How to use Hoare Logic to verify program correctness

Huajun Lu

Nanjing University
231220006@smail.nju.edu.cn

Dec 23, 2023

Contents

Introduction

Why we need Hoare Logic?

Why we need Hoare Logic?

- We often fail to write programs that meet our expectations, so when a programmer is programming, it is important to verify that the code is correct.

Why we need Hoare Logic?

- We often fail to write programs that meet our expectations, so when a programmer is programming, it is important to verify that the code is correct.
- Thus it is desirable to use a valid logic to verify the program correctness.

What is Hoare Logic?

What is Hoare Logic?

- **Hoare Logic can establish a transformation between code and logic formulas thus ensuring that our programs are validated.**

What is Hoare Logic?

- **Hoare Logic can establish a transformation between code and logic formulas thus ensuring that our programs are validated.**
- All the consequences of excuting programs can be found out "by means of purely deductive reasoning"[1] with Hoare Logic.

What is Hoare Logic?

- **Hoare Logic can establish a transformation between code and logic formulas thus ensuring that our programs are validated.**
- All the consequences of excuting programs can be found out "by means of purely deductive reasoning"[1] with Hoare Logic.
- And Hoare Logic consists of basic axioms and rules of inference, which will be elucidated next.

Axioms and Rules

Hoare triple

Hoare triple

Definition

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

Hoare triple

Definition

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

P: **Pre-condition** → C: **Command(Code)** → Q: **Post-condition**

Hoare triple

Definition

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

P: **Pre-condition** $\longrightarrow$ C: **Command(Code)** $\longrightarrow$ Q: **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.

Example

Definition

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

For example:

Example

Definition

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

For example:

$$\{x = y\}z = x\{y = z\} : true$$

Example

Definition

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

For example:

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

The Assignment Axiom

The Assignment Axiom

Definition

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

The Assignment Axiom

Definition

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

- Assignment is the most characteristic and basic feature of a program.

The Assignment Axiom

Definition

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

- Assignment is the most characteristic and basic feature of a program.
- $\vdash$ is the notation means that the proposition can be syntactically derived.

The Assignment Axiom

Definition

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

- Assignment is the most characteristic and basic feature of a program.
- $\vdash$ is the notation means that the proposition can be syntactically derived.
- $:=$ means assignment.

The Assignment Axiom

Definition

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

- Assignment is the most characteristic and basic feature of a program.
- $\vdash$ is the notation means that the proposition can be syntactically derived.
- $:=$ means assignment.
- Here V is a variable identifier, E is an identified expression, Q is any statement.

The Assignment Axiom

Definition

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

- Assignment is the most characteristic and basic feature of a program.
- $\vdash$ is the notation means that the proposition can be syntactically derived.
- $:=$ means assignment.
- Here V is a variable identifier, E is an identified expression, Q is any statement.
- $Q[E/V]$ means the result of replacing all occurrences of V in Q by E.

Example

Definition

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

Example

Definition

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

Code

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

Example

Definition

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

Code

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

And the code above is equal to the triple below:

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

The Sequencing Rule

The Sequencing Rule

Definition

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

The Sequencing Rule

Definition

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

- The rule permits the deduction of new theoroms from one proved theorem or axiom to new theorems.

The Sequencing Rule

Definition

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

- The rule permits the deduction of new theoroms from one proved theorem or axiom to new theorems.
- Here $\dfrac{P_1}{P_2}$ means that, if the correctness of P_1 is ensured, P_2 can be proved correct.

The Sequencing Rule

Definition

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

- The rule permits the deduction of new theoroms from one proved theorem or axiom to new theorems.
- Here $\frac{P_1}{P_2}$ means that, if the correctness of P_1 is ensured, P_2 can be proved correct.
- After the execution of C_1 and C_2, state P can produce Q, and then Q, as the **mid-condition**, can produce R sequentially.

Example

Definition

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

Example

Definition

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

Code

```
R:=X
Y:=R
```

Example

Definition

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

Code

```
R:=X
Y:=R
```

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

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

The Conditional Rule

The Conditional Rule

Definition

$$\frac{\vdash \{P \land S\} C_1 \{Q\}, \vdash \{P \land \neg S\} C_2 \{Q\}}{\vdash \{P\} IF\ S\ THEN\ C_1\ ELSE\ C_2 \{Q\}}$$

The Conditional Rule

Definition

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

- As programmers, we often write IF-ELSE code.

The Conditional Rule

Definition

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

- As programmers, we often write IF-ELSE code.
- Here $\wedge$ means "and", $\vee$ means "or", $\neg$ means not.

The Conditional Rule

Definition

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

- As programmers, we often write IF-ELSE code.
- 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

Definition

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

Example

Definition

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

Code

```
IF X <= Y THEN
   Z := X
ELSE
   Z := Y
```

Example

Definition

$$\frac{\vdash \{P \land S\} C_1 \{Q\}, \vdash \{P \land \neg S\} C_2 \{Q\}}{\vdash \{P\} IF\ S\ THEN\ C_1\ ELSE\ C_2 \{Q\}}$$

Code

```
IF X <= Y THEN
   Z := X
ELSE
   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:

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

The Iteration Rule

The Iteration Rule

Definition

$$\frac{\vdash \{P \wedge S\} C \{P\}}{\vdash \{P\}\, WHILE\ S\ DO\ C \{P \wedge \neg S\}}$$

The Iteration Rule

Definition

$$\frac{\vdash \{P \wedge S\} C \{P\}}{\vdash \{P\}\, WHILE\ S\ DO\ C \{P \wedge \neg S\}}$$

- We often write all kinds of loop code, and now I am going to introduce the Iteration Rule.

The Iteration Rule

Definition

$$\frac{\vdash \{P \wedge S\}C\{P\}}{\vdash \{P\}\,\mathit{WHILE}\ S\ \mathit{DO}\ C\{P \wedge \neg S\}}$$

- We often write all kinds of loop code, and now I am going to introduce the Iteration Rule.
- P is the **invariant** of the whole While-Command and is always true while this part of code is being excuted.

The Iteration Rule

Definition

$$\frac{\vdash \{P \wedge S\}C\{P\}}{\vdash \{P\}\,\mathit{WHILE}\ S\ \mathit{DO}\ C\{P \wedge \neg S\}}$$

- We often write all kinds of loop code, and now I am going to introduce the Iteration Rule.
- P is the **invariant** of the whole While-Command and is always true while this part of code is being excuted.
- S is the condition to check whether the loop should be terminated or continue.

Example

Definition

$$\frac{\vdash \{P \wedge S\}\, C \{P\}}{\vdash \{P\}\, WHILE\ S\ DO\ C \{P \wedge \neg S\}}$$

Example

Definition

$$\frac{\vdash \{P \wedge S\}\, C\, \{P\}}{\vdash \{P\}\, WHILE\ S\ DO\ C\, \{P \wedge \neg S\}}$$

Code

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

Example

Definition

$$\frac{\vdash \{P \wedge S\}\, C \{P\}}{\vdash \{P\}\, WHILE\ S\ DO\ C \{P \wedge \neg S\}}$$

Code

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

- $X \equiv 1 (mod\ 3)$ is an **appropriate** invariant.

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

Example

Definition

$$\frac{\vdash \{P \wedge S\}\, C \{P\}}{\vdash \{P\}\, WHILE\ S\ DO\ C \{P \wedge \neg S\}}$$

Code

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

- $X \equiv 1 (mod\ 3)$ is an **appropriate** invariant.

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

- Obviously $X \equiv 1 (mod\ 3) \wedge X \leq 10 \wedge X > 7$ is equal to $X = 10$.

A Complete Example

Code

Code

Code

```
X := A
Y := B
RES := 0
[Assertion: $\{True\}C_1\{I\}$]
WHILE NOT (X = Y) DO [$L : (\neg(X = Y))$]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    [Assertion: $\{I\}C_2\{I'\}$]
    RES := RES + 1
    [Assertion: $\{I'\}C_3\{I\}$]
[Assertion: $\{I \wedge S\}WHILE\ S\ DO\ C\{I \wedge \neg S\}$]
```

Code

Code

```
X := A
Y := B
RES := 0
[Assertion: $\{True\}C_1\{I\}$]
WHILE NOT (X = Y) DO [$L : (\neg(X = Y))$]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    [Assertion: $\{I\}C_2\{I'\}$]
    RES := RES + 1
    [Assertion: $\{I'\}C_3\{I\}$]
[Assertion: $\{I \wedge S\}WHILE\ S\ DO\ C\{I \wedge \neg S\}$]
```

Our target is to prove that

$$\{True\}C\{RES = |A - B|\}.$$

Initialization

Initialization

Code

```
X := A
Y := B
RES := 0
```

[Assertion: $\{True\}C_1\{I\}$]

Initialization

Code

```
1   X := A
2   Y := B
3   RES := 0
4   [Assertion: {True}C_1{I}]
```

Line 2 ,Line 3 and Line 4 are three assignments. And now proposition 1, namely P, is true.

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

Invariant

Invariant

Code

```
X := A
Y := B
RES := 0
```

[Assertion: $\{True\} C_1 \{I\}$]

Invariant

Code

```
X := A
Y := B
RES := 0
```

[Assertion: $\{True\}C_1\{I\}$]

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

We need to find a proper invariant.

Invariant

Code

```
X := A
Y := B
RES := 0
[Assertion: {True}C1{I}]
```

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

We need to find a proper invariant.

$$X = A \land Y = B \land RES = 0 \Rightarrow RES + |X - Y| = |A - B|$$

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

Invariant

Code

```
X := A
Y := B
RES := 0
[Assertion: {True}C₁{I}]
```

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

We need to find a proper invariant.

$$X = A \wedge Y = B \wedge RES = 0 \Rightarrow RES + |X - Y| = |A - B|$$
$$I : RES + |X - Y| = |A - B| \text{ (invariant)}$$

- I is closely related to the final target.

Invariant

Code

```
X := A
Y := B
RES := 0
[Assertion: {True}C1{I}]
```

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

We need to find a proper invariant.

$$X = A \wedge Y = B \wedge RES = 0 \Rightarrow RES + |X - Y| = |A - B|$$

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

- I is closely related to the final target.
- I reveals useful properties of RES.

Invariant

Code

```
X := A
Y := B
RES := 0
[Assertion: {True}C_1{I}]
```

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

We need to find a proper invariant.

$$X = A \wedge Y = B \wedge RES = 0 \Rightarrow RES + |X - Y| = |A - B|$$

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

- I is closely related to the final target.
- I reveals useful properties of RES.

However, the invariant remains to be checked during the loop.

Loop

Loop

Code

```
WHILE NOT (X = Y) DO [L : (¬(X = Y))]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    [Assertion: {I}C₂{I′}]
    RES := RES + 1
    [Assertion: {I′}C₃{I}]
```

Loop

Code

```
WHILE NOT (X = Y) DO [$L : (\neg(X = Y))$]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    [Assertion: $\{I\} C_2 \{I'\}$]
    RES := RES + 1
    [Assertion: $\{I'\} C_3 \{I\}$]
```

Start from the loop condition L.

$$L : X \neq Y$$

Loop

Code

```
WHILE NOT (X = Y) DO [L : (¬(X = Y))]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    [Assertion: {I}C2{I'}]
    RES := RES + 1
    [Assertion: {I'}C3{I}]
```

Start from the loop condition L.

$$L : X \neq Y$$

Next is a conditional statement with the condition S and an assignment.

IF-ELSE

IF-ELSE

Code

```
IF X > Y THEN
   X := X - 1
ELSE
   Y := Y - 1
[Assertion: {I}C_2{I'}]
```

IF-ELSE

Code

```
IF X > Y THEN
   X := X - 1
ELSE
   Y := Y - 1
[Assertion: {I}C_2{I'}]
```

The property of I will be temporarily changed after the conditional statement.

IF-ELSE

Code

```
IF X > Y THEN
   X := X - 1
ELSE
   Y := Y - 1
[Assertion: {I}C2{I'}]
```

The property of I will be temporarily changed after the conditional statement. We name the post-condition I'.

$$S : x > y$$

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

IF-ELSE

Code

```
IF X > Y THEN
    X := X - 1
ELSE
    Y := Y - 1
[Assertion: {I}C2{I'}]
```

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.

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

Assignment of RES

Assignment of RES

Code

```
RES := RES + 1
[Assertion: {I'}C_3{I}]
```

Assignment of RES

Code

```
RES := RES + 1
[Assertion: {I'}C3{I}]
```

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

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

Assignment of RES

Code

```
        RES := RES + 1
        [Assertion: {I'}C_3{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.

Final Step

Final Step

Code

```
X := A, Y := B, RES := 0
[Assertion: $\{True\} C_1 \{I\}$]
WHILE NOT (X = Y) DO [$L : (\neg(X = Y))$]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    RES := RES + 1
[Assertion: $\{I \wedge S\}$ $WHILE\ S\ DO\ C\{I \wedge \neg S\}$]
```

Final Step

Code

```
X := A, Y := B, RES := 0
[Assertion: {True}C_1{I}]
WHILE NOT (X = Y) DO [L : (¬(X = Y))]
    IF X > Y THEN
        X := X - 1
    ELSE
        Y := Y - 1
    RES := RES + 1
[Assertion: {I ∧ S} WHILE S DO C{I ∧ ¬S}]
```

$$\cfrac{\cfrac{\vdash \{I \wedge L\}C\{I\}}{\vdash \{I\}\,\mathit{WHILE}\ L\ \mathit{DO}\ C\{I \wedge \neg L\}}}{\vdash \{RES + |X - Y| = |A - B| \wedge X = Y\}\mathit{Empty}\{RES = |A - B|\}}$$

Conclusion

Conclusion

Conclusion

- This article discusses the core concepts of Hoare Logic, and give a complete example of code verification with Hoare Logic.

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.

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.

Reference

Reference

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