Introduction to Hoare Logic

October 11, 2016

Abstract

Hoare Logic was created by C.A.R. Hoare in an attempt to put computer programming on the same logical foundations as mathematics. It consists of a set of axioms and rules of inference which can be used to show proofs of the properties of particular computer programs.

1 Introduction

I'm going to (quickly) talk about the programming language IMP, then we're going to see how Hoare logic makes it easier to derive proofs of programs.

IMP is a *very* basic imperative programming language which can only store and operate on numeric values.

2 Syntax

The abstract syntax of IMP is defined inductively as follows:

$s ::= \textbf{skip} \mid x := e \mid s; s \mid \textbf{if } e\ s\ s \mid \textbf{while } e\ s$
$e ::= c \mid x \mid e + e \mid e * e$
$c \in \mathbb{Z}$
$x \in \{x_1, x_2, ..., y_1, y_2, ..., z_1, z_2, ..., ...\}$

2.1 Heap

In order to store our variables somewhere, we need to introduce the concept of a *heap*, which is a total function from variables to constants. More formally, a heap is:

$H ::= \cdot \mid H, x \to c$

And the associated lookup function:

$$H(x) = \begin{cases} c & \textbf{if } H = H', x \to c \\ H'(x) & \textbf{if } H = H', y \to c', \textbf{ and } y \neq x \\ 0 & \textbf{if } H = \cdot \end{cases}$$

We can now associate *judgements* to our programs. The syntax "$H; e \Downarrow c$" is read "e evaluates to c under heap H".

3 Semantics

Now that we have a notion of the symbols that can be used, lets see what happens when we give them meaning!

$$\textbf{Constant: } \frac{}{H; c \Downarrow c}$$

$$\textbf{Variable: } \frac{}{H; x \Downarrow H(x)}$$

$$\textbf{Add: } \frac{H; e_1 \Downarrow c_1 \qquad H; e_2 \Downarrow c_2}{H; e_1 + e_2 \Downarrow c_1 + c_2}$$

$$\textbf{Multiply: } \frac{H; e_1 \Downarrow c_1 \qquad H; e_2 \Downarrow c_2}{H; e_1 * e_2 \Downarrow c_1 * c_2}$$

4 Deductive Logic

With the above, we have enough to prove (contrived) programs in IMP! For example, if we wanted to show that the program "$\cdot, y \to 4; (3 + y) + 5 \Downarrow 12$" is correct, we could simply apply deductive logic to see a complete derivation as follows:

$$\frac{\dfrac{\dfrac{}{\cdot, y \to 4; y \Downarrow 4} \qquad \dfrac{}{\cdot, y \to 4; 3 \Downarrow 3}}{\cdot, y \to 4; 3 + y \Downarrow 7} \qquad \dfrac{}{\cdot, y \to 4; 5 \Downarrow 5}}{\cdot, y \to 4; (3 + y) + 5 \Downarrow 12}$$

5 Hoare Logic

In 1969, C.A.R. Hoare noted that "[c]omputer programming is an exact science in that all the properties of a program and all the consequences of executing it can be found out from the text of the program itself by means of purely deductive reasoning."[2]

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:

$$\{P\}C\{R\}$$

The above is read *"If the assertion P is true before initiation of a program C, then the assertion R will be true on its completion"*.

As should be fairly evident, this appears (on the surface) to look similar to our IMP syntax – we have some precondition (the heap in IMP), a program (an expression in IMP), and a resulting postcondition (the resulting heap).

5.1 Rules

The original paper lays out the following rules[1]:

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

(where $P[E/x]$ denotes the assertion P in which each free occurrence of x has been replaced by the expression E)

$$\textbf{If Statements:} \quad \frac{\{B \wedge P\}S\{Q\} \qquad \{\neg B \wedge P\}T\{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ S\ \textbf{else}\ T\ \textbf{endif}\ \{Q\}}$$

$$\textbf{Sequencing:} \quad \frac{\{P\}S\{Q\} \qquad \{Q\}T\{R\}}{\{P\}S;T\{R\}}$$

$$\textbf{While Loops:} \quad \frac{\{P \wedge B\}S\{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{done}\ \{\neg B \wedge P\}}$$

5.2 A Note on Completeness

Due to the fact that a language (with some appropriate construct such as iteration) can create a program which will potentially never terminate, we need to prove termination conditions separately (as is done in the *Probabilistic Hoare-Style Logic* paper[1]). In general, this is complicated.

[1] The notation has changed since the original paper. The notation in this paper follows modern convention.

6 Probabilistic Hoare Logic

When moving to probabilistic Hoare logic, we need to be able to encode some idea of probability into our Hoare triples, such as $\{Pr(x = 1) > \frac{1}{2}\}c\{Pr(z = 3) = \frac{2}{3}\}$. In [3], the authors propose adding one more rule to the Hoare logic rules:

$$\textbf{Coin toss:} \quad \frac{y \text{ free in } P}{\{P\}y := \texttt{toss}(p)\{P \triangleleft_p^y\}}$$

(where $P \triangleleft_p^y$ is read "P conditioned on y with probability p")

The difficulty with probabilistic Hoare logic is almost entire tied up in the *"if"* and *"while"* rules, particularly when the guard is probabilistic. Quite frankly, I don't understand these sections yet.

The paper [1] introduces a new syntax "$s \oplus_\rho s'$" which says that s will occur with probability ρ, and s' will occur with probability $1 - \rho$. This introduces a different rule than above, but it retains a similar feel:

$$\textbf{Probabilistic:} \quad \frac{\{P\}C\{R\} \qquad \{P\}C'\{R'\}}{\{P\}C \oplus_\rho C'\{R \oplus_\rho R'\}}$$

6.1 Example

Using deductive logic, we can prove the following simple program which contains some probabilistic element:

$$\{Pr(x = 1) = 1\}(x := x + 1) \oplus_{\frac{1}{2}} (x = x + 2)\{Pr(x = 2) = \frac{1}{2} \wedge Pr(x = 3) = \frac{1}{2}\}$$

And the complete derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{x+1=2\}x := x+1\{x=2\}}}{\{x=1\}x := x+1\{x=2\}} \qquad \dfrac{\dfrac{}{\{x+2=3\}x := x+2\{x=3\}}}{\{x=1\}x := x+2\{x=3\}}}{\{x=1\}(x := x+1) \oplus_{\frac{1}{2}} (x = x+2)\{x = 2 \oplus_{\frac{1}{2}} x = 3}}{\{x=1\}(x := x+1) \oplus_{\frac{1}{2}} (x = x+2)\{Pr(x=2) = \frac{1}{2} \wedge Pr(x=3) = \frac{1}{2}}$$

References

[1] Jerry den Hartogs. A probabilistic hoare-style logic, 2002.

[2] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[3] Robert Rand and Steve Zdancewic. VPHL: A verified partial-correctness logic for probabilistic programs. *Electronic Notes in Theoretical Computer Science*, 319:351–367, 2015.