Hoare Logic and Program Verification

Qi'ao Chen
21210160025@m.fudan.edu.cn

March 29, 2022

Outline

Formal Methods

- Formal Specification: using mathematical notation to give a precise description of what a program should do
- Formal Verification: using precise rules to mathematically prove that a program satisfies a formal specification
- Formal Development (Refinement): developing programs in a way that ensures mathematically they meet their formal specifications

Why

For some applications, correctness is especially important

- nuclear reactor controllers
- car braking systems
- fly-by-wire aircraft
- software controlled medical equipment
- voting machines
- cryptographic code

Bird's Eye View

- Also known as Floyd Hoare Logic is a formal system for reasoning rigorously about the correctness of *imperative* programs
- First proposed by C. A. R. Hoare (Turing Award, 1980)
- Original Idea seeded by Robert Floyd (Turing Award, 1978)

Formally

- A Proof System for reasoning about partial corretness of certain kinds of programs
  - set of axioms
  - rules of inference
  - underlying logic
- Motivation: Assertion checking in (sequential) programs

What does a program like

Backus–Naur form

- Backus–Naur form or Backus normal form (BNF) is a metasyntax notation for Chomsky's context-free grammars, often used to describe the syntax of languages used in computing
- Context-free grammar has the same computability as pushdown automata (a proof)

Example

$\langle S \rangle$ ::= '-' $\langle FN \rangle$ | $\langle FN \rangle$

$\langle FN \rangle$ ::= $\langle DL \rangle$ | $\langle DL \rangle$ '.' $\langle DL \rangle$

$\langle DL \rangle$ ::= $\langle D \rangle | \langle D \rangle$ $\langle DL \rangle$

$\langle D \rangle$ ::= '0' | '1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'

Here `S` is the start symbol, `FN` products a fractional number, `DL` is a digit list, while `D` is a digit Then for `S`, we have

$$\texttt{S} \Rightarrow \texttt{FN} \Rightarrow \texttt{DL . DL} \Rightarrow \texttt{D . DL} \Rightarrow \texttt{3 . DL} \Rightarrow \texttt{3 . D DL} \Rightarrow \texttt{3 . D D} \Rightarrow \texttt{3 . 1 D} \Rightarrow \texttt{3 . 1 4}$$

A simple imperative language

- Expressions

$$E \ ::= \ n \mid x \mid -E \mid E + E \mid \ldots$$

- Boolean Conditions

$$B \ ::= \ \texttt{true} \mid E = E \mid E >= E \mid \neg B \mid B \wedge B$$

- Program Statements

$$P \ ::= \ x := E \mid P; P \mid \ \texttt{if} \ B \ \texttt{then} \ P \ \texttt{else} \ P \mid \ \texttt{while} \ B \ P$$

A simple assertion language

Assertion : A logical formula describing a set of valuations on program variables with some *interesting* property.
Expressed in the underlying logic (FO here)

- Expressions

$$E \ ::= \ n \mid x \mid -E \mid E + E \mid \ldots$$

  Here the set of variables is not restricted to the set of program variables

- Basic Propositions

$$E \ ::= \ E = E \mid E >= E$$

- Assertions

$$A \ ::= \ \texttt{true} \mid B \mid \neg A \mid A \wedge A \mid \forall v \ A$$

Assertion Semantics

- As program executes, the valuation of variables (read state) changes
- An execution of a program statement, transforms one state to another state
- At some point during execution, let the state be s
- Program satisfies assertion A at this point iff $s \vDash A$

$$
\begin{array}{rcl}
s \vDash B & \text{iff} & [\![B]\!]_s = \texttt{true} \\
s \vDash \neg A & \text{iff} & s \nvDash A \\
s \vDash A_1 \wedge A_2 & \text{iff} & s \vDash A_1 \text{ and } s \vDash A_2 \\
s \vDash \forall v.A & \text{iff} & \forall x \in \mathbb{Z}.s[x \mapsto v] \vDash A
\end{array}
$$

Here, the free variables in assertions are assumed to be included in the set of program variables

Example program

Consider the following program written in our imperative language, annotated with assertions from our assertions language:

```
_(ensures n>= 0)
k := 0;
j := 1;
while (k != n) {
k := k+1;
j := 2*j;
}
_(assert j = 2^n)
```

We wish to check if starting from a positive value for n, is the value of j equal to 2^n after having executed all the statements?

Hoare Triple: Syntax

A Hoare triple $\{\phi_1\}P\{\phi_2\}$ is a formula:

- ϕ_1 and ϕ_2 are formulae in a base logic (FO logic for us)
- P is a program in our imperative language
- ϕ_1: Precondition, ϕ_2: Postcondition

Examples of syntactically correct Hoare triples

- $\{(n \geq 0) \wedge (n^2 > 28)\}\ m := n + 1; m := m * m\ \{\neg(m = 36)\}$
- $\{\exists x, y.(y > 0) \wedge (n = x^y)\}\ n := n * (n + 1)\ \{\exists x, y.(n = x^y)\}$

Hoare Triple: Semantics

- The partial correctness specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state s satisfying ϕ_1
  - Whenever an execution of P terminates in state s', then $s' \models \phi_2$
- The total corretness specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state s satisfying ϕ_1
  - Every execution of P terminates, and
  - Whenever an execution of P terminates in state s', then $s' \models \phi_2$

Partial/Total Correctness

For programs without loops, both semantics coincide

Assignment Rule

Program Construct

$$E ::= x \mid n \mid E + E \mid E \mid \dots$$
$$P ::= x := E$$

Inference Rule

$$\frac{}{\{\phi([x \mapsto E])\} x := E \{\phi(x)\}}$$

where $\phi([x \mapsto E])$ replaces every free occurrence of x in ϕ by E

Example:

$$\{(z \cdot y > 5) \wedge (\exists x. y = x^x)\} x := z * y \{(x > 5) \wedge (\exists x. y = x^x)\}$$

Rule for Sequential Composition

Program Construct

$P ::= P; P$

Inference Rule

$$\frac{\{\phi\}P_1\{\eta\} \qquad \{\eta\}P_2\{\psi\}}{\{\phi\}P_1; P_2\{\psi\}}$$

Example:

$$\frac{\{y + z > 4\}y := y + z\{y > 4\} \qquad \{y > 4\}x := y + 2\{x > 6\}}{\{y + z > 4\}y := y + z; x := y + 2\{x > 6\}}$$

Rule of Consequence

Inference Rule

$$\frac{\phi \Rightarrow \phi_1 \qquad \{\phi_1\}P\{\psi_1\} \qquad \psi_1 \Rightarrow \psi}{\{\phi\}P\{\psi\}}$$

$\phi \Rightarrow \phi_1$ and $\psi_1 \Rightarrow \psi$ are implications in underlying (FO) logic

Rules for Conditional Branch

Program Construct

$$E ::= n \mid x \mid -E \mid E + E \mid ...$$
$$B ::= \texttt{true} \mid E = E \mid E >= E \mid \neg B \mid B \wedge B$$
$$P ::= \texttt{if}\ \ P\ \ \texttt{then}\ \ P\ \ \texttt{else}\ \ P$$

Inference Rule

$$\frac{\{\phi \wedge B\}P_1\{\psi\} \qquad \{\phi \wedge \neg B\}P_2\{\psi\}}{\{\phi\}\texttt{if}\ \ B\ \ \texttt{then}\ \ P_1\ \ \texttt{else}\ \ P_2\{\ \}}$$

Example:

$$\frac{\{(y>4) \wedge (z>1)\}y := y+z\{y>3\} \qquad \{(y>4) \wedge \neg(z>1)\}y := y-1\{y>3\}}{\{y>4\}\ \texttt{if}\ \ (z>1)\ \ \texttt{then}\ \ y := y+z\ \ \texttt{else}\ \ y := y-1\{y>3\}}$$

Partial Corretness of Loops

Program Construct

$$E ::= n \mid x \mid -E \mid E + E \mid \dots$$
$$B ::= \texttt{true} \mid E = E \mid E >= E \mid \neg B \mid B \wedge B$$
$$P ::= \texttt{while}\ \ B\ P$$

Inference Rule

$$\frac{\{\phi \wedge B\} P \{\phi\}}{\{\phi\}\ \ \texttt{while}\ \ B\ P \{\phi \wedge \neg B\}}$$

- ϕ is loop invariant
- Partial Corretness Semantics:
  - If loop does not terminate, Hoare triples is vacuously satisfied
  - If it terminates, $\phi \wedge \neg B$ must be satisfied after termination

Partial Correctness of Loops

Inference Rule

$$\frac{\{\phi \wedge B\}P\{\phi\}}{\{\phi\}\ \texttt{while}\ B\ P\{\phi \wedge \neg B\}}$$

Example:

$$\frac{\{(y = x + z) \wedge (z \neq 0)\}x := x + 1; z := z - 1\{y = x + z\}}{\{y = x + z\}}$$

Summary of Axioms

- Assignment

$$\frac{}{\{\phi([x \mapsto E])\}x := E\{\phi(x)\}}$$

- Sequential Composition

$$\frac{\{\phi\}P_1\{\eta\} \qquad \{\eta\}P_2\{\psi\}}{\{\phi\}P_1; P_2\{\psi\}}$$

- Conditional Statement

$$\frac{\{\phi \wedge B\}P_1\{\psi\} \qquad \{\phi \wedge \neg B\}P_2\{\psi\}}{\{\phi\}\texttt{if}\ B\ \texttt{then}\ P_1\ \texttt{else}\ P_2\texttt{\{ \}}}$$

- Iteration

$$\frac{\{\phi \wedge B\}P\{\phi\}}{\{\phi\}\ \texttt{while}\ B\ P\{\phi \wedge \neg B\}}$$

- Weakening pre-condition, Strengthening post-condition

$$\frac{\phi \Rightarrow \phi_1 \qquad \{\phi_1\}P\{\psi_1\} \qquad \psi_1 \Rightarrow \psi}{\{\phi\}P\{\psi\}}$$

Structural Rules

- Conjunction

$$\frac{\{\phi_1\}P\{\psi_1\} \quad \{\phi_2\}P\{\psi_2\}}{\{\phi_1 \wedge \phi_2\}P\{\psi_1 \wedge \psi_2\}}$$

- Disjunction

$$\frac{\{\phi_1\}P\{\psi_1\} \quad \{\phi_2\}P\{\psi_2\}}{\{\phi_1 \vee \psi_2\}P\{\psi_1 \vee \psi_2\}}$$

- Existential Quantification(v is not free in P)

$$\frac{\{\phi\}P\{\psi\}}{\{\exists v.\phi\}P\{\exists v.\psi\}}$$

- Universal Quantification(v is not free in P)

$$\frac{\{\phi\}P\{\psi\}}{\{\forall v.\phi\}P\{\forall v.\psi\}}$$

A Hoare logic proof

Let P be

```
k := 0
j := 1
while (k != n) {
  k := k + 1;
  j := 2 + j;
}
```

Our goal is to prove the validity of $\{n > 0\}P\{j = 1 + 2 * n\}$

A Hoare logic proof

Sequential composition rule will give us a proof if we can fill in the template

$$\{n > 0\}$$

```
k := 0
```

$$\{\varphi_1\}$$

```
j := 1
```

$$\{\varphi_2\}$$

```
while (k != n) {k := k+1; j := 2+j;}
```

$$\{j = 1 + 2 * n\}$$

A Hoare logic proof

To prove

$$\{\varphi_2\}\texttt{while(k != n)\{k := k+1;j := 2+j;\}}\{j = 1 + 2 * n\}$$

using loop invariant $j = 1 + 2 * k$
We only need to show that

- $\varphi_2 \Rightarrow (j = 1 + 2 * k)$
- $\{(j = 1 + 2 * k) \wedge (k \neq n)\}\texttt{k:=k+1;j:=2+j}\{j = 1 + 2 * k\}$
- $((j = 1 + 2 * k) \wedge \neg(k \neq n)) \Rightarrow (j = 1 + 2 * n)$

A Hoare logic proof

- $\varphi_2 \Rightarrow (j = 1 + 2 * k)$ holds if φ_2 is $j = 1 + 2 * k$
- $(j = 1 + 2 * k) \wedge \neg(k \neq n) \Rightarrow (j = 1 + 2 * n)$ holds in integer arithmetic

A Hoare logic proof

To show

$$\{(j = 1 + 2 * k) \wedge (k \neq n)\}\texttt{k:=k+1;j:=2+j}\{j = 1 + 2 * k\}$$

Applying assignment rule twice

$$\{2 + j = 1 + 2 * k\}\texttt{j:=2+j}\{j = 1 + 2 * k\}$$
$$\{2 + j = 1 + 2 * (k + 1)\}\texttt{k:=k+1}\{2 + j = 1 + 2 * k\}$$

Simplifying and applying sequential compositon rule we we get

$$\{j = 1 + 2 * k\}\texttt{k:=k+1;j:=2+j}\{j = 1 + 2 * k\}$$

Then apply rule for strengthening precedent

$$\frac{\begin{array}{c}(j = 1 + 2 * k) \wedge (k \neq n) \Rightarrow (j = 1 + 2 * k) \\ \{j = 1 + 2 * k\}\texttt{k:=k+1;j:=2+j}\{j = 1 + 2 * k\}\end{array}}{\{(j = 1 + 2 * k) \wedge (k \neq n)\}\texttt{k:=k+1;j:=2+j}\{j = 1 + 2 * k\}}$$

A Hoare logic proof

we have thus show that

$$\{n > 0\}$$

```
k := 0
```

$$\{\varphi_1\}$$

```
j := 1
```

$$\{\varphi_2 : j = 1 + 2 * k\}$$

```
while (k != n) {k := k+1; j := 2+j;}
```

$$\{j = 1 + 2 * n\}$$

A Hoare logic proof

Similarly, we choose φ_1 as $k = 0$, hence we have

$$\{n > 0\}$$

```
k := 0
```

$$\{\varphi_1 : k = 0\}$$

```
j := 1
```

$$\{\varphi_2 : j = 1 + 2 * k\}$$

```
while (k != n) {k := k+1; j := 2+j;}
```

$$\{j = 1 + 2 * n\}$$

Soundness

We use $\vdash \{p\}c\{q\}$ to represent that there is a derivation of $\{p\}c\{q\}$ following the rules

Hoare Logic has a sound proof system

Relative Completeness of Hoare Logic

Hoare logic is incomplete: $\vDash \{\texttt{true}\}P\{\texttt{false}\}$ iff P does not halt. But the halting problem is undecidable

Theorem (Cook, 1974)

If there is a complete proof system for proving assertions in the underlying logic, then all valid Hoare triples have a proof