Hoare triples I & II

Farzaneh Derakhshan
based on material by Stefan Muller, Jim Sasaki, and Mike Gordon

CS 536: Science of Programming, Fall 2023
Lectures 7 and 8

1 A brief history

Hoare Logic, sometimes called Floyd-Hoare Logic, was developed based on the work of two computer scientists – Robert Floyd and Tony Hoare. They were independently seeking an answer to a fundamental question: *How do we know that a program is correct?* In 1969, in his seminal paper "Assigning Meaning to Programs," Robert Floyd formalized definitions of the meaning of programs using flowcharts. Twenty years later, In 1989, Tony Hoare published a paper called "An Axiomatic Basis for Computer Programming" to reason about the correctness of computer programs. Instead of using flowcharts, he formalized the meaning of programs using Hoare triples and described a set of logical rules to reason about the triples. Hoare logic we know today uses the same notation and rules as in Hoare's paper but is also due to the contributions of Floyd to the underlying ideas.

2 Hoare's notation

In his seminal work, Hoare introduced the notation

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

for specifying what a program does [1]. This notation is called Hoare triple since it consists of three main parts, P, S, and Q:

- S is a program written in our programming language.
- P and Q are formulas in the predicate logic. They state properties of the starting and ending states.

We interpret $\{P\}S\{Q\}$ as "*if we start with a state that satisfies P, and run S until it terminates, then the resulting state satisfies Q.*" In other words, P is the property we *assume* to be true *before running* S, and Q is the property we *assert* to be true *after running* S. Usually, P is called the *preconditon* and Q the *postcondition* [2].

In particular, when σ is the start state, we write $\sigma \vDash \{P\}S\{Q\}$ and read it as σ *satisfies the triple* $\{P\}S\{Q\}$. More formally, $\sigma \vDash \{P\}S\{Q\}$ is defined as

- if $\sigma \vDash P$, and (if the start state σ satisfies P, and)
- $\langle S, \sigma\rangle \to^* \langle \mathsf{skip}, \sigma'\rangle$, then (the configuration $\langle S, \sigma\rangle$ terminates in the final state σ', then)
- $\sigma' \vDash Q$. (the resulting state σ' satisfies Q.)

We call a triple $\{P\}S\{Q\}$ *valid* and write $\vDash \{P\}S\{Q\}$ if any state σ satisfies the triple, i.e., for all σ we have $\sigma \vDash \{P\}S\{Q\}$.

[1] In fact, the original Hoare notation is $P\{S\}Q$, but now $\{P\}S\{Q\}$ is used more widely.
[2] Brackets {} are not part of the conditions.

3 Examples

To understand the concept of Hoare triple better, let's look at a few examples.

3.1 Square root program

The function $sqrt(x)$, given an integer x, returns the square root of x. If x is not a perfect square, then the function returns $floor(\sqrt{x})$.

Example 1. $\{x \geq 0\}\, y := sqrt(x)\, \{y^2 = x\}$ is a Hoare triple – the program (S) is $y := sqrt(x)$, the precondition (P) is $x \geq 0$, and the postcondition (Q) is $y^2 = x$.

To prove that a start state σ satisfies the triple $\{x \geq 0\}\, y := sqrt(x)\, \{y^2 = x\}$ we need to show

$$\texttt{if } \sigma \vDash x \geq 0, \texttt{ and } \langle y := sqrt(x), \sigma\rangle \rightarrow^* \langle \mathsf{skip}, \sigma'\rangle, \texttt{ then } \sigma' \vDash y^2 = x.$$

Example 2. Let's say that our start state is $\sigma_1 = \{x = -1\}$. Does σ_1 satisfies the triple $\{x \geq 0\}\, y := sqrt(x)\, \{y^2 = x\}$ from Example 1?

Yes! The precondition does not hold in the start state ($\sigma_1 \nvDash \{x \geq 0\}$), so we don't need to prove anything about the postcondition.

Note 1. The triple says nothing if the precondition (P) is false in the start state.

Example 3. Now let's pick a different start state $\sigma_2 = \{x = 3\}$. Does σ_2 satsify the triple from Example 1? This time the answer is No! – The start state satisfies the precondition ($\sigma_2 \vDash x \geq 0$), and the configuration terminates $\langle y := sqrt(x), \sigma_2\rangle \rightarrow^* \langle \mathsf{skip}, \sigma_2[y \mapsto 1]\rangle$. So we have to show that the resulting state satisfies the postcondition. But we know that $\sigma_2[y \mapsto 1] \nvDash y^2 = x$, because $1^2 \neq 3$. Our triple has a bug!

Note 2. If the start state satisfies the precondition and the program terminates, then the resulting state has to satisfy the postcondition. Otherwise the start state does not satisfy the triple.

In Example 3 we found a state that does not satisfy the triple $\{x \geq 0\}\, y := sqrt(x)\, \{y^2 = x\}$. It means that the triple is not valid. Later in the Example 6 we build a valid triple for the program $y := sqrt(x)$.

3.2 Factorial program

Remember the Factorial program from Lecture 5. Factorial of x, assuming x has a positive value $n \geq 0$, calculates $r = n!$. Let's put our program (S_1) to be the Factorial of x:

$$\begin{array}{rcl} S_1 & = & i := x \\ & & r := \overline{1}; \\ & & \mathsf{while}\ i \geq \overline{1} \\ & & \mathsf{do} \\ & & \quad r := r * i; \\ & & \quad i := i - \overline{1} \\ & & \mathsf{od} \end{array}$$

Example 4. $\{x \geq 0\}\, S_1\, \{r = x!\}$ is a Hoare triple. S_1 is the Factorial of x, the precondition is $x \geq 0$, and the postcondition is $r = x!$. This Hoare triple is valid, meaning that for any start state σ, we have $\sigma \vDash \{x \geq 0\}\, S_1\, \{r = x!\}$. In Lecture 8, we will provide the rules with which we can prove validity of triples; for now, you can convince yourself that this triple is indeed valid by trying out different start states.

Note 3. The rules of predicate logic are not enough for proving validity of a Hoare triple. Hoare introduced a set of rules, called Hoare Logic, to reason about these triples. We'll study these rules soon.

Example 5. We can form different triples for one program. For example, $\{\mathbf{T}\}\, S_1\, \{r = x!\}$ also is a Hoare triple reasoning about S_1 (the Factorial of x). Here the precondition is $\mathbf{T}$, meaning that there is no extra condition on the start state (remember that $\mathbf{T}$ is a tautology, so for any state σ, we have $\sigma \vDash \mathbf{T}$).

This Hoare triple is not a valid one though. For example, for $\sigma_3 = \{x = -1\}$, we have $\sigma_3 \vDash \mathbf{T}$, and $\langle S_1, \sigma_3\rangle \to^* \langle \mathsf{skip}, \{x = -1, r = 1\}\rangle$. But $\{x = -1, r = 1\} \nvDash r = x!$.

Note 4. $\{\mathbf{T}\}S\{Q\}$ says that whenever S terminates, Q holds.

3.3 Ghost variables

Recall the square root program $y := sqrt(x)$ from Example 1. The triple that we introduced in Example 1 was not a valid one. Example 5 shows a valid triple that reasons about the square root program using a ghost variable.

Example 6. $\vDash \{x = k^2\} y := sqrt(x) \{y = |k|\}$

We call k a ghost variable: *a variable that does not occur in the program, but is used in the precondition or postcondition.*

Exercise 1. The factorial program in Example 4 is slightly different from what we presented in Lecture 5. In particular, in Example 4 we introduce a variable i for the loop counter and keep the value of x intact. Now, let's consider the original version of the Factorial program from Lecture 5, which is shown below:

$$
\begin{array}{rcl}
S_1' & = & r := \overline{1}; \\
 & & \mathsf{while}\ x \geq \overline{1} \\
 & & \mathsf{do} \\
 & & \quad r := r * x; \\
 & & \quad x := x - \overline{1} \\
 & & \mathsf{od}
\end{array}
$$

Is the triple from Example 4 still valid for S_1'? Using a ghost variable write a valid triple for reasoning about this program.

3.4 Invariants

A condition which is true before and after running the program is called an invariant.

Example 7. The triple below says that if the precondition $x = k!$ holds in the start state, and the program terminates, then the condition $x = k!$ continutes to hold in the final state.

$$\vDash \{x = k!\}\, x := x * (k + 1); k := k + 1\, \{x = k!\}$$

We call the condition $x = k!$ the invariant.

Note 5. The condition $x = k!$ remains true in the final state, even though the value of x and k change by running the program.

4 How to fix a triple to make it valid?

Consider the triple $\{x > 0\}\, x := x - 1\, \{x > 0\}$. This triple is not valid. For example, it is not satisfied in the initial state $\sigma_4 = \{x = 1\}$: we have $\sigma_4 \vDash x > 0$, and $\langle x := x - 1, \sigma_4\rangle \to^* \langle \mathsf{skip}, \{x = 0\}\rangle$. But $\{x = 0\} \nvDash x > 0$. But, how can we fix this triple? There are three options:

1. Make the *precondition* stronger (more restrictive):

$$\vDash \{x > 1\}\, x := x - 1\, \{x > 0\}.$$

2. Make the *postcondition* weaker (less restrictive):

$$\vDash \{x > 0\}\, x := x - 1\, \{x \geq 0\}.$$

3. Fix the program:

$$\vDash \{x > 0\}\, \mathsf{if}\ x > 1\ \mathsf{then}\ x := x - 1\ \mathsf{else}\ x := 1\ \mathsf{fi}\, \{x > 0\}.$$

5 Partial vs. total correctness

The triple $\{P\}S\{Q\}$ is called partial correctness triple – It is *partial* because for $\{P\}S\{Q\}$ to be valid it is not necessary that S terminates when started in a state satisfying P. It is only required that if S terminates, then Q holds.

A stronger kind of Hoare triple $[P]S[Q]$ is called *total correctness*. We interpret the total correctness triple as "*if we start with a state that satisfies P, then S terminates, and the resulting state satisfies Q.*" In other words, when started in a state satisfying P, it is required that S terminates and the resulting state satisfies Q.

The relationship between partial and total correctness can be informally expressed by the equation:

$$\text{Total correctness} = \text{Partial correctness} + \text{Termination}.$$

Formally, when σ is the start state, we define $\sigma \vDash [P]S[Q]$ as

- if $\sigma \vDash P$, then (if the start state σ satisfies P, then)
- $\langle S, \sigma\rangle \to^* \langle \mathsf{skip}, \sigma'\rangle$, and (the configuration $\langle S, \sigma\rangle$ terminates in the final state σ', and)
- $\sigma' \vDash Q$. (the resulting state σ' satisfies Q.)

Similar to the partial correctness, we call a triple $[P]S[Q]$ *valid* and write $\vDash [P]S[Q]$ if any state σ satisfies the triple, i.e., for all σ we have $\sigma \vDash [P]S[Q]$.

5.1 Examples

Example 8. *A program that never terminates successfully – always diverges:*

The partial correctness triple

$$\{\mathbf{T}\}\, \mathsf{while\ true\ do}\ \mathit{skip}\ \mathsf{od}\, \{x > 1 \wedge x < 1\}$$

is valid for this diverging program – the program never terminates, so there is nothing to be proved for the partial correctness triple.

But, the corresponding total correctness triple

$$[\mathbf{T}]\, \mathsf{while\ true\ do}\ \mathit{skip}\ \mathsf{od}\, [x > 1 \wedge x < 1]$$

is not valid. In fact, there is no initial state σ that satisfies this total correctness triple (**Exercise 2.** Can you explain why?).

We can generalize this example to any preconditon P and postcondition Q:

The partial correctness triple is always valid, i.e.,

$$\vDash \{P\}\, \mathsf{while\ true\ do}\ \mathit{skip}\ \mathsf{od}\, \{Q\}.$$

And for any initial state σ that satisfies the precondition P, we have

$$\sigma \nvDash [P]\, \mathsf{while\ true\ do}\ \mathit{skip}\ \mathsf{od}\, [Q].$$

Exercise 3. Can we say that for any preconditon P and postcondition Q, the total correctness triple is not valid, i.e., $\nvDash [P]\, \mathsf{while\ true\ do}\ \mathit{skip}\ \mathsf{od}\, [Q]$?

Example 9. *A program that never terminates successfully – always returns an error:*

Similar to Example 8, for any precondition P and postcondition Q, the partial correctness triple is always valid, i.e.,

$$\models \{P\}\, y := sqrt(\overline{-1})\, \{Q\}.$$

And for any initial state σ that satisfies the precondition P, we have

$$\sigma \not\models [P]\, y := sqrt(\overline{-1})\, [Q].$$

Exercise 4. Can we say that for any preconditon P and postcondition Q, the total correctness triple is not valid, i.e., $\not\models [P]\, y := sqrt(\overline{-1})\, [Q]$?

Example 10. *(Contradictory precondition)* If the precondition is a contradictory predicate, then both partial and total correctness triples are valid: $\models \{F\}S\{Q\}$ and $\models [F]S[Q]$.

Example 11. *(Tautology postcondition)* If the postcondition is a tautology, then the partial correctness triple is valid for any precondition P and any program S, i.e., $\models \{P\}S\{\mathbf{T}\}$ – The postcondition requires nothing, so the triple does not require anything to be proved about the program. However, the same is not true for the total correctness triple, i.e., $[P]S[\mathbf{T}]$ is not necessary valid – The postcondition requires nothing, but still we have to prove that S is terminating.

Exercise 5. Provide a program S' and a precondition P' such that $\models [P']S'[\mathbf{T}]$. Now, provide a program S'' and a precondition P'' such that $\not\models [P']S'[\mathbf{T}]$

5.2 Equivalent notations

We can define partial and total correctness equivalently as:

- $\models \{P\}S\{Q\}$ means that for any start state σ,
  - if $\sigma \models P$ and — (if the start state σ satisfies P and)
  - $\sigma' \in M(s, \sigma)$ for $\sigma' \neq \perp$, then — ($\langle S, \sigma\rangle$ terminates in the final state σ', then)
  - $\sigma' \models Q$. — (the resulting state σ' satisfies Q.)
- $\models [P]S[Q]$ means that for any start state σ,
  - if $\sigma \models P$, then — (if the start state σ satisfies P, then)
  - $\perp \notin M(s, \sigma)$ and there is a $\sigma' \in M(S, \sigma)$ such that — ($\langle S, \sigma\rangle$ terminates in the final state σ' and)
  - $\sigma' \models Q$. — (the resulting state σ' satisfies Q.)

We can reverse them as:

- $\not\models \{P\}S\{Q\}$ means that there exists a start state σ,
  - $\sigma \models P$ and — (the start state σ satisfies P and)
  - $\sigma' \in M(s, \sigma)$ for $\sigma' \neq \perp$, but — ($\langle S, \sigma\rangle$ terminates in the final state σ', but)
  - $\sigma' \not\models Q$. — (the resulting state σ' does not satisfy Q.)
- $\not\models [P]S[Q]$ means that there exists a start state σ such that,
  - $\sigma \models P$, and either — (the start state σ satisfies P, and either)
    - $\perp \in M(s, \sigma)$ or — ($\langle S, \sigma\rangle$ gives an error or)
    - there is a $\sigma' \in M(s, \sigma)$ for $\sigma' \neq \perp$ such that $\sigma' \not\models Q$. — (the resulting state σ' does not satisfy Q.)

For Examples 12-14, let the program S_2 be $\mathsf{while}\ x \neq 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}$

Example 12. $\models [x \geq 0]S_2[x = 0]$, i.e., if we start in a state where x is greater than or equal to zero, then the loop is guaranteed to terminate in a state satisfying $x = 0$.

Example 13. $\vDash \{x = -1\}S_2\{x = 0\}$, but $\nvDash [x = -1]S_2[x = 0]$. The partial correctness triple is valid, but its corresponding total correctness is not. Because the program diverges for the start states $\{x = -1\}$. Also note that the partial correctness triple would hold if we substitute any predicate Q for $x = 0$, i.e., $\vDash \{x = -1\}S_2\{Q\}$.

Example 14. $\vDash \{\mathbf{T}\}S_2\{x = 0\}$ but $\nvDash [\mathbf{T}]S_2[x = 0]$. The triple is partially correct but not totally correct because it diverges for at least one value of x.

6 Same Code, Different Conditions

The same piece of code can be annotated with conditions in different ways, and there's not always a "best" annotation. An annotation might be the most general one possible (we'll discuss this concept soon), but depending on the context, we might prefer a different annotation. Here, we give several examples of the same programs annotated with different preconditions and postcondition of various strengths (strength = generality).

- For Examples 15 and 16 [3], let the program S_3 be $\mathsf{while}\ x > 0\ \mathsf{do}\ x := x - 1\ \mathsf{od}$.

  Example 15. $\vDash [\mathbf{T}]S_3[x \leq 0]$

  Example 16. $\vDash [x = c_0]S_3[(c_0 \leq 0 \to x = c_0) \wedge (c_0 \geq 0 \to x = 0)]$. This refines the annotations of Example 15 using a ghost variable c_0 to get the "strongest" (most precise) postcondition possible.

- For Examples 17-21, let the program S_4 be $i := 0; s := 0$.

  We define $sum(x, y) = x + (x + 1) + (x + 2) + \cdots + y$. If $x > y$, let $sum(x, y) = 0$.

  Example 17. $\vDash \{\mathbf{T}\}S_4\{i = s \wedge s = 0\}$. This is the strongest (most precise) annotation for S_4.

  Example 18. $\vDash \{\mathbf{T}\}S_4\{i \geq 0 \wedge s \geq 0\}$. This is a strictly weaker (and therefore also correct) annotation. (**Exercise 6.** Prove that validity of Example 17 implies validity of Example 18.)

  Example 19. $\vDash \{\mathbf{T}\}S_4\{i = 0 \wedge s = 0 \wedge s = sum(0, i)\}$. This adds a trivial summation relationship to i and s.

  Example 20. $\vDash \{n \geq 0\}S_4\{i = 0 \wedge 0 \leq n \wedge s = 0 \wedge s = sum(0, i)\}$. This limits i to a range of values $0, \cdots n$ There's no way in the postcondition to know $n \geq 0$ unless we assume it in the precondition.

  Example 21. $\vDash \{n \geq 0\}S_4\{i \geq 0 \wedge i \leq n \wedge s = sum(0, i)\}$. The postcondition no longer includes $i = s = 0$ and is therefore weaker, which might seem like a disadvantage but will turn out to be an advantage later.

- The next two examples (Examples 22 and 23) relate to calculating the midpoint in binary search. Let S_5 to be $M := (L + R)/2$. Though the code is the same, whether the midpoint(M) is strictly between the left(L) and right(R) endpoints depends on whether or not the endpoints are nonadjacent.

  Example 22. $\vDash \{L < R \wedge L \neq R - 1\}S_5\{L < M < R\}$.

  Example 23. $\vDash \{L < R\}S_5\{L \leq M < R\}$

- In the next example, given two integers x and y and a function $f : int \to int$ that maps integers to integers, we search downward for a nonnegative x where $f(x)$ is less than or equal to y; we stop if x goes negative or we find an x with $f(x) \leq y$.

  Example 24.

  $\{x \geq 0\}$
  $\mathsf{while}\ x \geq 0 \wedge f(x) > y\ \mathsf{do}\ x := x - 1\ \mathsf{od}$
  $\{x < 0 \vee f(x) \leq y\}$

[3] S_3 is similar to S_2 in Examples 12-14, but we're changing the loop test so that it terminates immediately when x is negative.

Example 25. This is Example 24 rephrased as an array search; as long as we have a legal index i and $b[i]$ is not less that or equal to y, we move left. We stop if the index becomes illegal or we find an index with $b[i] \leq y$.

$$\{i \geq 0\}$$
$$\mathsf{while}\ i \geq 0 \wedge b[i] > y\ \mathsf{do}\ i := i - 1\ \mathsf{od}$$
$$\{i < 0 \vee b[i] \leq y\}$$

7 Conditions can have quantifiers

Remeber that P and Q are formulas from the predicate logic. So, they can also have quantifiers.

Example 7. The program $i := \overline{0}; \mathsf{while}\ i < size(a)\ \mathsf{do}\ x := x + sqrt(a[i])\ \mathsf{od}$ only terminates successfully if for all indices j of the array, we have $a[j] \geq 0$. We formalize this observation using a total correctness triple:

$$[\forall j.(0 \geq j < size(a) \rightarrow a[j] \geq 0)]$$
$$i := \overline{0}; \mathsf{while}\ i < size(a)\ \mathsf{do}\ x := x + sqrt(a[i])\ \mathsf{od}$$
$$[\mathbf{T}]$$