariant?

$$\frac{\vdots}{\vdash_{FOL} P' \Rightarrow P} \qquad \frac{\vdash \{P \wedge B\}\ C\ \{P\}}{\vdash \{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{P \wedge \neg B\}} \qquad \frac{\vdots}{\vdash_{FOL} P \wedge \neg B \Rightarrow Q'}$$

$$\vdash \{P'\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{Q'\}$$

Here, $P$ is an invariant, meaning that it

- must hold initially;
- must be preserved by the loop body when $B$ is true; and

Moreover, to be useful, it must imply the desired postcondition when $B$ is false.

# Analysing the factorial implementation

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$

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

$$\{Y = x!\}$$

How does this program work?

# Observations about the factorial implementation

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$

**while** $X \neq 0$ **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

$$\{X = x \wedge X \geq 0 \wedge Y = 1\}$$

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

$$\{Y = x!\}$$

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

(We need $X \geq 0$ for $X!$ to make sense.)

# 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** (

$\{Y \times X! = x! \wedge X \geq 0 \wedge X \neq 0\}$

$\{(Y \times X) \times (X - 1)! = x! \wedge (X - 1) \geq 0\}$

$Y := Y \times X;$

$\{Y \times (X - 1)! = x! \wedge (X - 1) \geq 0\}$

$X := X - 1$

$\{Y \times X! = x! \wedge X \geq 0\}$

$)\{Y \times X! = x! \wedge X \geq 0 \wedge \neg(X \neq 0)\}$

$\{Y = x!\}$

# Fibonacci

# 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:

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

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?

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

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**

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

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

$Z$ 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
$Y = \mathit{fib}(Z) \wedge X = \mathit{fib}(Z - 1) \wedge Z > 0$.
(We need $Z > 0$ for $\mathit{fib}(Z - 1)$ to make sense.)

# Trying an invariant for the Fibonacci implementation

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

$$\{I\}$$

**while** $Z < N$ **do**

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

$$\{I \wedge \neg(Z < N)\}$$

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

Take $I \equiv Y = \mathit{fib}(Z) \wedge X = \mathit{fib}(Z - 1) \wedge Z > 0$.

Then we have to prove:

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

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**

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

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

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

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

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

# Proof outline for the loop of the Fibonacci implementation

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

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

**while** $Z < N$ **do**

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

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

$Y := X + Y;$

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

$X := Y - X;$

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

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

$Z := Z + 1$

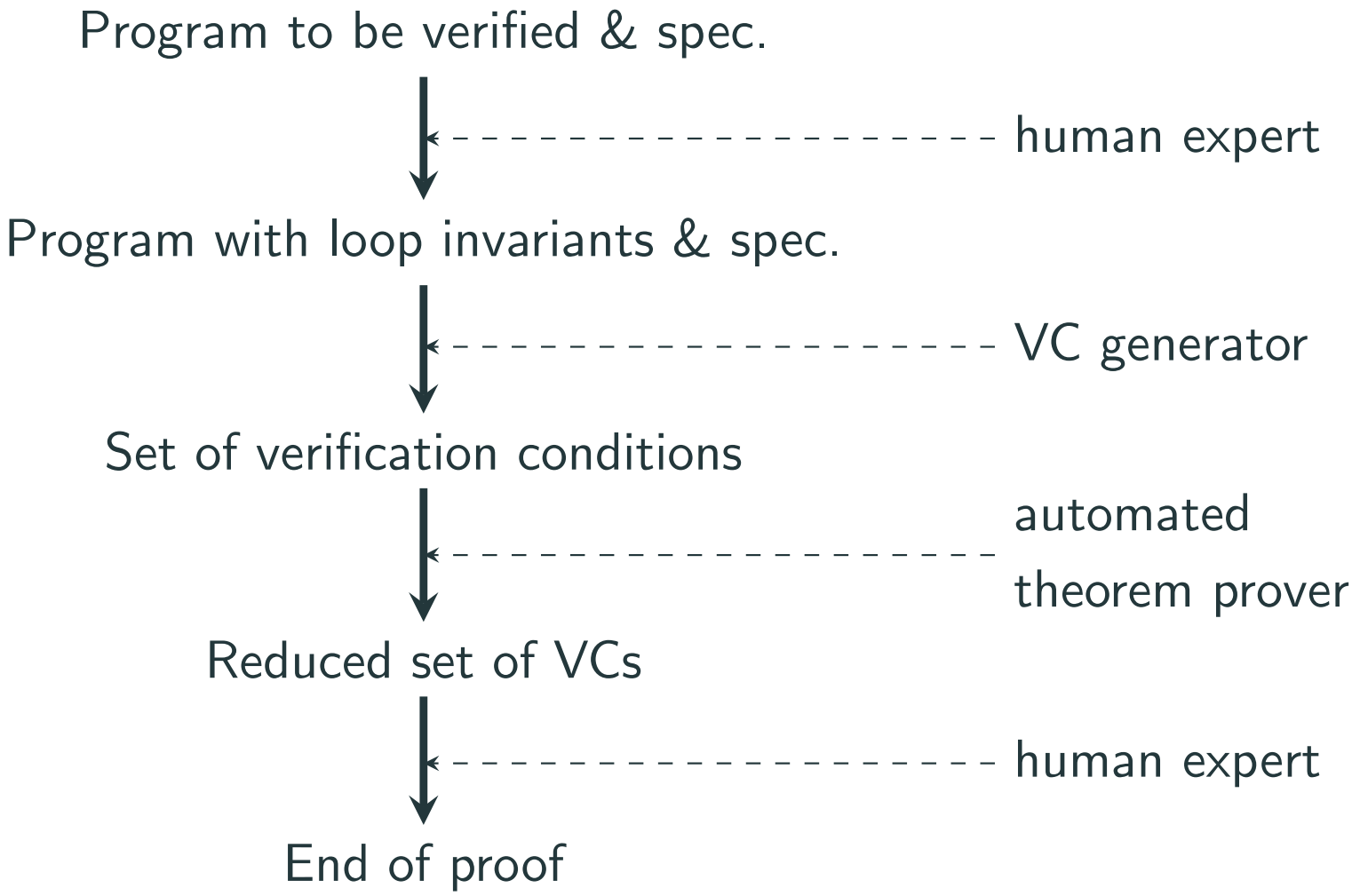
$\{Y = \mathit{fib}(Z) \wedge X = \mathit{fib}(Z-1) \wedge Z > 0 \wedge Z \leq N \wedge N = n\})$

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

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

# Verification condition generation

# Architecture of a verifier



# Verification condition generation

Finding invariants is difficult (as we will see in the next lecture).

However, we can write a simple recursive function $VC$ that takes a precondition $P$, an annotated program $\mathcal{C}$ in which loop invariants are provided as annotations, and a postcondition $Q$, and returns a set of assertions (called "verification conditions") such that, if they all hold, then $\{P\}\ |\mathcal{C}|\ \{Q\}$ holds (where $|\mathcal{C}|$ is $\mathcal{C}$ without the annotations).

Formally,

$$\forall \mathcal{C}, P, Q.\, (\forall R \in VC(P, \mathcal{C}, Q).\ \vdash_{FOL} R) \Rightarrow (\vdash \{P\}\ |\mathcal{C}|\ \{Q\})$$

# Summary

We have used Hoare logic to verify a few simple examples, and seen how finding invariants is the core difficulty.

Writing out full proof trees or even proof outlines by hand is tedious and error-prone, even for simple programs.
However, the trivia can be mechanised, leaving only finding invariants and proving difficult implications to the user.

In the next lecture, we will formalise the intuitions we gave in the first lecture, and prove soundness of Hoare logic.