How to use Hoare Logic to verify program correctness

Huajun Lu
Nanjing University
Nanjing, Jiangsu, CHN

Abstract

Program correctness is of great importance when programming. Hoare Logic is a powerful tool to verify program correctness. It can establish a transformation between program code and logic formulas.

1 Introduction

Computer programming is an exact science, so when a programmer is programming, it is important to verify that the code is correct. So it is desirable to use a valid logic to verify the program correctness. In fact, all the consequences of excuting a program can be found out "by means of purely deductive reasoning." [1], and Hoare Logic is such a powerful dedutive logic system based on predicate logic. On its basis, we can establish a transformation between code and logic formulas, thus ensuring that our programs are validated. To ensure the process of verification of program is valid, the axioms and rules of inference must be elucidated clearly. Therefore, I will next give the definitions of Hoare Triple, the Assignment Axiom, the Conditional Rule, the Sequencing Rule, and the Iteration Rule.

2 Axioms and Rules

2.1 Hoare triple

Hoare introduced the following notation to specify what a program does:

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

In many cases, the validity of the results of a program will depend on the initial conditions. So before proving the result is correct, we must introduce Hoare Triple to describe the transformation between precondition to postcondition after excuting a piece of code. It is the formalized representation of the verification of a program. In this triples, C means a **command**, which is a code block. P is the **pre-condition**, and Q is the **post-condition**. The meaning of the triple is that, assuming C is executable and executed in a state satisfying P, when C is executed, the state will satisfy Q.
For example:

$$\{x = y\}z = x\{y = z\} : true$$
$$\{x = 1, y = 1\}y = 0\{x = y\} : false$$

2.2 The Assignment Axiom

Assignment is the most characteristic and basic feature of a program, It distinguishes computer science from any other branch of mathematics.

$$Definition: \quad \vdash \{Q[E/V]\}V := E\{Q\}$$

$\vdash$ is the notation means that the proposition can be syntactically derived. := means assignment, and $Q[E/V]$ means the result of replacing all occurrences of V in Q by E.
Here V is a variable identifier, E is an identified expression, Q is any statement. The axiom means the final value of V is the value of E in the initial state after the assignment.
Example

```
1  X:=Y+1
```

And the code above is equal to the triple below:

$$\vdash \{Y + 1 = V\}X = Y + 1\{X = V\}$$

2.3 The Sequencing Rule

The deduction system requires a valid rule of inference which permits the deduction of new theoroms from one proved theorem or axiom to new theorems. So as a result we need to introduce the Sequencing Rule.

$$Definition: \quad \frac{\vdash \{P\}C_1\{Q\}, \vdash \{Q\}C_2\{R\}}{\vdash \{P\}C_1; C_2\{R\}}$$

Here $\frac{P_1}{P_2}$ means that, if the correctness of P_1 is ensured, P_2 can be proved correct. And this triple means that, after the execution of C_1 and C_2, state P can produce Q and R sequentially.
Example

```
1  R:=X
2  Y:=R
```

And the expected result $Y = X$ can be verified sequentially with the triple below:

$$\frac{\vdash \{X = X\}R = X\{R = X\}, \vdash \{R = X\}Y = R\{Y = X\}}{\vdash \{X = X\}R = X; Y = R\{Y = X\}}$$

2.4 The Conditional Rule

In some cases, the argumentation process will branch out. This means that we need to discuss the situation separately. And then the Conditional Rule will have to be considered.

$$Definition: \quad \frac{\vdash \{P \wedge S\}C_1\{Q\}, \vdash \{P \wedge \neg S\}C_2\{Q\}}{\vdash \{P\}IF\ S\ THEN\ C_1\ ELSE\ C_2\{Q\}}$$

Here $\wedge$ means "and", $\vee$ means "or", $\neg$ means not. In initial state P is true, and if S is true then execute C_1, if S is false then executed C_2. After execution, Q is true.
Example

```
1  IF X <= Y THEN
2      Z := X
3  ELSE
4      Z := Y
```

The code above is to assign the greater value of X and Y to Z. And we can formalize the code and verify the correctness as the proposition below:

$$\frac{\vdash\{X\leq Y\}Z{:=}X\{Z{=}min\{X,Y\}\},\vdash\{\neg(X\leq Y)\}Z{:=}Y\{Z{=}min\{X,Y\}\}}{\vdash\{True\}IF\ X\leq Y\ THEN\ Z{:=}X\ ELSE\ Z{:=}Y\{Z{=}min\{X,Y\}\}}$$

2.5 The Iteration Rule

Since it is inevitable in programming that we want to make the computer perform a large number of repetitive steps, the cyclic execution is of great importance when coding. How to ensure the correctness of a piece of cyclic code requires the introduction of the Iteration Rule.

$$Definition: \quad \frac{\vdash \{P \wedge S\}C\{P\}}{\vdash \{P\}WHILE\ S\ DO\ C\{P \wedge \neg S\}}$$

It should be mentioned that P is the **invariant** of the whole While-Command, which means P is always true while this part of code is being excuted. And S is the condition to check whether the loop should be terminated or continue. When executing this piece of code, the computer first tests the condition S. If S is false, C is omitted, and execution of the cyclic program is complete. Otherwise, C is executed and S is tested again. This action is repeated until S is found to be false. So when a While-Command terminates, then the state $P \wedge S$ must be false.

Example

```
1  X := 1
2  WHILE X <= 7 DO
3      X := X + 3
```

We want to prove that the value of X is 10 after the loop, but how to find an appropriate invariant so that we can confirm the result is exactly what we are expecting? It can be noted that $X \equiv 1(mod\ 3)$ is a useful invariant when doing verification. So formally we can formalize the verification process with the proposition below.

$$\frac{\vdash\{X{\equiv}1(mod\ 3){\wedge}X{\leq}10{\wedge}X{\leq}7\}X{:=}X{+}3\{X{\equiv}1(mod\ 3){\wedge}X{\leq}10\}}{\vdash\{X{\equiv}1(mod\ 3){\wedge}X{\leq}10\}WHILE\ X{\leq}7\ DO\ X{:=}X{+}3\{X{\equiv}1(mod\ 3){\wedge}X{\leq}10{\wedge}X{>}7\}}$$

And obviously it is easy to find proposition $X \equiv 1(mod\ 3) \wedge X \leq 10 \wedge X > 7$ is equal to $X = 10$.

3 A Complete Example

```
1  X := A
2  Y := B
3  RES := 0;
4  WHILE NOT (X = Y) DO
5      IF X > Y THEN
6          X := X - 1
7      ELSE
8          Y := Y - 1
9      RES := RES + 1
```

A and B are constanst. Our target is to prove that $RES = |A - B|$. And now we can use Hoare Logic to transform code into formalized propositions.

At the beginning, Line 2 ,Line 3 and Line 4 are three assignments, so we use the Assignment Axiom to assign values A and B to variable identifiers X and Y. And now proposition 1, namely P, is true.

$$P : X = A \wedge Y = B \wedge RES = 0$$

To achieve our final target, we need to find a proper invariant. And here I just meets our initial condition:

$$\frac{\vdash X = A \wedge Y = B \wedge RES = 0}{\vdash RES + |X - Y| = |A - B|}$$

$$I : RES + |X - Y| = |A - B| \text{ (invariant)}$$

Why I is much better than other invariants like $RES \geq 0$? Because the former one is closely related to the final target and the loop next, which reveals useful properties of RES. However, the invariant remains to be checked during the loop.

As we get the initial condition, we begin to transform the loop code. Start from the loop condition L.

$$L : X \neq Y$$

After the loop condition, the body of the loop code has two statements. One is a conditional statement with the condition S and the other is an assignment. First we transform the conditional statement using the Conditional Rule. Here we immediately realize that if I is the pre-condition, the property of I will be temporarily changed after the conditional statement. We name the post-condition I'.

$$S : x > y$$

$$I' = RES + |X - Y| = |A - B| - 1 \text{ (Temporary Change)}$$

And the IF-ELSE code can be then transformed into the proposition below.

$$\frac{\vdash \{I \wedge S\}X := X - 1\{I'\}, \{I \wedge \neg S\}Y := Y - 1\{I'\}}{\vdash \{I\}IF\ S\ THEN\ X := X - 1\ ELSE\ Y := Y - 1\{I'\}}$$

Next is the assignment of RES, which changes I' back into I.

$$\vdash \{I'\}RES := RES + 1\{I\} \text{ (Line 9)}$$

And now, we can say that I is indeed a invariant that never changes after each loop. Since the correctness of I is ensured, we can formalize the whole WHILE-DO code. Here C represents the body of the loop code.

$$\frac{\vdash \{I \wedge L\}C\{I\}}{\vdash \{I\}WHILE\ L\ DO\ C\{I \wedge \neg L\}}$$

And it's easy to prove $I \wedge \neg L$, namely $RES + |X - Y| = |A - B| \wedge X = Y$, is equal to our target $RES = |A - B|$.

4 Conclusion

This article discusses the core concepts of Hoare Logic, and give a complete example of code verification with Hoare Logic. However, the formal material presented only represents a small proportion of Hoare Logic. If you are interested in Hoare Logic, consider going deeper into the relevant papers.

References

[1] C. A. R. Hoare. 1983. An Axiomatic Basic for Computer Programming. *Commun. ACM* 26, 1 (1983), 53–56.