COMP2111 Week 5
Term 1, 2019
Hoare Logic

Sir Tony Hoare

- Pioneer of formal verification
- Invented quicksort
- Invented the null reference
- Invented CSP (formal specification language)
- Invented Hoare Logic

Summary

- $\mathcal{L}$: A simple imperative programming language
- Hoare triples (SYNTAX)
- Hoare logic (PROOF)
- Semantics for Hoare logic

Summary

- $\mathcal{L}$: A simple imperative programming language
- Hoare triples (SYNTAX)
- Hoare logic (PROOF)
- Semantics for Hoare logic

$\mathcal{L}$: A simple imperative programming language

Consider the vocabulary of basic arithmetic:

- Constant symbols: $0, 1, 2, \ldots$
- Function symbols: $+, *, \ldots$
- Predicate symbols: $<, \leq, \geq, |, \ldots$

- An **(arithmetic) expression** is a term over this vocabulary.
- A **boolean expression** is a predicate formula over this vocabulary.

$\mathcal{L}$: A simple imperative programming language

Consider the vocabulary of basic arithmetic:

- Constant symbols: $0, 1, 2, \ldots$
- Function symbols: $+, *, \ldots$
- Predicate symbols: $<, \leq, \geq, |, \ldots$

- An **(arithmetic) expression** is a term over this vocabulary.
- A **boolean expression** is a predicate formula over this vocabulary.

$\mathcal{L}$: A simple imperative programming language

Consider the vocabulary of basic arithmetic:

- Constant symbols: $0, 1, 2, \ldots$
- Function symbols: $+, *, \ldots$
- Predicate symbols: $<, \leq, \geq, |, \ldots$

- An **(arithmetic) expression** is a term over this vocabulary.
- A **boolean expression** is a predicate formula over this vocabulary.

The language $\mathcal{L}$

The language $\mathcal{L}$ is a simple imperative programming language made up of four statements:

Assignment: $x := e$

where x is a variable and e is an arithmetic expression.

Sequencing: $P;Q$

Conditional: **if** b **then** P **else** Q **fi**

where b is a boolean expression.

While: **while** b **do** P **od**

The language $\mathcal{L}$

The language $\mathcal{L}$ is a simple imperative programming language made up of four statements:

Assignment: $x{:=}e$
where x is a variable and e is an arithmetic expression.

Sequencing: $P;Q$

Conditional: **if** b **then** P **else** Q **fi**
where b is a boolean expression.

While: **while** b **do** P **od**

The language $\mathcal{L}$

The language $\mathcal{L}$ is a simple imperative programming language made up of four statements:

Assignment: $x{:=}e$
where x is a variable and e is an arithmetic expression.

Sequencing: $P;Q$

Conditional: **if** b **then** P **else** Q **fi**
where b is a boolean expression.

While: **while** b **do** P **od**

The language $\mathcal{L}$

The language $\mathcal{L}$ is a simple imperative programming language made up of four statements:

Assignment: $x{:=}e$
where x is a variable and e is an arithmetic expression.

Sequencing: $P;Q$

Conditional: **if** b **then** P **else** Q **fi**
where b is a boolean expression.

While: **while** b **do** P **od**

Factorial in $\mathcal{L}$

Example

$$
\begin{array}{l}
f := 1; \\
k := 0; \\
\textbf{while } k < n \textbf{ do} \\
\quad k := k + 1; \\
\quad f := f * k \\
\textbf{od}
\end{array}
$$

Summary

- $\mathcal{L}$: A simple imperative programming language
- Hoare triples (SYNTAX)
- Hoare logic (PROOF)
- Semantics for Hoare logic

Summary

- $\mathcal{L}$: A simple imperative programming language
- Hoare triples (SYNTAX)
- Hoare logic (PROOF)
- Semantics for Hoare logic

Hoare triple (Syntax)

$$\{\varphi\}\, P\, \{\psi\}$$

Intuition:

- φ**:** The **precondition** – an assertion about the state prior to the execution of the code fragment.
- P**:** The **code fragment**
- ψ**:** The **postcondition** – an assertion about the state after to the execution of the code fragment *if it terminates*.

Hoare triple (Syntax)

$$\{\varphi\}\, P\, \{\psi\}$$

Intuition:

- φ**:** The **precondition** – an assertion about the state prior to the execution of the code fragment.
- P**:** The **code fragment**
- ψ**:** The **postcondition** – an assertion about the state after to the execution of the code fragment *if it terminates*.

Hoare triple (Syntax)

$$\{\varphi\}\, P\, \{\psi\}$$

Intuition:

φ**:** The **precondition** – an assertion about the state prior to the execution of the code fragment.

P**:** The **code fragment**

ψ**:** The **postcondition** – an assertion about the state after to the execution of the code fragment *if it terminates*.

Hoare triple: Examples

Example

$$\{(x = 0)\}\, x := 1\, \{(x = 1)\}$$

$$\{(x = 0)\}\, x := 1\, \{(x = 500)\}$$

$$\{(x > 0)\}\, y := 0 - x\, \{(y < 0) \land (x \neq y)\}$$

Hoare triple: Examples

Example

$$\{(x = 0)\}\, x := 1\, \{(x = 1)\}$$

$$\{(x = 0)\}\, x := 1\, \{(x = 500)\}$$

$$\{(x > 0)\}\, y := 0 - x\, \{(y < 0) \wedge (x \neq y)\}$$

Hoare triple: Examples

Example

$$\{(x = 0)\}\, x := 1\, \{(x = 1)\}$$

$$\{(x = 0)\}\, x := 1\, \{(x = 500)\}$$

$$\{(x > 0)\}\, y := 0 - x\, \{(y < 0) \wedge (x \neq y)\}$$

Hoare triple: Examples

Example

$$
\begin{array}{l}
\{n \geq 0\} \\
f := 1; \\
k := 0; \\
\textbf{while } k < n \textbf{ do} \\
\quad k := k + 1; \\
\quad f := f * k \\
\textbf{od} \\
\{f = n!\}
\end{array}
$$

Summary

- $\mathcal{L}$: A simple imperative programming language
- Hoare triples (SYNTAX)
- Hoare logic (PROOF)
- Semantics for Hoare logic

Motivation

Question

We know what we want informally; how do we establish when a triple is valid?

- Develop a semantics (see next lecture), OR
- Derive the triple in a syntactic manner (i.e. proof)

Hoare logic consists of one axiom and four inference rules for deriving Hoare triples.

Motivation

Question

We know what we want informally; how do we establish when a triple is valid?

- Develop a semantics (see next lecture), OR
- Derive the triple in a syntactic manner (i.e. proof)

Hoare logic consists of one axiom and four inference rules for deriving Hoare triples.

Motivation

Question

We know what we want informally; how do we establish when a triple is valid?

- Develop a semantics (see next lecture), OR
- Derive the triple in a syntactic manner (i.e. proof)

Hoare logic consists of one axiom and four inference rules for deriving Hoare triples.

Assignment

$$\frac{}{\{\varphi[e/x]\}\; x := e\; \{\varphi\}} \quad \text{(ass)}$$

Intuition:
If x has property Q *after* executing the assignment; then e must have property Q *before* executing the assignment

Assignment: Example

Example

$$\{(y = 0)\}\, x := y\, \{(x = 0)\}$$

$$\{\quad\quad\}\, x := y\, \{(x = y)\}$$

Assignment: Example

Example

$$\{(y = 0)\}\; x := y\; \{(x = 0)\}$$

$$\{(y = y)\}\; x := y\; \{(x = y)\}$$

$$\{\qquad\}\; x := 1\; \{(x < 2)\}$$

$$\{(y = 3)\}\; x := y\; \{(x > 2)\}$$

Assignment: Example

Example

$$\{(y = 0)\}\; x := y\; \{(x = 0)\}$$

$$\{(y = y)\}\; x := y\; \{(x = y)\}$$

$$\{\qquad\qquad\}\; x := 1\; \{(x < 2)\}$$

$$\{(y = 3)\}\; x := y\; \{(x > 2)\}$$

Assignment: Example

Example

$$\{(y = 0)\}\; x := y\; \{(x = 0)\}$$

$$\{(y = y)\}\; x := y\; \{(x = y)\}$$

$$\{(1 < 2)\}\; x := 1\; \{(x < 2)\}$$

$$\{(y = 3)\}\; x := y\; \{(x > 2)\}$$

Assignment: Example

Example

$$\{(y = 0)\}\, x := y\, \{(x = 0)\}$$

$$\{(y = y)\}\, x := y\, \{(x = y)\}$$

$$\{(1 < 2)\}\, x := 1\, \{(x < 2)\}$$

$$\{(y = 3)\}\, x := y\, \{(x > 2)\}$$

Problem!

Assignment: Example

Example

$$\{(y = 0)\}\, x := y\, \{(x = 0)\}$$

$$\{(y = y)\}\, x := y\, \{(x = y)\}$$

$$\{(1 < 2)\}\, x := 1\, \{(x < 2)\}$$

$$\{(y = 3)\}\, x := y\, \{(x > 2)\} \qquad \textit{Problem}!$$

Sequence

$$\frac{\{\varphi\}\ P\ \{\psi\} \qquad \{\psi\}\ Q\ \{\rho\}}{\{\varphi\}\ P; Q\ \{\rho\}} \quad \text{(seq)}$$

Intuition:
If the postcondition of P matches the precondition of Q we can sequentially combine the two program fragments

Sequence: Example

Example

$$\frac{\{(0=0)\}\; x := 0 \;\{(x=0)\} \qquad \{(x=0)\}\; y := 0 \;\{(x=y)\}}{\{(0=0)\}\; x := 0; y := 0 \;\{(x=y)\}} \quad (\text{seq})$$

Sequence: Example

Example

$$\frac{\{(0=0)\}\, x := 0\, \{(x=0)\} \qquad \{(x=0)\}\, y := 0\, \{(x=y)\}}{\{(0=0)\}\, x := 0; y := 0\, \{(x=y)\}} \quad (\text{seq})$$

Sequence: Example

Example

$$\frac{\{(0 = 0)\}\, x := 0\, \{(x = 0)\} \qquad \{(x = 0)\}\, y := 0\, \{(x = y)\}}{\{(0 = 0)\}\, x := 0; y := 0\, \{(x = y)\}} \quad \text{(seq)}$$

Conditional

$$\frac{\{\varphi \wedge g\}\, P\, \{\psi\} \qquad \{\varphi \wedge \neg g\}\, Q\, \{\psi\}}{\{\varphi\}\, \textbf{if}\ g\ \textbf{then}\ P\ \textbf{else}\ Q\ \textbf{fi}\, \{\psi\}} \quad \text{(if)}$$

Intuition:

- When a conditional is executed, either P or Q will be executed.
- If ψ is a postcondition of the conditional, then it must be a postcondition of *both* branches
- Likewise, f φ is a precondition of the conditional, then it must be a precondition of both branches
- Which branch gets executed depends on g, so we can assume g to be a precondition of P and $\neg g$ to be a precondition of Q (strengthen the preconditions).

While

$$\frac{\{\varphi \wedge g\}\ P\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ g\ \textbf{do}\ P\ \textbf{od}\ \{\varphi \wedge \neg g\}} \quad \text{(loop)}$$

Intuition:

- φ is a **loop-invariant**. It must be both a pre- and postcondition of P so that sequences of Ps can be run together.
- If the while loop terminates, g cannot hold.

Precondition strengthening and Postcondition weakening

$$\frac{\varphi' \to \varphi \qquad \{\varphi\}\, P\, \{\psi\} \qquad \psi \to \psi'}{\{\varphi'\}\, P\, \{\psi'\}} \quad \text{(cons)}$$

Intuition:

- Adding assertions to the precondition makes it more likely the postcondition will be reached
- Removing assertions to the postcondition makes it more likely the postcondition will be reached
- If you can reach the postcondition initially, then you can reach it in the more likely scenario