Hoare logic

Hoare logic (also known as **Floyd–Hoare logic** or **Hoare rules**) is a formal system with a set of logical rules for reasoning rigorously about the correctness of computer programs. It was proposed in 1969 by the British computer scientist and logician C. A. R. Hoare, and subsequently refined by Hoare and other researchers.[1] The original ideas were seeded by the work of Robert Floyd, who had published a similar system[2] for flowcharts.

Hoare Triple

The central feature of **Hoare logic** is the **Hoare triple**. A triple describes how the execution of a piece of code changes the state of the computation. A Hoare triple is of the form

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

where *P* and *Q* are *assertions* and *C* is a *command*. *P* is named the *precondition* and *Q* the *postcondition*: when the precondition is met, the command establishes the postcondition. Assertions are formulas in predicate logic.

Hoare logic provides axioms and inference rules for all the constructs of a simple imperative programming language. In addition to the rules for the simple language in Hoare's original paper, rules for other language constructs have been developed since then by Hoare and many other researchers. There are rules for concurrency, procedures, jumps, and pointers.

Partial and total correctness

Standard Hoare logic proves only partial correctness, while termination needs be proved separately. Thus the intuitive reading of a Hoare triple is: Whenever *P* holds of the state before the execution of *C*, then *Q* will hold afterwards, or *C* does not terminate. Note that if *C* does not terminate, then there is no "after", so *Q* can be any statement at all. Indeed, one can choose *Q* to be false to express that *C* does not terminate.

Total correctness can be also proven with an extended version of the While rule.

Rules

Empty statement axiom schema

The empty statement rule asserts that the **skip** statement does not change the state of the program, thus whatever holds true before **skip** also holds true afterwards.

$$\frac{}{\{P\}\ \mathbf{skip}\ \{P\}}$$

Assignment axiom schema

The assignment axiom states that after the assignment any predicate holds for the variable that was previously true for the right-hand side of the assignment:

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

Here $P[E/x]$ denotes the expression *P* in which all free occurrences of the variable *x* have been replaced with the expression *E*.

The assignment axiom means that the truth of $\{P[E/x]\}$ is equivalent to the after-assignment truth of $\{P\}$. Thus were $\{P[E/x]\}$ *true* prior to the assignment, by the assignment axiom, then $\{P\}$ would be *true* subsequent to which. Conversely, were $\{P[E/x]\}$ *false* prior to the assignment statement, $\{P\}$ must then be *false* consequently.

Examples of valid triples include:

- $\{x+1=43\}\ y := x+1\ \{y=43\}$
- $\{x+1\leq N\}\ x := x+1\ \{x\leq N\}$

The assignment axiom proposed by Hoare *does not apply* when more than one name may refer to the same stored value. For example,

$$\{y=3\}\ x := 2\ \{y=3\}$$

is not a true statement if *x* and *y* refer to the same variable, because no precondition can cause *y* to be 3 after *x* is set to 2.

Rule of composition

Hoare's rule of composition applies to sequentially-executed programs *S* and *T*, where *S* executes prior to *T* and is written *S;T*.

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

For example, consider the following two instances of the assignment axiom:

$$\{x+1=43\}\ y := x+1\ \{y=43\}$$

and

$$\{y=43\}\ z := y\ \{z=43\}$$

By the sequencing rule, one concludes:

$$\{x+1=43\}\ y := x+1; z := y\ \{z=43\}$$

Conditional rule

$$\frac{\{B\wedge P\}\ S\ \{Q\}\ ,\ \{\neg B\wedge P\}\ T\ \{Q\}}{\{P\}\ \mathbf{if}\ B\ \mathbf{then}\ S\ \mathbf{else}\ T\ \mathbf{endif}\ \{Q\}}$$

Consequence rule

$$\frac{P'\rightarrow\ P\ ,\ \{P\}\ S\ \{Q\}\ ,\ Q\rightarrow\ Q'}{\{P'\}\ S\ \{Q'\}}$$

While rule

$$\frac{\{P\wedge B\}\ S\ \{P\}}{\{P\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{done}\ \{\neg B\wedge P\}}$$

Here *P* is the loop invariant.

While rule for total correctness

$$\frac{<\ \text{is well-founded},\ [P\wedge B\wedge t=z]\ S\ [P\wedge t<z]}{[P]\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{done}\ [\neg B\wedge P]}$$

In this rule, in addition to maintaining the loop invariant, one also proves termination by way of a term, called the loop variant, here *t*, whose value strictly decreases with respect to a well-founded relation during each iteration. Note that, given the invariant *P*, the condition *B* must imply that *t* is not a minimal element of its range, for otherwise the premise of this rule would be false. Because the relation "<" is well-founded, each step of the loop is counted by decreasing members of a finite chain. Also note that square brackets are used here instead of curly braces to denote total correctness, i.e. termination as well as partial correctness. (This is one of various notations for total correctness.)

Examples

Example 1			
$\{x+1=43\}$	$y := x+1$	$\{y=43\}$	(Assignment Axiom)
			$(x+1=43 \Leftrightarrow x=42)$
$\{x=42\}$	$y := x+1$	$\{y=43 \wedge x=42\}$	(Consequence Rule)
Example 2			
$\{x+1 \leq N\}$	$x := x+1$	$\{x \leq N\}$	(Assignment Axiom)
			($x < N \implies x+1 \leq N$ for *x*, *N* with integer types)
$\{x < N\}$	$x := x+1$	$\{x \leq N\}$	(Consequence Rule)

References

[1] C. A. R. Hoare. " An axiomatic basis for computer programming (http://sunnyday.mit.edu/16.355/Hoare-CACM-69.pdf)". *Communications of the ACM*, 12(10):576–580,583 October 1969. doi:10.1145/363235.363259

[2] R. W. Floyd. " Assigning meanings to programs. (http://laser.cs.umass.edu/courses/cs521-621.Spr06/papers/Floyd.pdf)" Proceedings of the American Mathematical Society Symposia on Applied Mathematics. Vol. 19, pp. 19–31. 1967.

Further reading

- Robert D. Tennent. *Specifying Software* (http://www.cs.queensu.ca/home/specsoft/) (a textbook that includes an introduction to Hoare logic, written in 2002) ISBN 0-521-00401-2

External links

- Project Bali (http://isabelle.in.tum.de/Bali/) has defined Hoare logic-style rules for a subset of the Java programming language, for use with the Isabelle theorem prover
- KeY-Hoare (http://www.key-project.org/download/hoare/) is a semi-automatic verification system built on top of the KeY theorem prover. It features a Hoare calculus for a simple while language.
- j-Algo-modul Hoare calculus (http://j-algo.binaervarianz.de/index.php?language=en) — A visualisation of the Hoare calculus in the algorithm visualisation program j-Algo

Article Sources and Contributors

Hoare logic *Source*: http://en.wikipedia.org/w/index.php?oldid=418574210 *Contributors*: (:Julien:), Andy Fugard, BBar, Bobblewik, CarlHewitt, Cataphract, Chalst, Chinju, Deepmath, Dogah, Dwheeler, Eug, Ferris37, Frecklefoot, Gaius Cornelius, Gazpacho, Heathhunnicutt, Heron, Hooperbloob, Iskander s, Jdinolt, Jim Apple, Jpbowen, KoenDelaere, Leibniz, Luckyz, Lycurgus, MarkSweep, MathMartin, Mgreenbe, Mhss, Michiel Helvensteijn, Mukerjee, Oliver Pereira, Ordago, Ospalh, PabloStraub, Rangek, Ruud Koot, SlackerMom, Suruena, Tango, Tatzelworm, Tobias Bergemann, Topdeck, TrulyBlue, Vinhtantran, 49 anonymous edits

License

Creative Commons Attribution-Share Alike 3.0 Unported
http://creativecommons.org/licenses/by-sa/3.0/