Introduction to Hoare Logic

Umang Mathur

Department of Computer Science
University of Illinois, Urbana Champaign

October 22, 2015
Outline

1. Introduction
   - Bird’s Eye View
   - Formal Introduction

2. Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3. Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4. Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
Outline

1. Introduction
   - Bird’s Eye View
   - Formal Introduction

2. Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3. Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4. Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
Also known as **Floyd Hoare Logic** is a formal system for reasoning rigorously about the correctness of computer programs.
Bird’s Eye View

- Also known as **Floyd Hoare Logic** is a formal system for reasoning rigorously about the correctness of computer programs

- First proposed by C. A. R. Hoare (Turnig Award, 1980)
Bird’s Eye View

- Also known as **Floyd Hoare Logic** is a formal system for reasoning rigorously about the correctness of computer programs.

- First proposed by C. A. R. Hoare (Turnig Award, 1980)

- Original Idea seeded by Robert Floyd (Turing Award, 1978)
Outline

1. Introduction
   - Bird’s Eye View
   - Formal Introduction

2. Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3. Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4. Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
Formally
Formally

- Proof System for reasoning about *partial correctness* of certain kinds of programs
Formally

- Proof System for reasoning about *partial correctness* of certain kinds of programs
  - Set of axioms
Formally

- Proof System for reasoning about *partial correctness* of certain kinds of programs
  - Set of axioms
  - Rules of Inference
Formally

- Proof System for reasoning about *partial correctness* of certain kinds of programs
  - Set of axioms
  - Rules of Inference
  - Underlying logic
Formally

- Proof System for reasoning about *partial correctness* of certain kinds of programs
  - Set of axioms
  - Rules of Inference
  - Underlying logic

- **Motivation**: Assertion checking in (sequential) programs
Formally

- Proof System for reasoning about \textit{partial correctness} of certain kinds of programs
  - Set of axioms
  - Rules of Inference
  - Underlying logic

- \textbf{Motivation} : Assertion checking in (sequential) programs (can do much more !)
Outline

1. Introduction
   - Bird’s Eye View
   - Formal Introduction

2. Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3. Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4. Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
A simple Imperative Language
A simple Imperative Language

- Expressions:

\[ E ::= n \mid x \mid -E \mid E + E \mid \ldots \]
A simple Imperative Language

- **Expressions**: 
  \[ E ::= \ n \ | \ x \ | \ -E \ | \ E + E \ | \ldots \]

- **Boolean Conditions**: 
  \[ B ::= \ \text{true} \ | \ E = E \ | \ E \geq E \ | \neg B \ | \ B \land B \]
A simple Imperative Language

- Expressions:
  \[ E ::= n \mid x \mid -E \mid E + E \mid \ldots \]

- Boolean Conditions:
  \[ B ::= \text{true} \mid E = E \mid E \geq E \mid \neg B \mid B \land B \]

- Program Statements:
  \[ P ::= x := E \mid P;P \mid \text{if } B \text{ then } P \text{ else } P \mid \text{while } B \ P \]
Outline

1. Introduction
   - Bird’s Eye View
   - Formal Introduction

2. Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3. Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4. Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
A simple Assertion Language
A simple Assertion Language

**Assertion** : A logical formula describing a set of valuations on program variables with some *interesting* property.
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)
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:
  \[ E ::= n \mid x \mid -E \mid E + E \mid \ldots \]
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.
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**:
  \[ E ::= n | x | -E | E + E | \ldots \]

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

- **Basic Propositions**:
  \[ B ::= E = E | E \geq E \]
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**:
  \[ B ::= E = E \mid E >= E \]

- **Assertions**:
  \[ A ::= true \mid B \mid \neg A \mid A \land A \mid \forall v \ A \]
Assertion Semantics

- As program executes, the valuation of variables (read state) changes
Assertion Semantics

- As program executes, the valuation of variables (read \textit{state}) changes.
- An execution of a program statement, transforms one state to another state.
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 \( s \).
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 $s \models A$.

\[
\begin{align*}
  s \models B & \text{ iff } \Lbracket B \Rbracket_s = \text{true} \\
  s \models \neg A & \text{ iff } s \not\models A \\
  s \models A_1 \land A_2 & \text{ iff } s \models A_1 \text{ and } s \models A_2 \\
  s \models \forall v. A & \text{ iff } \forall x \in \mathbb{Z}. s[v \leftarrow x] \models A
\end{align*}
\]

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

1 Introduction
   - Bird’s Eye View
   - Formal Introduction

2 Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3 Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4 Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
Example Program

Consider the following program written in our imperative language, annotated with assertions from our assertions language:
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)
```
Example Program

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

\[
\begin{align*}
&(\text{ensures } n \geq 0) \\
k &:= 0; \\
j &:= 1; \\
\text{while (} k \neq n \text{) } \\
&
\begin{align*}
k &:= k + 1; \\
j &:= 2 \cdot j;
\end{align*}
\end{align*}
\]

\(\text{(assert } j = 2^n)\)

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

- $\phi_1$ and $\phi_2$ are formulae in a base logic (FO logic for us)
A Hoare triple $\{ \phi_1 \} P \{ \phi_2 \}$ is a formula:

- $\phi_1$ and $\phi_2$ are formulae in a base logic (FO logic for us)
- $P$ is a program in our imperative language
A Hoare triple $\{\phi_1\}P\{\phi_2\}$ is a formula:

- $\phi_1$ and $\phi_2$ are formulae in a base logic (FO logic for us)
- $P$ is a program in our imperative language
- Note how programs and formulae in base logic are intertwined
A **Hoare triple** $\{\phi_1\}P\{\phi_2\}$ is a formula:

- $\phi_1$ and $\phi_2$ are formulae in a base logic (FO logic for us)
- $P$ is a program in our imperative language
- Note how programs and formulae in base logic are intertwined
- $\phi_1$: **Precondition**
A **Hoare triple** \( \{ \phi_1 \} P \{ \phi_2 \} \) is a formula:

- \( \phi_1 \) and \( \phi_2 \) are formulae in a base logic (FO logic for us)
- \( P \) is a program in our imperative language
- Note how programs and formulae in base logic are intertwined
- \( \phi_1 \): **Precondition** , \( \phi_2 \): **Postcondition**
A **Hoare triple** $\{ \phi_1 \} P \{ \phi_2 \}$ is a formula:

- $\phi_1$ and $\phi_2$ are formulae in a base logic (FO logic for us)
- $P$ is a program in our imperative language
- Note how programs and formulae in base logic are intertwined
- $\phi_1$: **Precondition**, $\phi_2$: **Postcondition**

Examples of syntactically correct Hoare triples:
A **Hoare triple** \( \{\phi_1\} P \{\phi_2\} \) is a formula:

- \(\phi_1\) and \(\phi_2\) are formulae in a base logic (FO logic for us)
- \(P\) is a program in our imperative language
- Note how programs and formulae in base logic are intertwined
- \(\phi_1\): **Precondition**, \(\phi_2\): **Postcondition**

Examples of syntactically correct Hoare triples:

- \(\{(n \geq 0) \land (n^2 > 28)\} \ m := n + 1; \ m := m \ast m \ \neg(m = 36)\)
A **Hoare triple** $\{\phi_1\}P\{\phi_2\}$ is a formula:

- $\phi_1$ and $\phi_2$ are formulae in a base logic (FO logic for us)
- $P$ is a program in our imperative language
- Note how programs and formulae in base logic are intertwined
- $\phi_1$: **Precondition**, $\phi_2$: **Postcondition**

Examples of syntactically correct Hoare triples:

- $\{(n \geq 0) \land (n^2 > 28)\} \ m := n + 1; \ m := m \ast m \ \{\neg(m = 36)\}$
- $\{\exists x, y.(y > 0) \land (n = x^y)\} \ n := n \ast (n + 1) \ \{\exists x, y.(n = x^y)\}$
The partial correctness specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state $s$ satisfying $\phi_1$. 

The partial correctness specification $\{\phi_1\} P \{\phi_2\}$ is valid iff starting from a state $s$ satisfying $\phi_1$,
- Whenever an execution of $P$ terminates in state $s'$, then $s' \models \phi_2$
Hoare Triple : Semantics

- The **partial correctness** specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state $s$ satisfying $\phi_1$,
  - Whenever an execution of $P$ terminates in state $s'$, then $s' \models \phi_2$
- The **total correctness** specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state $s$ satisfying $\phi_1$, 
Hoare Triple : Semantics

- The **partial correctness** specification \( \{ \phi_1 \} P \{ \phi_2 \} \) is valid iff starting from a state \( s \) satisfying \( \phi_1 \),
  - Whenever an execution of \( P \) terminates in state \( s' \), then \( s' \models \phi_2 \)

- The **total correctness** specification \( \{ \phi_1 \} P \{ \phi_2 \} \) is valid iff starting from a state \( s \) satisfying \( \phi_1 \),
  - Every execution of \( P \) terminates, and
Hoare Triple: Semantics

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

- The **partial correctness** specification \( \{ \phi_1 \} P \{ \phi_2 \} \) is valid iff starting from a state \( s \) satisfying \( \phi_1 \),
  - Whenever an execution of \( P \) terminates in state \( s' \), then \( s' \models \phi_2 \)

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

**Partial v/s Total Correctness**

For programs without loops, both semantics coincide
Hoare Triple : Semantics

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

**Partial v/s Total Correctness**

For programs without loops, both semantics coincide

We will stick to partial correctness semantics and not talk about . . . .
Assignment Rule
Assignment Rule

Program Construct

\[ E ::= x \mid n \mid E + E \mid E \mid \ldots \]

\[ P ::= x := E \]
Assignment Rule

Program Construct

\[ E ::= \ x \mid n \mid E + E \mid E \mid \ldots \]

\[ P ::= \ x := E \]

Inference Rule

\[ \{ \phi([x \leftarrow E]) \} \ x := E \ \{ \phi(x) \} \]

where, \( \phi([x \leftarrow E]) \) replaces every free occurrence of \( x \) in \( \phi \) by \( E \).
Assignment Rule

Program Construct

\[
E ::= x \mid n \mid E + E \mid E \mid \ldots
\]

\[
P ::= x := E
\]

Inference Rule

\[
\{\phi([x \leftarrow E])\} x := E \{\phi(x)\}
\]

where, \(\phi([x \leftarrow E])\) replaces every free occurrence of \(x\) in \(\phi\) by \(E\)

Example:
Assignment Rule

Program Construct

\[ E ::= x \mid n \mid E + E \mid E \mid \ldots \]
\[ P ::= x ::= E \]

Inference Rule

\[ \{ \phi([x \leftarrow E]) \} \quad x ::= E \quad \{ \phi(x) \} \]

where, \( \phi([x \leftarrow E]) \) replaces every free occurrence of \( x \) in \( \phi \) by \( E \)

Example:

\[ \{(z \cdot y > 5) \land (\exists x. y = x^x)\} \quad x ::= z \cdot y \quad \{(x > 5) \land (\exists x. y = x^x)\} \]
Assignment Rule

Program Construct

\[ E ::= x | n | E + E | E | \ldots \]
\[ P ::= x := E \]

Inference Rule

\[ \{ \phi([x \leftarrow E]) \} \ x := E \ {\phi(x)} \]

where, \( \phi([x \leftarrow E]) \) replaces every free occurrence of \( x \) in \( \phi \) by \( E \)

Example:

\[ \{ (z \cdot y > 5) \land (\exists x. y = x^x) \} \ x := z \cdot y \ \{ (x > 5) \land (\exists x. y = x^x) \} \]

(replace only free occurrences of \( x \) in \( \phi \))
Assignment Rule

Program Construct

\[ E ::= x \mid n \mid E + E \mid E \mid \ldots \]

\[ P ::= x := E \]

Inference Rule

\[ \{ \phi([x \leftarrow E]) \} \ x := E \ \{ \phi(x) \} \]

where, \( \phi([x \leftarrow E]) \) replaces every free occurrence of \( x \) in \( \phi \) by \( E \)

Example:

\[ \{(z \cdot y > 5) \land (\exists x. y = x^x)\} \ x := z \cdot y \ \{(x > 5) \land (\exists x. y = x^x)\} \]

(replace only free occurrences of \( x \) in \( \phi \))

Forward Rule ?
Assignment Rule

Program Construct

\[ E ::= x \mid n \mid E + E \mid E \mid \ldots \]

\[ P ::= x := E \]

Inference Rule

\[ \{\phi([x \leftarrow E])\} \ x := E \ \{\phi(x)\} \]

where, \( \phi([x \leftarrow E]) \) replaces every free occurrence of \( x \) in \( \phi \) by \( E \)

Example:

\[ \{(z \cdot y > 5) \land (\exists x. y = x^x)\} \ x := z \ast y \ \{(x > 5) \land (\exists x. y = x^x)\} \]

(replace only free occurrences of \( x \) in \( \phi \))

Forward Rule?

\[ \{\phi(x)\} \ x := E \ \{\exists x_0 \phi(x_0) \land x = E[x \leftarrow x_0]\} \]
Rule for Sequential Composition
Rule for Sequential Composition

Program Construct

\[ P ::= P; P \]
Rule for Sequential Composition

**Program Construct**

\[ P ::= P; P \]

**Inference Rule**

\[
\begin{array}{c}
\{ \phi \} \; P_1 \; \{ \eta \} \quad \{ \eta \} \; P_2 \; \{ \psi \} \\
\hline
\{ \phi \} \; P_1; P_2 \; \{ \psi \}
\end{array}
\]
Rule for Sequential Composition

Program Construct

\[ P ::= P; P \]

Inference Rule

\[
\frac{\{\phi\} \ P_1 \ \{\eta\} \ \{\eta\} \ P_2 \ \{\psi\}}{\{\phi\} \ P_1; P_2 \ \{\psi\}}
\]

Example:

\[
\begin{align*}
\{y + z > 4\} \ y & := y + z \ \{y > 3\} \\
\{y > 3\} \ x & := y + 2 \ \{x > 5\} \\
\{y + z > 4\} \ y & := y + z; \ x := y + 2 \ \{x > 5\}
\end{align*}
\]
Rule of Consequence
Rule of Consequence

Inference Rule

\[
\frac{\phi \Rightarrow \phi_1}{\{\phi\} P \{\psi_1\}} \quad \frac{\psi_1 \Rightarrow \psi}{\{\phi\} P \{\psi\}}
\]

\(\phi \Rightarrow \phi_1\) and \(\psi_1 \Rightarrow \psi\) are implications in underlying (FO) logic
Rule of Consequence

**Inference Rule**

\[
\frac{\phi \Rightarrow \phi_1 \quad \{\phi_1\} \quad P \quad \{\psi_1\}}{\psi_1 \Rightarrow \psi \quad \{\phi\} \quad P \quad \{\psi\}}
\]

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

Example:

\[
((y > 4) \land (z > 1)) \Rightarrow (y + z > 5) \quad \{y + z > 5\} \quad y := y + z \quad \{y > 5\} \quad (y > 5) \Rightarrow (y > 3)
\]

\[
\{(y > 4) \land (z > 1)\} \quad y := y + z \quad \{y > 3\}
\]
Rule of Consequence

Inference Rule

\[
\phi \Rightarrow \phi_1 \quad \{\phi_1\} \quad P \quad \{\psi_1\} \quad \psi_1 \Rightarrow \psi
\]

\[
\{\phi\} \quad P \quad \{\psi\}
\]

\[\phi \Rightarrow \phi_1 \text{ and } \psi_1 \Rightarrow \psi \] are implications in underlying (FO) logic

Example:

\[
((y > 4) \land (z > 1)) \Rightarrow (y + z > 5) \quad \{y + z > 5\} \quad y := y + z \quad \{y > 5\} \quad (y > 5) \Rightarrow (y > 3)
\]

\[
\{(y > 4) \land (z > 1)\} \quad y := y + z \quad \{y > 3\}
\]

- Weakest precondition ?
- Strongest postcondition ?
Rule for Conditional Branch

Program Construct

\[ E ::= n \mid x \mid -E \mid E + E \mid \ldots \]

\[ B ::= \text{true} \mid E = E \mid E \geq E \mid \neg B \mid B \land B \]

\[ P ::= \text{if } B \text{ then } P \text{ else } P \]
Rule for Conditional Branch

Program Construct

\[
E := n | x | -E | E + E | \ldots
\]

\[
B := \text{true} | E = E | E >= E | \neg B | B \land B
\]

\[
P := \text{if } B \text{ then } P \text{ else } P
\]

Inference Rule

\[
\begin{align*}
\{ \phi \land B \} & P_1 \{ \psi \} & \{ \phi \land \neg B \} & P_2 \{ \psi \} \\
\{ \phi \} & \text{if } B \text{ then } P_1 \text{ else } P_2 \{ \psi \}
\end{align*}
\]
Rule for Conditional Branch

Program Construct

\[ E := n | x | -E | E + E | \ldots \]
\[ B := \text{true} | E = E | E \geq E | \neg B | B \land B \]
\[ P := \text{if } B \text{ then } P \text{ else } P \]

Inference Rule

\[
\frac{\{\phi \land B\} \ P_1 \ \{\psi\} \quad \{\phi \land \neg B\} \ P_2 \ \{\psi\}}{\{\phi\} \text{ if } B \text{ then } P_1 \text{ else } P_2 \ \{\psi\}}
\]

Example:

\[
\{(y > 4) \land (z > 1)\} \quad y := y + z \quad \{y > 3\} \quad \{(y > 4) \land \neg(z > 1)\} \quad y := y_1 \quad \{y > 3\}
\]
\[
\quad \{y > 4\} \quad \text{if } (z > 1) \text{ then } y := y + z \quad \text{else } y := y - 1 \quad \{y > 3\}
\]
Rule for Conditional Branch

Program Construct

\[ E := \ n \ | \ x \ | \ -E \ | \ E + E \ | \ldots \]

\[ B := \ \text{true} \ | \ E = E \ | \ E \geq E \ | \ \neg B \ | \ B \land B \]

\[ P := \ \text{if } B \ \text{then } P \ \text{else } P \]

Inference Rule

\[
\frac{\{ \phi \land B \} \ P_1 \ \{ \psi \} \quad \{ \phi \land \neg B \} \ P_2 \ \{ \psi \}}{\{ \phi \} \ \text{if } B \ \text{then } P_1 \ \text{else } P_2 \ \{ \psi \}}
\]

Example:

\[
\{(y > 4) \land (z > 1)\} \ y := y + z \ \{y > 3\} \quad \{(y > 4) \land \neg(z > 1)\} \ y := y_1 \ \{y > 3\}
\]

\[
\{y > 4\} \ \text{if } (z > 1) \ \text{then } y := y + z \ \text{else } y := y - 1 \ \{y > 3\}
\]

What can we conclude if we have \( \{\phi \land B\} \ P_1 \ \{\psi_1\} \) and \( \{\phi \land \neg B\} \ P_2 \ \{\psi_2\} \)
Partial Correctness of Loops
Partial Correctness of Loops

Program Construct

\[ E ::= n \mid x \mid -E \mid E + E \mid \ldots \]

\[ B ::= \text{true} \mid E = E \mid E \geq E \mid \neg B \mid B \wedge B \]

\[ P ::= \text{while } B \ P \]
Partial Correctness of Loops

Program Construct

\[ E := \ n \ | \ x \ | \ -E \ | \ E + E \ | \ldots \]

\[ B := \ true \ | \ E = E \ | \ E >= E \ | \ \neg B \ | \ B \land B \]

\[ P := \text{while} \ B \ P \]

Inference Rule

\[
\begin{align*}
\{\phi \land B\} & \ P \ {\{\phi\}} \\
\{\phi\} & \text{while} \ B \ P \ {\{\phi \land \neg B\}}
\end{align*}
\]
Partial Correctness of Loops

Program Construct

\[
E := \ n \ | \ x \ | \ - E \ | \ E + E \ | \ldots \\
B := \text{true} \ | \ E = E \ | \ E \geq E \ | \neg B \ | \ B \land B \\
P := \text{while } B \ P
\]

Inference Rule

\[
\begin{align*}
\{ \phi \land B \} & \ P \ \{ \phi \} \\
\{ \phi \} & \text{while } B \ P \ \{ \phi \land \neg B \}
\end{align*}
\]

- \( \phi \) is **loop invariant**
- Partial Correctness Semantics:
Partial Correctness of Loops

Program Construct

\[ E ::= \ n \mid x \mid -E \mid E + E \mid \ldots \]

\[ B ::= \ true \mid E = E \mid E \geq E \mid \neg B \mid B \land B \]

\[ P ::= \text{while } B \ P \]

Inference Rule

\[ \{ \phi \land B \} \ P \ {\phi} \]
\[ \{ \phi \} \text{ while } B \ P \ {\phi \land \neg B} \]

- \( \phi \) is **loop invariant**
- Partial Correctness Semantics:
  - If loop does not terminate, Hoare triple is vacuously satisfied
Partial Correctness of Loops

Program Construct

\[ E ::= \ n \mid x \mid \neg E \mid E + E \mid \ldots \]

\[ B ::= \ true \mid E = E \mid E \geq E \mid \neg B \mid B \land B \]

\[ P ::= \text{while } B \ P \]

Inference Rule

\[
\frac{\{\phi \land B\} \quad P \quad \{\phi\}}{\{\phi\} \quad \text{while } B \ P \quad \{\phi \land \neg B\}}
\]

- $\phi$ is loop invariant

- Partial Correctness Semantics:
  - If loop does not terminate, Hoare triple is vacuously satisfied
  - If it terminates, $\phi \land \neg B$ must be satisfied after termination
Partial Correctness of Loops
Partial Correctness of Loops

\[
\begin{align*}
\{\phi \land B\} & \quad P \quad \{\phi\} \\
\{\phi\} \quad \text{while} \quad & \quad B \quad P \quad \{\phi \land \neg B\}
\end{align*}
\]
Partial Correctness of Loops

Inference Rule

\[
\begin{align*}
\{\phi \land B\} & \ P \ {\phi} \\
\{\phi\} & \ while \ B \ P \ {\phi \land \neg B}
\end{align*}
\]

Example:
Partial Correctness of Loops

Inference Rule

\[
\begin{align*}
\{\phi \land B\} & \Rightarrow P \Rightarrow \{\phi\} \\
\{\phi\} & \text{ while } B \Rightarrow P \Rightarrow \{\phi \land \neg B\}
\end{align*}
\]

Example:

\[
\begin{align*}
\{(y = x + z) \land (z \neq 0)\} & \Rightarrow x := x + 1; z := z - 1 \Rightarrow \{y = x + z\} \\
\{y = x + z\} & \text{ while } (z \neq 0) \Rightarrow x := x + 1; z := z - 1 \Rightarrow \{(y = x + z) \land (z = 0)\}
\end{align*}
\]
Partial Correctness of Loops

Inference Rule

\[
\begin{align*}
\{ \phi \land B \} & \quad P \quad \{ \phi \} \\
\{ \phi \} & \quad \text{while } B \quad P \quad \{ \phi \land \neg B \}
\end{align*}
\]

Example:

\[
\begin{align*}
\{(y = x + z) \land (z \neq 0)\} & \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} & \quad \text{while } (z! = 0) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land (z = 0)\}
\end{align*}
\]

\[
\begin{align*}
\{(y = x + z) \land \text{true}\} & \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} & \quad \text{while } (\text{true}) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land \text{false}\}
\end{align*}
\]
Partial Correctness of Loops

Inference Rule

\[
\begin{align*}
\{\phi \land B\} & \quad P \quad \{\phi\} \\
\{\phi\} & \quad \text{while } B \quad P \quad \{\phi \land \neg B\}
\end{align*}
\]

Example:

\[
\begin{align*}
\{(y = x + z) \land (z \neq 0)\} & \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} & \quad \text{while } (z! = 0) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land (z = 0)\}
\end{align*}
\]

\[
\begin{align*}
\{(y = x + z) \land \text{true}\} & \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} & \quad \text{while } (\text{true}) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land \text{false}\}
\end{align*}
\]
Partial Correctness of Loops

Inference Rule

\[
\frac{\{\phi \land B\} \quad P \quad \{\phi\}}{\{\phi\} \quad \text{while} \quad B \quad P \quad \{\phi \land \neg B\}}
\]

Example:

\[
\begin{align*}
\{(y = x + z) \land (z \neq 0)\} & \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} & \quad \text{while} \quad (z! = 0) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land (z = 0)\} \\
\{(y = x + z) \land \text{true}\} & \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} & \quad \text{while} \quad \text{(true)} \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land \text{false}\}
\end{align*}
\]

\[
\{\phi\} \quad \text{while} \quad \text{(true)} \quad P \quad \{\psi\} \quad \text{holds vacuously for all} \quad \phi, \quad P \quad \text{and} \quad \psi
\]
Partial Correctness of Loops

Inference Rule

\[
\begin{array}{c}
\{\phi \land B\} \quad P \quad \{\phi\} \\
\hline
\{\phi\} \quad \text{while } B \quad P \quad \{\phi \land \neg B\}
\end{array}
\]

Example:

\[
\{(y = x + z) \land (z \neq 0)\} \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} \quad \text{while } (z! = 0) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land (z = 0)\}
\]

\[
\{(y = x + z) \land \text{true}\} \quad x := x + 1; z := z - 1 \quad \{y = x + z\} \\
\{y = x + z\} \quad \text{while } (\text{true}) \quad x := x + 1; z := z - 1 \quad \{(y = x + z) \land \text{false}\}
\]

- \{\phi\} \quad \text{while } (\text{true}) \quad P \quad \{\psi\} \quad \text{holds vacuously for all } \phi, \quad P \quad \text{and } \psi
- can be proved using rule for loops and rules of strengthening (weakening) post(pre) -conditions
Summary of Axioms

- \( \{ \phi([x \leftarrow E]) \} \ x := E \{ \phi(x) \} \)
  
  Assignment

- \( \{ \phi \} P_1 \{ \eta \} \ {\eta} P_2 \{ \psi \} \)
  \[ \{ \phi \} P_1; P_2 \{ \psi \} \]
  
  Sequential Composition

- \( \{ \phi \land B \} P_1 \{ \psi \} \ {\phi \land \neg B} P_2 \{ \psi \} \)
  
  Conditional Statement

- \( \{ \phi \} \text{if } B \text{ then } P_1 \text{ else } P_2 \{ \psi \} \)
  
  Iteration

- \( \{ \phi \land B \} P \{ \psi \} \)
  \[ \{ \phi \} \text{while } B \ P \{ \psi \land \neg B \} \]
  
  Weakening pre-condition, Strenghtening post-condition

- \( \phi \Rightarrow \phi_1 \)
  \( \{ \phi_1 \} P \{ \psi_1 \} \)
  \( \psi_1 \Rightarrow \psi \)
  
  \( \{ \phi \} P \{ \psi \} \)
Some Structural Rules

Structural rules do not depend on program statements.

\[
\begin{align*}
\{\phi_1\} P\{\psi_1\} & \quad \{\phi_2\} P\{\psi_2\} \\
\{\phi_1 \land \phi_2\} P\{\psi_1 \land \psi_2\} \\
\{\phi_1\} P\{\psi_1\} & \quad \{\phi_2\} P\{\psi_2\} \\
\{\phi_1 \lor \phi_2\} P\{\psi_1 \lor \psi_2\} \\
\{\phi\} P\{\psi\} & \\
\{\exists v.\phi\} P\{\exists v.\psi\} \\
\{\phi\} P\{\psi\} & \\
\{\forall v.\phi\} P\{\forall v.\psi\}
\end{align*}
\]

Conjunction
Disjunction
Existential Quantification \((\nu \text{ is not free in } P)\)
Universal Quantification \((\nu \text{ is not free in } P)\)
Proving properties of simple programs

Let \( P \): Sequence of executable statements in bar

\[
\begin{align*}
P &::= k := 0; \\
& \quad \quad j := 1; \\
& \quad \quad \text{while } (k \neq n) \{ \\
& \quad \quad \quad k := k + 1; \\
& \quad \quad \quad j := 2 + j; \\
& \quad \quad \}
\end{align*}
\]

Our goal is to prove the validity of \( \{ n > 0 \} \) \( P \) \( \{ j = 1 + 2 \cdot n \} \)
A Hoare logic proof

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

\[
\begin{align*}
\{n > 0\} & \quad \text{Precondition} \\
k := 0 & \quad \text{Midcondition} \\
\{\varphi_1\} & \quad \text{Midcondition} \\
j := 1 & \\
\{\varphi_2\} & \\
\text{while } (k != n) \{ k := k+1; j := 2+j \} & \\
\{j = 1 + 2.n\} & \quad \text{Postcondition}
\end{align*}
\]

- How do we prove \( \{\varphi_2\} \quad \text{while } (k != n) \{ k := k+1; j := 2+j \} \{j = 1 + 2.n\} \)?
- Recall rule for loops requires a loop invariant
- “Guess” a loop invariant \( j = 1 + 2.k \)
A Hoare logic proof

To prove:
\{ \varphi_2 \} \text{ while } (k \neq n) \ k := k+1; \ j := 2+j \ \{ j = 1 + 2.n \}
using loop invariant \ (j = 1 + 2.k)\)

If we can show:
- \varphi_2 \Rightarrow (j = 1 + 2.k)
- \{(j = 1 + 2.k) \land (k \neq n)\} \ k := k+1; \ j := 2+j \ \{ j = 1 + 2.k \}
- ((j = 1 + 2.k) \land \neg(k \neq n)) \Rightarrow (j = 1 + 2.n)

then

By inference rule for loops
\{(j = 1 + 2.k) \land (k \neq n)\} \ k := k+1; \ j := 2+j \ \{ j = 1 + 2.k \}
\{j = 1 + 2.k\} \text{ while } (k \neq n) \ k := k+1; \ j := 2+j \ \{(j = 1 + 2.k) \land \neg(k \neq n)\}

By inference rule for strengthening precedents and weakening consequents
\varphi_2 \Rightarrow (j = 1 + 2.k)
\{j = 1 + 2.k\} \text{ while } (k \neq n) \ k := k+1; \ j := 2+j \ \{(j = 1 + 2.k) \land \neg(k \neq n)\}
((j = 1 + 2.k) \land \neg(k \neq n)) \Rightarrow (j = 1 + 2.n)
\{\varphi_2\} \text{ while } (k \neq n) \ k := k+1; \ j := 2+j \ \{(j = 1 + 2.n)\}
A Hoare logic proof

How do we show:

- $\varphi_2 \Rightarrow (j = 1 + 2.k)$
- $\{ (j = 1 + 2.k) \land (k \neq n) \} \quad k := k+1; \quad j := 2+j \quad \{ j = 1 + 2.k \}$
- $((j = 1 + 2.k) \land \neg(k \neq n)) \Rightarrow (j = 1 + 2.n)$

Note:

- $\varphi_2 \Rightarrow (j = 1 + 2.k)$ holds trivially if $\varphi_2$ is $(j = 1 + 2.k)$
- $((j = 1 + 2.k) \land \neg(k \neq n)) \Rightarrow (j = 1 + 2.n)$ holds trivially in integer arithmetic

Only remaining proof subgoal:

$\{ (j = 1 + 2.k) \land (k \neq n) \} \quad k := k+1; \quad j := 2+j \quad \{ j = 1 + 2.k \}$
A Hoare logic proof

To show:
\[ (j = 1 + 2.\,k) \land (k \neq n) \]  
\[ k := k+1; \; j := 2+j \quad \{ j = 1 + 2.\,k \} \]

Applying assignment rule twice
\[ \{2 + j = 1 + 2.\,k\} \quad j := 2+j \quad \{ j = 1 + 2k \} \]
\[ \{2 + j = 1 + 2.(k + 1)\} \quad k := k+1 \quad \{ 2 + j = 1 + 2.\,k \} \]

Simplifying and applying sequential composition rule
\[ \{1 + j = 2.\,k\} \quad j := 2+j \quad \{ j = 1 + 2k \} \]
\[ \{j = 1 + 2.\,k)\} \quad k := k+1 \quad \{1 + j = 2.\,k\} \]
\[ \{j = 1 + 2.\,k\} \quad k := k+1; \; j := 2+j \quad \{ j = 1 + 2k \} \]

Applying rule for strengthening precedent
\[ (j = 1 + 2.\,k) \land (k \neq n) \Rightarrow (j = 1 + 2.\,k) \]
\[ \{j = 1 + 2.\,k\} \quad k := k+1; \; j := 2+j \quad \{ j = 1 + 2k \} \]
\[ \{(j = 1 + 2.\,k) \land (k \neq n)\} \quad k := k+1; \; j := 2+j \quad \{ j = 1 + 2k \} \]
A Hoare logic proof

We have thus shown that with $\varphi_2$ as $(j = 1 + 2 \cdot k)$

\[
\{\varphi_2\} \text{ while } (k \neq n) \ k := k+1; \ j := 2+j \ \{j = 1 + 2 \cdot n\} \text{ is valid}
\]

Recall our template:

\[
\begin{align*}
\{ n > 0 \} & \quad \text{Precondition} \\
\quad k := 0 & \quad \text{Midcondition} \\
\quad \{ \varphi_1 \} & \quad \text{Midcondition} \\
\quad j := 1 & \quad \text{Midcondition} \\
\quad \{ \varphi_2 : j = 1 + 2 \cdot k \} & \\
\text{while } (k \neq n) \ k := k+1; \ j := 2+j & \quad \text{Postcondition} \\
\quad \{ j = 1 + 2 \cdot n \}
\end{align*}
\]

The only missing link now is to show

\[
\begin{align*}
\{ n > 0 \} & \quad k := 0 \quad \{ \varphi_1 \} \text{ and} \\
\{ \varphi_1 \} & \quad j := 1 \quad \{ j = 1 + 2 \cdot k \}
\end{align*}
\]
A Hoare logic proof

To show
\{ n > 0 \} k := 0 \{ \varphi_1 \} and
\{ \varphi_1 \} j := 1 \{ j = 1 + 2.k \}

Applying assignment rule twice and simplifying:
\{ 0 = k \} j := 1 \{ j = 1 + 2.k \}
\{ true \} k := 0 \{ 0 = k \}

Choose \varphi_1 as (k = 0), so \{ \varphi_1 \} j := 1 \{ j = 1 + 2.k \} holds.

Applying rule for strengthening precedent:
\( (n > 0) \Rightarrow true \)
\{ true \} k := 0 \{ \varphi_1 : k = 0 \}
\{ n > 0 \} k := 0 \{ \varphi_1 : k = 0 \}

We have proved partial correctness of function bar in Hoare Logic !!!
Is Hoare Logic **sound**?
Is Hoare Logic **sound**?

- Yes. Very much!
Is Hoare Logic **sound**?
- Yes. Very much!

Is Hoare Logic **complete**?
Is Hoare Logic **sound**?  
▶ Yes. Very much!

Is Hoare Logic **complete**?  
▶ Sort of..  
▶ **Relatively** Complete
Outline

1 Introduction
   • Bird’s Eye View
   • Formal Introduction

2 Preliminaries
   • A simple Imperative Language
   • A simple assertion Language
   • Assertion Semantics
   • Example Program

3 Hoare Logic
   • Hoare Triples: Syntax and Semantics
   • Axioms

4 Soundness and Completeness
   • Soundness
   • Relative Completeness
   • Weakest Precondition
Soundness of Hoare Logic
Soundness of Hoare Logic

- Hoare Logic has a **sound** proof system
Soundness of Hoare Logic

- Hoare Logic has a **sound** proof system
- That is, each theorem in Hoare Logic is valid:
Soundness of Hoare Logic

- Hoare Logic has a sound proof system
- That is, each theorem in Hoare Logic is valid:

\[ \vdash \{ \phi \} P \{ \psi \}, \text{ then } \models \{ \phi \} P \{ \psi \} \]
Soundness of Hoare Logic

- Hoare Logic has a sound proof system
- That is, each theorem in Hoare Logic is valid:

\[ \vdash \{ \phi \} P \{ \psi \} , \text{ then } \models \{ \phi \} P \{ \psi \} \]

- Follows from soundness of Hoare rules (axioms).
Soundness of Hoare Logic

- Hoare Logic has a sound proof system
- That is, each theorem in Hoare Logic is valid:

\[ \vdash \{\phi\} P\{\psi\} , \text{ then } \models \{\phi\} P\{\psi\} \]

- Follows from soundness of Hoare rules (axioms).
- Reason about the number of steps required to terminate loop for the loop rule.
Soundness of Hoare Logic

- Hoare Logic has a **sound** proof system
- That is, each theorem in Hoare Logic is valid:

  \[
  \text{If } \vdash \{\phi\} P \{\psi\}, \text{ then } \models \{\phi\} P \{\psi\}
  \]

- Follows from soundness of Hoare rules (axioms).
- Reason about the number of steps required to terminate loop for the *loop* rule.
- Then use induction on the structure of the proof tree.
Outline

1 Introduction
   - Bird’s Eye View
   - Formal Introduction

2 Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3 Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4 Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
Relative Completeness of Hoare Logic
Relative Completeness of Hoare Logic

Hoare logic is relatively complete.
Relative Completeness of Hoare Logic

Hoare logic is relatively complete.

**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*
Relative Completeness of Hoare Logic

Hoare logic is relatively complete.

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*

- First Order Logic is incomplete! (Kurt Gödel)
Hoare logic is relatively complete.

**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*

- First Order Logic is incomplete! (Kurt Gödel)
- The result uses *weakest pre-condition*
Outline

1 Introduction
   - Bird’s Eye View
   - Formal Introduction

2 Preliminaries
   - A simple Imperative Language
   - A simple assertion Language
   - Assertion Semantics
   - Example Program

3 Hoare Logic
   - Hoare Triples: Syntax and Semantics
   - Axioms

4 Soundness and Completeness
   - Soundness
   - Relative Completeness
   - Weakest Precondition
Weakest Precondition
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)
**Weakest Precondition**

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program \( P \) is executed, the resulting states satisfy a given post-condition \( \psi \). \((wp(P, \psi))\)

---

**Definition (Weakest Precondition)**

Given program \( P \) and postcondition \( \psi \), a weakest precondition \( wcp(P, \psi) \) is an assertion such that
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{wp(P, \psi)\} P \{\psi\}$
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{ wp(P, \psi) \} P \{ \psi \}$
- Is weakest, i.e., for any assertion $\phi$, if $\models \{ \phi \} P \{ \psi \}$, then $\models \phi \Rightarrow wp(P, \psi)$
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program \( P \) is executed, the resulting states satisfy a given post-condition \( \psi \). \( (wp(P, \psi)) \)

**Definition (Weakest Precondition)**

Given program \( P \) and postcondition \( \psi \), a weakest precondition \( wcp(P, \psi) \) is an assertion such that

- it is a precondition: \( \models \{wp(P, \psi)\} P \{\psi\} \)
- Is weakest, i.e., for any assertion \( \phi \), if \( \models \{\phi\} P \{\psi\} \), then \( \models \phi \Rightarrow wp(P, \psi) \)

- Weakest precondition:
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{wp(P, \psi)\} P \{\psi\}$
- Is weakest, i.e., for any assertion $\phi$, if $\models \{\phi\} P \{\psi\}$, then $\models \phi \Rightarrow wp(P, \psi)$

- Weakest precondition:
  - exists
Weakest Precondition

Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{wp(P, \psi)\} P \{\psi\}$
- Is weakest, i.e., for any assertion $\phi$, if $\models \{\phi\} P \{\psi\}$, then $\models \phi \Rightarrow wp(P, \psi)$

**Weakest precondition:**

- exists
- is unique, up to equivalence of assertions
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{wp(P, \psi)\} P \{\psi\}$
- Is weakest, i.e., for any assertion $\phi$, if $\models \{\phi\} P \{\psi\}$, then $\models \phi \Rightarrow wp(P, \psi)$

- **Weakest precondition:**
  - exists
  - is unique, upto equivalence of assertions
  - can be expressed in our Assertion logic
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{ wp(P, \psi) \} P \{ \psi \}$
- Is weakest, i.e., for any assertion $\phi$, if $\models \{ \phi \} P \{ \psi \}$, then $\models \phi \Rightarrow wp(P, \psi)$

**Weakest precondition:**

- exists
- is unique, upto equivalence of assertions
- can be expressed in our Assertion logic
- Can be proved to be a precondition using Hoare logic
  $\models \{ wp(P, \psi) \} P \{ \psi \}$
Weakest Precondition

- Intuitively, the **largest set of states** (represented as an assertion) starting from which if a program $P$ is executed, the resulting states satisfy a given post-condition $\psi$. ($wp(P, \psi)$)

**Definition (Weakest Precondition)**

Given program $P$ and postcondition $\psi$, a weakest precondition $wcp(P, \psi)$ is an assertion such that

- it is a precondition: $\models \{wp(P, \psi)\} P \{\psi\}$
- Is weakest, i.e., for any assertion $\phi$, if $\models \{\phi\} P \{\psi\}$, then $\models \phi \Rightarrow wp(P, \psi)$

**Weakest precondition:**

- exists
- is unique, upto equivalence of assertions
- can be expressed in our Assertion logic
- Can be proved to be a precondition using Hoare logic
- $\models \{wp(P, \psi)\} P \{\psi\}$
Existence: Weakest Preconditions for Basic Constructs
Existence: Weakest Preconditions for Basic Constructs

- Assignment:
  \[ wp(x := E, \psi) = \psi([x \leftarrow E]) \]
Existence: Weakest Preconditions for Basic Constructs

- Assignment:
  \[ wp(x := E, \psi) = \psi([x \leftarrow E]) \]

- Sequential Composition:
  \[ wp(P_1; P_2, \psi) = wp(P_1, wp(P_2, \psi)) \]
Existence: Weakest Preconditions for Basic Constructs

- **Assignment:**
  \[ wp(x := E, \psi) = \psi([x \leftarrow E]) \]

- **Sequential Composition:**
  \[ wp(P_1; P_2, \psi) = wp(P_1, wp(P_2, \psi)) \]

- **Conditional Statements:**
  \[ wp(\text{if } B \text{ then } P_1 \text{ else } P_2, \psi) = (B \land wp(P_1, \psi)) \lor (\neg B \land wp(P_2, \psi)) \]
Existence: Weakest Preconditions for Basic Constructs

- **Assignment:**
  \[ wp(x := E, \psi) = \psi([x \leftarrow E]) \]

- **Sequential Composition:**
  \[ wp(P_1; P_2, \psi) = wp(P_1, wp(P_2, \psi)) \]

- **Conditional Statements:**
  \[ wp(\text{if } B \text{ then } P_1 \text{ else } P_2, \psi) = (B \land wp(P_1, \psi)) \lor (\neg B \land wp(P_2, \psi)) \]

- **Loops:**
  \[ wp(\text{while } B \ P, \psi) = \bigwedge_{k \geq 0} \phi_k \]
Existence: Weakest Preconditions for Basic Constructs

- **Assignment:**
  \[ wp(x := E, \psi) = \psi([x \leftarrow E]) \]

- **Sequential Composition:**
  \[ wp(P_1; P_2, \psi) = wp(P_1, wp(P_2, \psi)) \]

- **Conditional Statements:**
  \[ wp(\text{if } B \text{ then } P_1 \text{ else } P_2, \psi) = (B \land wp(P_1, \psi)) \lor (\neg B \land wp(P_2, \psi)) \]

- **Loops:**
  \[ wp(\text{while } B \ P, \psi) = \bigwedge_{k\geq0} \phi_k \]
  - \[ \phi_0 = \text{true} \]
  - \[ \phi_{k+1} = (B \land wp(P, \phi_k)) \lor (\neg B \land \psi) \]
Existence: Weakest Preconditions for Basic Constructs

- **Assignment:**
  \[ wp(x := E, \psi) = \psi([x \gets E]) \]

- **Sequential Composition:**
  \[ wp(P_1; P_2, \psi) = wp(P_1, wp(P_2, \psi)) \]

- **Conditional Statements:**
  \[ wp(\text{if } B \text{ then } P_1 \text{ else } P_2, \psi) = (B \land wp(P_1, \psi)) \lor (\neg B \land wp(P_2, \psi)) \]

- **Loops:**
  \[ wp(\text{while } B \ P, \psi) = \bigwedge_{k \geq 0} \phi_k \]
  - \[ \phi_0 = \text{true} \]
  - \[ \phi_{k+1} = (B \land wp(P, \phi_k)) \lor (\neg B \land \psi) \]
  - Can be expressed as an assertion (Gödel’s $\beta$ function)
If $\models \{\phi\} P \{\psi\}$, then $\vdash \{\phi\} P \{\psi\}$
Back to Relative Completeness

If $\models \{ \phi \} P \{ \psi \}$, then $\vdash \{ \phi \} P \{ \psi \}$

- Existence and provability of weakest pre-condition:
  $\vdash \{ wp(P, \psi) \} P \{ \psi \}$
Back to Relative Completeness

If $\models \{\phi\} P \{\psi\}$, then $\vdash \{\phi\} P \{\psi\}$

- Existence and provability of weakest pre-condition:

  $\vdash \{wp(P, \psi)\} P \{\psi\}$

Here, $\{wp(P, \psi)\}$ is expressed in our base logic
Back to Relative Completeness

If \( \models \{ \phi \} P \{ \psi \} \), then \( \vdash \{ \phi \} P \{ \psi \} \)

- Existence and provability of weakest pre-condition:
  \[ \vdash \{ wp(P, \psi) \} P \{ \psi \} \]

Here, \( \{ wp(P, \psi) \} \) is expressed in our base logic

- From definition of weakest precondition:
  \[ \models \phi \Rightarrow \{ wp(P, \psi) \} P \{ \psi \} \]
Back to Relative Completeness

If \( \models \{ \phi \} P \{ \psi \} \), then \( \vdash \{ \phi \} P \{ \psi \} \)

- Existence and provability of weakest pre-condition:
  \( \vdash \{ wp(P, \psi) \} P \{ \psi \} \)

Here, \( \{ wp(P, \psi) \} \) is expressed in our base logic

- From definition of weakest precondition:
  \( \models \phi \Rightarrow \{ wp(P, \psi) \} P \{ \psi \} \)

- By assumption of completeness in base logic:
  \( \vdash \phi \Rightarrow \{ wp(P, \psi) \} P \{ \psi \} \)
If $\models \{\phi\} P \{\psi\}$, then $\vdash \{\phi\} P \{\psi\}$

- Existence and provability of weakest pre-condition:
  $\vdash \{wp(P, \psi)\} P \{\psi\}$

  Here, $\{wp(P, \psi)\}$ is expressed in our base logic

- From definition of weakest precondition:
  $\models \phi \Rightarrow \{wp(P, \psi)\} P \{\psi\}$

- By assumption of completeness in base logic:
  $\vdash \phi \Rightarrow \{wp(P, \psi)\} P \{\psi\}$

- Now, the proof for relative completeness goes like:
  $\vdash \phi \Rightarrow \{wp(P, \psi)\} P \{\psi\}$
Conclusions and Interesting Insights
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
- Given pre-condition $\phi$, postcondition $\psi$, can we come up with a program $P$ such that $\{\phi\}P\{\psi\}$ holds?
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
- Given pre-condition $\phi$, postcondition $\psi$, can we come up with a program $P$ such that $\{\phi\}P\{\psi\}$ holds?
- Invariant synthesis
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
- Given pre-condition $\phi$, postcondition $\psi$, can we come up with a program $P$ such that $\{\phi\}P\{\psi\}$ holds?
- Invariant synthesis

Other interesting followups:
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
- Given pre-condition $\phi$, postcondition $\psi$, can we come up with a program $P$ such that $\{\phi\}P\{\psi\}$ holds?

Other interesting followups:
- Axiomatic approach for (possibly recursive) procedures
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
- Given pre-condition $\phi$, postcondition $\psi$, can we come up with a program $P$ such that $\{\phi\} P \{\psi\}$ holds?
- Invariant synthesis

Other interesting followups:
- Axiomatic approach for (possibly recursive) procedures
- Concurrency
Conclusions and Interesting Insights

- Formalism to verify Hoare triples
  - Design by contract
- Hoare Logic not only gives a formalism to verify Hoare triples, but also describes a method to come up with *good enough* pre (post) conditions.
  - Can be used to design specifications / documentations of programs
- Given pre-condition \( \phi \), postcondition \( \psi \), can we come up with a program \( P \) such that \( \{ \phi \} P \{ \psi \} \) holds?
- Invariant synthesis

Other interesting followups:
- Axiomatic approach for (possibly recursive) procedures
- Concurrency
- Termination of loops: *variants*
References

C. A. R. Hoare.
An axiomatic basis for computer programming.
Commun. ACM 12, 10 (October 1969), 576-580.

Robert Floyd
Assigning Meanings to Programs

Supratik Chakraborty
A Short Introduction on Hoare Logic
www.cse.iitb.ac.in/~supratik/courses/cs615/msri_ss08.pdf

Radu Rugina
Lecture Notes, CS 611: Advanced Programming Languages
www.cs.cornell.edu/courses/cs611/2002fa/schedule.html