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

- $L$: A simple imperative programming language
- Hoare triples (SYNTAX)
- Hoare logic (PROOF)
- Semantics for Hoare logic
Consider the vocabulary of basic arithmetic:

- **Constant symbols:** 0, 1, 2, …
- **Function symbols:** +, ∗, …
- **Predicate symbols:** <, ≤, ≥, |, …

- An (arithmetic) expression is a term over this vocabulary.
- A boolean expression is a predicate formula over this vocabulary.
Consider the vocabulary of basic arithmetic:

- Constant symbols: 0, 1, 2, …
- Function symbols: +, *, …
- Predicate symbols: <, ≤, ≥, |, …

An **(arithmetic) expression** is a term over this vocabulary.

A **boolean expression** is a predicate formula over this vocabulary.
\(L\): A simple imperative programming language

Consider the vocabulary of basic arithmetic:
- Constant symbols: 0, 1, 2, …
- Function symbols: +, ∗, …
- Predicate symbols: <, ≤, ≥, |, …

An (arithmetic) expression is a term over this vocabulary.
A boolean expression is a predicate formula over this vocabulary.
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 $\mathcal{L}$ 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 $\mathcal{L}$ 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 $\mathcal{L}$ 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

\[
\begin{align*}
    f &:= 1; \\
    k &:= 0; \\
    \textbf{while} & k < n \textbf{ do} \\
        & k := k + 1; \\
        & f := f \times k \\
    \textbf{od}
\end{align*}
\]
**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:

\( \varphi \): The **precondition** – an assertion about the state prior to the execution of the code fragment.

\( P \): The **code fragment**

\( \psi \): 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:

\(\varphi\): The **precondition** – an assertion about the state prior to the execution of the code fragment.

\(P\): The **code fragment**

\(\psi\): The **postcondition** – an assertion about the state after to the execution of the code fragment *if it terminates*. 
Hoare triple (Syntax)

\{ \varphi \} \text{ } P \{ \psi \}

Intuition:

\varphi:  The \textbf{precondition} – an assertion about the state prior to the execution of the code fragment.

\textit{P}:  The \textbf{code fragment}

\psi:  The \textbf{postcondition} – an assertion about the state after to the execution of the code fragment \textit{if it terminates}. 
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

\[
\{ (x = 0) \} x := 1 \{ (x = 1) \}
\]

\[
\{ (x = 0) \} x := 1 \{ (x = 500) \}
\]

\[
\{ (x > 0) \} y := 0 - x \{ (y < 0) \land (x \neq y) \}
\]
Hoare triple: Examples

Example

\[
\begin{align*}
\{(x = 0)\} & \ x := 1 \ \{(x = 1)\} \\
\{(x = 0)\} & \ x := 1 \ \{(x = 500)\} \\
\{(x > 0)\} & \ y := 0 - x \ \{(y < 0) \land (x \neq y)\}
\end{align*}
\]
Hoare triple: Examples

Example

\{ n \geq 0 \} \\
\text{\hspace{1em} } f := 1; \\
\text{\hspace{1em} } k := 0; \\
\textbf{while} \ k < n \ \textbf{do} \\
\hspace{2em} k := k + 1; \\
\hspace{2em} f := f \times k \\
\textbf{od} \\
\{ f = n! \}
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

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

Intuition:
If \( x \) has property \( Q \) after executing the assignment; then \( e \) must have property \( Q \) before executing the assignment
Example

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

\{(x = y)\} \ x := y \{(x = y)\}
Example

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

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

\{(y = 1)\} 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)\}
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)\}  \hspace{1cm} Problem!
Sequence

\[
\{\varphi\} \quad P \quad \{\psi\} \quad \{\psi\} \quad Q \quad \{\rho\}
\]

(\text{seq})

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

\[
\begin{align*}
\{(0 = 0)\} & \ x := 0 \ \{(x = 0)\} & \{(x = 0)\} & \ y := 0 \ \{(x = y)\} \\
\{(0 = 0)\} & \ x := 0; \ y := 0 \ \{(x = y)\} & \{\text{seq}\}
\end{align*}
\]
Example:

\[
\begin{align*}
\{(0 = 0)\} & \ x := 0 \{(x = 0)\} \\
\{(x = 0)\} & \ y := 0 \{(x = y)\} \\
\{(0 = 0)\} & \ x := 0; y := 0 \{(x = y)\} \\
\end{align*}
\]
Sequence: Example

Example

\[
\begin{align*}
\{ (0 = 0) \} & \ x := 0 \ { (x = 0) } \\
\{ (x = 0) \} & \ y := 0 \ { (x = y) } \\
\{ (0 = 0) \} & \ x := 0 ; \ y := 0 \ { (x = y) } \\
\end{align*}
\]
Conditional

\[
\{\varphi \land g\} P \{\psi\} \quad \{\varphi \land \neg g\} Q \{\psi\}
\]

\[
\{\varphi\} \text{ if } g \text{ then } P \text{ else } Q \text{ fi } \{\psi\}
\]

(if)

Intuition:

- When a conditional is executed, either \( P \) or \( Q \) will be executed.
- If \( \psi \) is a postcondition of the conditional, then it must be a postcondition of both branches.
- Likewise, if \( \varphi \) 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

\[
\begin{align*}
\{\varphi \land g\} & \ P \ \{\varphi\} \\
\{\varphi\} & \text{while } g \ \text{do } P \ \text{od } \{\varphi \land \neg g\}
\end{align*}
\] (loop)

Intuition:
- $\varphi$ is a **loop-invariant**. It must be both a pre- and postcondition of $P$ so that sequences of $P$s can be run together.
- If the while loop terminates, $g$ cannot hold.
Precondition strengthening and Postcondition weakening

\[
\varphi' \rightarrow \varphi \quad \{\varphi\} P \{\psi\} \quad \psi \rightarrow \psi' \quad (\text{cons})
\]

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