Hoare Logic

Program Verification

Your Name

July 31, 2025

Outline

Syntax of the Language

Based on Backus-Naur Form (BNF)

Expressions:

$$E ::= N \mid V \mid E_1 + E_2 \mid E_1 - E_2 \mid E_1 \times E_2 \mid \ldots$$

Boolean expressions:

$$B ::= \mathbf{T} \mid \mathbf{F} \mid E_1 = E_2 \mid E_1 \leq E_2 \mid \ldots$$

Commands:

$$\begin{array}{rcl} C & ::= & V := E \\ & \mid & C_1;\ C_2 \\ & \mid & \mathrm{IF}\ B\ \mathrm{THEN}\ C_1\ \mathrm{ELSE}\ C_2 \\ & \mid & \mathrm{WHILE}\ B\ \mathrm{DO}\ C' \end{array}$$

Example Programs - 1

Illustrating the language syntax

Factorial of a number ‘n‘

This program computes $n!$ and stores the result in the variable ‘fact‘. It assumes the variable ‘n‘ holds a non-negative integer. The body of the ‘while‘ loop is a sequence of two assignment commands.

```
fact := 1;
i := n;
while i > 0 do
    fact := fact * i;
    i := i - 1
```

Example Programs - 2

Maximum of two numbers ‘x‘ and ‘y‘

This program uses a conditional statement to find the maximum of two numbers, ‘x‘ and ‘y‘, and stores the result in ‘max‘.

```
if x <= y then
    max := y
else
    max := x
```

What is a Program Specification?

The Contract

A program specification acts as a formal contract. It precisely describes the expected behavior of a piece of code.

- It does **not** describe *how* the program works.
- It **does** describe *what* the program must accomplish.

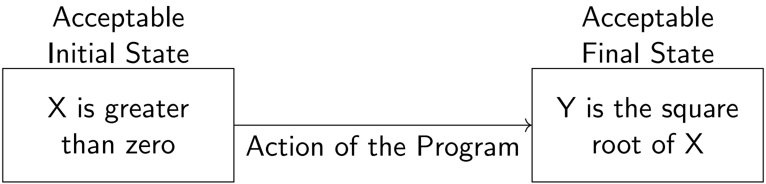
Key Components

A specification consists of two main parts:

- **Precondition:** A condition that must be true *before* the program is executed.
- **Postcondition:** A condition that is guaranteed to be true *after* the program terminates.

Visualizing a Specification

From Initial to Final State



Hoare's Notation

Historical Context

C.A.R. Hoare introduced the following notation called a **partial correctness specification** for specifying what a program does:

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

Components

- C is a command (a program or program fragment)
- P and Q are conditions on the program variables used in C
- P is called the **precondition**
- Q is called the **postcondition**

The Precondition (P)

Acceptable Initial State

The **precondition** defines the set of initial states for which the program is guaranteed to work correctly.

- It's an assumption about the values of program variables before execution.
- If the precondition is not met, the program has no obligations. It can crash, loop forever, or produce a wrong answer.
- Note: Reasoning about memory layout and heap requires *Separation Logic*, an extension of Hoare Logic.

Example

For a program that calculates the square root of X:

Informal: "X is greater than zero"

Formal: $\{X > 0\}$

The Postcondition (Q)

Acceptable Final State

The **postcondition** describes the state of the program after it has finished executing.

- It's the "promise" or "guarantee" of the specification.
- It typically relates the final values of variables to their initial values.

Example

For the square root program:

Informal: "Y is the square root of X"

Formal: $\{Y \times Y = X \wedge Y \geq 0\}$

(Note: we relate the final value of Y to the initial value of X).

Writing Conditions

Mathematical Notation

Conditions on program variables will be written using standard mathematical notations together with **logical operators**:

- $\wedge$ (and)
- $\vee$ (or)
- $\neg$ (not)
- $\Rightarrow$ (implies)

Example

Some example conditions:

- $x > 0 \wedge y \geq 0$ (x is positive AND y is non-negative)
- $x = 0 \vee y = 0$ (x equals zero OR y equals zero)
- $x > 0 \Rightarrow x^2 > 0$ (if x is positive, then x squared is positive)

Formal Specification: The Hoare Triple

Combining Pre- and Postconditions

Hoare Logic provides a formal notation to write specifications, called a **Hoare Triple**.

$$\{P\}\ S\ \{Q\}$$

This is read as:

> *If the precondition P is true before executing the program S, and if S terminates, then the postcondition Q will be true afterward.*

Example (Square Root Specification)

Combining our previous examples, the specification for a square root program S is:

$$\{X > 0\}\ S\ \{Y \times Y = X \land Y \geq 0\}$$

Here, S is the placeholder for the actual program code (the "Action").

Evolution of Notation

Historical Note

Hoare's original notation was $P\ \{C\}\ Q$ not $\{P\}\ C\ \{Q\}$, but the latter form is now more widely used.

Alternative Notations

You may encounter different notations in the literature:

- Original: $P\ \{C\}\ Q$
- Modern: $\{P\}\ C\ \{Q\}$
- Some texts: $\{P\}\ C\ \{Q\}$ (without special formatting)

All represent the same concept: a partial correctness specification.

Partial Correctness

What is Partial Correctness?

A Hoare triple $\{P\}\ C\ \{Q\}$ expresses **partial correctness**:

> *If the precondition P is true before executing command C, and if C terminates, then the postcondition Q will be true after execution.*

Important: Termination Not Guaranteed

Partial correctness does **not** guarantee that the program terminates!

- It only says what must be true *if* the program terminates
- A program that loops forever can still be partially correct
- Total correctness = Partial correctness + Termination

Reading Hoare Triples

How to Read $\{P\}\ C\ \{Q\}$

The triple $\{P\}\ C\ \{Q\}$ can be read as:

1. "If P is true, then after C executes, Q will be true"
2. "C transforms states satisfying P into states satisfying Q"
3. "Starting from P, command C establishes Q"

Example (Simple Assignment)

$\{x = 5\}\ y := x + 1\ \{y = 6\}$
This reads as: "If x equals 5 before the assignment, then y will equal 6 after the assignment."

Meaning of Hoare's Notation

Formal Definition

$\{P\}\ C\ \{Q\}$ is true if:

- whenever C is executed in a state satisfying P
- and *if* the execution of C terminates
- then the state in which C terminates satisfies Q

Example (Assignment Command)

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

- P is the condition that the value of X is 1
- Q is the condition that the value of X is 2
- C is the assignment command $X := X + 1$ (i.e. 'X becomes X+1')

Truth and Falsity of Hoare Triples

Example (True Triple)

$\{X = 1\}\ X := X + 1\ \{X = 2\}$ is **true**

Why? Starting from a state where $X = 1$, executing $X := X + 1$ results in $X = 2$.

Example (False Triple)

$\{X = 1\}\ X := X + 1\ \{X = 3\}$ is **false**

Why? Starting from $X = 1$, executing $X := X + 1$ results in $X = 2$, not $X = 3$.

Key Insight

A Hoare triple is a mathematical statement that can be either true or false. It makes a claim about what happens when a program executes.

Hoare Logic and Verification Conditions

What is Hoare Logic?

Hoare Logic is a **deductive proof system** for Hoare triples $\{P\}\ C\ \{Q\}$

- Provides axioms and inference rules for proving program correctness
- Forms the theoretical foundation for program verification

Direct Verification with Hoare Logic

Advantages:

- Original proposal by Hoare
- Provides complete formal proofs

Disadvantages:

- Tedious and error-prone for humans
- Impractical for large programs
- Requires detailed manual proof construction

Verification Conditions

Definition: What is a Verification Condition?

A **verification condition** is a mathematical formula (without program constructs) whose truth implies the correctness of a program.

- Generated from Hoare triples by analyzing the program structure
- Expressed purely in terms of logic and mathematics
- No references to program execution or state changes

Verification Conditions -2

Modern Approach: Verification Conditions

Can ‘compile’ proving $\{P\}\ C\ \{Q\}$ to **verification conditions**

- More natural for automated reasoning
- Basis for computer-assisted verification
- Separates program logic from mathematical reasoning

Verification Conditions -3

Key Property

Proof of verification conditions is **equivalent** to proof with Hoare Logic

- Hoare Logic can be used to *explain* verification conditions
- Both approaches prove the same correctness properties
- Verification conditions are more amenable to automation

Verification Condition Example

Example (Simple Verification Condition)

To prove $\{x > 0\}\ y := x + 1\ \{y > 1\}$:

Step 1: Analyze what the program does

- The assignment $y := x + 1$ sets y to the value of $x + 1$

Step 2: Generate the verification condition

- We need: if $x > 0$ initially, then $y > 1$ after assignment
- Since y will equal $x + 1$, we need: $x > 0 \Rightarrow (x + 1) > 1$

Step 3: The verification condition is:

$$x > 0 \Rightarrow (x + 1) > 1$$

This is a pure mathematical statement that can be proved using algebra, without any reference to program execution!