Introduction to Hoare Logic

Umang Mathur

Department of Computer Science
University of Illinois, Urbana Champaign

October 22, 2015

Outline

Outline

Bird's Eye View

Bird's Eye View

- 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

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 *partial correctness* of certain kinds of programs
  - Set of axioms
  - Rules of Inference
  - Underlying logic
- Motivation : Assertion checking in (sequential) programs (can do much more !)

Outline

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 :

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

- Boolean Conditions :

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

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 :

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

Outline

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 :

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

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

- Basic Propositions :

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

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 :

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

Outline

Assertion Semantics

Assertion Semantics

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

Assertion Semantics

- As program executes, the valuation of variables (read *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 $s \models A$.

$$
\begin{array}{rcr}
s \models B & \text{iff} & [\![B]\!]_s = \texttt{true} \\
s \models \neg A & \text{iff} & s \not\models A \\
s \models A_1 \wedge A_2 & \text{iff} & s \models A_1 \text{ and } s \models A_2 \\
s \models \forall v.A & \text{iff} & \forall x \in \mathbb{Z}.s[v \leftarrow x] \models A
\end{array}
$$

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

Outline

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:

```
_(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 ?

Outline

Hoare Triple : Syntax

Hoare Triple : Syntax

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

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)

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

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
- Note how programs and formulae in base logic are intertwined

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
- Note how programs and formulae in base logic are intertwined
- ϕ_1: **Precondition**

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
- Note how programs and formulae in base logic are intertwined
- ϕ_1: **Precondition** , ϕ_2: **Postcondition**

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
- Note how programs and formulae in base logic are intertwined
- ϕ_1: **Precondition** , ϕ_2: **Postcondition**

Examples of syntactically correct Hoare triples:

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
- Note how programs and formulae in base logic are intertwined
- ϕ_1: **Precondition** , ϕ_2: **Postcondition**

Examples of syntactically correct Hoare triples:

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

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
- Note how programs and formulae in base logic are intertwined
- 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

Hoare Triple : Semantics

- The partial correctness specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state s satisfying ϕ_1,

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 $s' \models \phi_2$

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 correctness specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state s satisfying ϕ_1,

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 correctness specification $\{\phi_1\}P\{\phi_2\}$ is valid iff starting from a state s satisfying ϕ_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 ϕ_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 ϕ_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 ϕ_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 ϕ_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 ϕ_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 ϕ_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 sematics and not talk about ....

Outline

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

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

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

Assignment Rule

Program Construct

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

$$P ::= x := E$$

Inference Rule

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

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

Example:

Assignment Rule

Program Construct

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

$$P ::= x := E$$

Inference Rule

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

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

Example:

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

Assignment Rule

Program Construct

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

$$P ::= x := E$$

Inference Rule

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

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

Example:

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

(replace only free occurences of x in ϕ)

Assignment Rule

Program Construct

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

$$P ::= x := E$$

Inference Rule

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

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

Example:

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

(replace only free occurences of x in ϕ)

Forward Rule ?

Assignment Rule

Program Construct

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

$$P ::= x := E$$

Inference Rule

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

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

Example:

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

(replace only free occurences of x in ϕ)

Forward Rule ?

- $\{\phi(x)\}\ x := E\ \{\exists x_0 \phi(x_0) \wedge x = E[x \leftarrow x_0]\}$

Rule for Sequential Composition

Rule for Sequential Composition

Program Construct

$$P ::= P; P$$

Rule for Sequential Composition

Program Construct

$$P ::= P; P$$

Inference Rule

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

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:

$$\frac{\{y + z > 4\}\ \mathtt{y} := \mathtt{y} + \mathtt{z}\ \{y > 3\} \qquad \{y > 3\}\ \mathtt{x} := \mathtt{y} + 2\ \{x > 5\}}{\{y + z > 4\}\ \mathtt{y} := \mathtt{y} + \mathtt{z}; \mathtt{x} := \mathtt{y} + 2\ \{x > 5\}}$$

Rule of Consequence

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

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

Example:

$$\frac{((y > 4) \wedge (z > 1)) \Rightarrow (y + z > 5) \qquad \{y + z > 5\}\ \mathtt{y} := \mathtt{y} + \mathtt{z}\ \{y > 5\} \qquad (y > 5) \Rightarrow (y > 3)}{\{(y > 4) \wedge (z > 1)\}\ \mathtt{y} := \mathtt{y} + \mathtt{z}\ \{y > 3\}}$$

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

Example:

$$\frac{((y > 4) \wedge (z > 1)) \Rightarrow (y + z > 5) \qquad \{y + z > 5\}\ \mathtt{y} := \mathtt{y} + \mathtt{z}\ \{y > 5\} \qquad (y > 5) \Rightarrow (y > 3)}{\{(y > 4) \wedge (z > 1)\}\ \mathtt{y} := \mathtt{y} + \mathtt{z}\ \{y > 3\}}$$

- Weakest precondition ?
- Strongest postcondition ?

Rule for Conditional Branch

Rule for Conditional Branch

Program Construct

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

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

$$P := \texttt{if } B \texttt{ then } P \texttt{ else } P$$

Rule for Conditional Branch

Program Construct

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

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

$$P := \texttt{if } B \texttt{ then } P \texttt{ else } P$$

Inference Rule

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

Rule for Conditional Branch

Program Construct

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

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

$$P := \texttt{if } B \texttt{ then } P \texttt{ else } P$$

Inference Rule

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

Example:

$$\frac{\{(y > 4) \wedge (z > 1)\}\ \texttt{y} := \texttt{y} + \texttt{z}\ \{y > 3\} \qquad \{(y > 4) \wedge \neg(z > 1)\}\ \texttt{y} := \texttt{y1}\ \{y > 3\}}{\{y > 4\}\ \texttt{if } (\texttt{z} > 1) \texttt{ then y} := \texttt{y} + \texttt{z} \texttt{ else y} := \texttt{y} - 1\ \{y > 3\}}$$

Rule for Conditional Branch

Program Construct

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

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

$$P := \texttt{if } B \texttt{ then } P \texttt{ else } P$$

Inference Rule

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

Example:

$$\frac{\{(y > 4) \wedge (z > 1)\}\ \texttt{y} := \texttt{y} + \texttt{z}\ \{y > 3\} \qquad \{(y > 4) \wedge \neg(z > 1)\}\ \texttt{y} := \texttt{y1}\ \{y > 3\}}{\{y > 4\}\ \texttt{if } (\texttt{z} > 1) \texttt{ then y} := \texttt{y} + \texttt{z} \texttt{ else y} := \texttt{y} - 1\ \{y > 3\}}$$

What can we conclude if we have $\{\phi \wedge B\}\ P_1\ \{\psi_1\}$ and $\{\phi \wedge \neg B\}\ P_2\ \{\psi_2\}$

Partial Correctness of Loops

Partial Correctness of Loops

Program Construct

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

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

$$P := \texttt{while}\ B\ P$$

Partial Correctness of Loops

Program Construct

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

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

$$P := \texttt{while}\ B\ P$$

Inference Rule

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

Partial Correctness of Loops

Program Construct

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

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

$$P := \texttt{while}\ B\ P$$

Inference Rule

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

- ϕ is **loop invariant**
- Partial Correctness Semantics:

Partial Correctness of Loops

Program Construct

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

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

$$P := \texttt{while}\ B\ P$$

Inference Rule

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

- ϕ is **loop invariant**
- Partial Correctness Semantics:
  - If loop does not terminate, Hoare triple is vacuously satisfied

Partial Correctness of Loops

Program Construct

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

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

$$P := \texttt{while}\ B\ P$$

Inference Rule

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

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

Partial Correctness of Loops

Partial Correctness of Loops

Inference Rule

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

Partial Correctness of Loops

Inference Rule

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

Example:

Partial Correctness of Loops

Inference Rule

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

Example:

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

Partial Correctness of Loops

Inference Rule

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

Example:

- $$\frac{\{(y = x + z) \wedge (z \neq 0)\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{z} != 0)\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge (z = 0)\}}$$
- $$\frac{\{(y = x + z) \wedge \texttt{true}\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{true})\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge \mathit{false}\}}$$

Partial Correctness of Loops

Inference Rule

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

Example:

- $$\frac{\{(y = x + z) \wedge (z \neq 0)\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{z} != 0)\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge (z = 0)\}}$$
- $$\frac{\{(y = x + z) \wedge \texttt{true}\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{true})\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge \mathit{false}\}}$$

Partial Correctness of Loops

Inference Rule

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

Example:

- $$\frac{\{(y = x + z) \wedge (z \neq 0)\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{z} != 0)\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge (z = 0)\}}$$
- $$\frac{\{(y = x + z) \wedge \texttt{true}\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{true})\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge \mathit{false}\}}$$

- $\{\phi\}\ \texttt{while}\ (\texttt{true})\ P\ \{\psi\}$ holds vacuously for all ϕ, P and ψ

Partial Correctness of Loops

Inference Rule

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

Example:

- $$\frac{\{(y = x + z) \wedge (z \neq 0)\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{z} != 0)\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge (z = 0)\}}$$
- $$\frac{\{(y = x + z) \wedge \texttt{true}\}\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{y = x + z\}}{\{y = x + z\}\ \texttt{while}\ (\texttt{true})\ \texttt{x} := \texttt{x} + 1; \texttt{z} := \texttt{z} - 1\ \{(y = x + z) \wedge \mathit{false}\}}$$

- $\{\phi\}\ \texttt{while}\ (\texttt{true})\ P\ \{\psi\}$ holds vacuosly for all ϕ, P and ψ
- can be proved using rule for loops and rules of strengthening (weakening) post(pre) -conditions

Summary of Axioms

- $$\frac{}{\{\phi([x \leftarrow E])\}\ x := E\ \{\phi(x)\}}$$ Assignment
- $$\frac{\{\phi\}\ P_1\ \{\eta\} \qquad \{\eta\}\ P_2\ \{\psi\}}{\{\phi\}\ P_1; P_2\ \{\psi\}}$$ Sequential Composition
- $$\frac{\{\phi \wedge B\}\ P_1\ \{\psi\} \qquad \{\phi \wedge \neg B\}\ P_2\ \{\psi\}}{\{\phi\}\ \texttt{if}\ B\ \texttt{then}\ P_1\ \texttt{else}\ P_2\ \{\psi\}}$$ Conditional Statement
- $$\frac{\{\phi \wedge B\}\ P\ \{\psi\}}{\{\phi\}\ \texttt{while}\ B\ P\ \{\psi \wedge \neg B\}}$$ Iteration
- $$\frac{\phi \Rightarrow \phi_1 \qquad \{\phi_1\}\ P\ \{\psi_1\} \qquad \psi_1 \Rightarrow \psi}{\{\phi\}\ P\ \{\psi\}}$$ Weakening pre-condition, Strenghtening post-condition

Some Structural Rules

Structural rules do not depend on program statements.

- $$\frac{\{\phi_1\}P\{\psi_1\} \qquad \{\phi_2\}P\{\psi_2\}}{\{\phi_1 \wedge \phi_2\}P\{\psi_1 \wedge \psi_2\}}$$ Conjunction
- $$\frac{\{\phi_1\}P\{\psi_1\} \qquad \{\phi_2\}P\{\psi_2\}}{\{\phi_1 \vee \phi_2\}P\{\psi_1 \vee \psi_2\}}$$ Disjunction
- $$\frac{\{\phi\}P\{\psi\}}{\{\exists v.\phi\}P\{\exists v.\psi\}}$$ Existential Quantification (v is not free in P)
- $$\frac{\{\phi\}P\{\psi\}}{\{\forall v.\phi\}P\{\forall v.\psi\}}$$ Universal Quantification (v is not free in P)

Proving properties of simple programs

Let P: Sequence of executable statements in `bar`

```
P ::    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:

$\{n > 0\}$ Precondition

`k := 0`

$\{\varphi_1\}$ Midcondition

`j := 1`

$\{\varphi_2\}$ Midcondition

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

$\{j = 1 + 2.n\}$ Postcondition

- How do we prove
  $\{\varphi_2\}$ `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:

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

using loop invariant $(j = 1 + 2.k)$

If we can show:

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

then

By inference rule for loops

$$\frac{\{(j = 1 + 2.k) \wedge (k \neq n)\} \texttt{ k := k+1; j:= 2+j } \{j = 1 + 2.k\}}{\{j = 1 + 2.k\} \texttt{ while (k != n) k := k+1; j:= 2+j } \{(j = 1 + 2.k) \wedge \neg(k \neq n)\}}$$

By inference rule for strengthening precedents and weakening consequents

$$\frac{\begin{array}{c}\varphi_2 \Rightarrow (j = 1 + 2.k) \\ \{j = 1 + 2.k\} \texttt{ while (k != n) k := k+1; j:= 2+j } \{(j = 1 + 2.k) \wedge \neg(k \neq n)\} \\ ((j = 1 + 2.k) \wedge \neg(k \neq n)) \Rightarrow (j = 1 + 2.n)\end{array}}{\{\varphi_2\} \texttt{ while (k != n) k := k+1; j:= 2+j } \{(j = 1 + 2.n)\}}$$

A Hoare logic proof

How do we show:

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

Note:

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

Only remaining proof subgoal:
$\{(j = 1 + 2.k) \wedge (k \neq n)\}$ `k := k+1; j:= 2+j` $\{j = 1 + 2.k\}$

A Hoare logic proof

To show:

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

Applying assignment rule twice

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

Simplifying and applying sequential composition rule

$$\frac{\begin{array}{c}\{1 + j = 2.k\} \quad \texttt{j := 2+j} \quad \{j = 1 + 2k\} \\ \{j = 1 + 2.k)\} \quad \texttt{k := k+1} \quad \{1 + j = 2.k\}\end{array}}{\{j = 1 + 2.k\} \quad \texttt{k := k+1; j := 2+j} \quad \{j = 1 + 2.k\}}$$

Applying rule for strengthening precedent

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

A Hoare logic proof

We have thus shown that with φ_2 as $(j = 1 + 2.k)$
$\{\varphi_2\}$ `while (k != n) k := k+1; j := 2+j` $\{j = 1 + 2.n\}$ is valid

Recall our template:

$\{n > 0\}$ Precondition
`k := 0`
$\{\varphi_1\}$ Midcondition
`j := 1`
$\{\varphi_2 : j = 1 + 2.k\}$ Midcondition
`while (k != n) k := k+1; j := 2+j`
$\{j = 1 + 2.n\}$ Postcondition

The only missing link now is to show
$\{n > 0\}$ `k := 0` $\{\varphi_1\}$ and
$\{\varphi_1\}$ `j := 1` $\{j = 1 + 2.k\}$

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` $\{j = 1 + 2.k\}$
$\{\text{true}\}$ `k := 0` $\{0 = k\}$

Choose φ_1 as $(k = 0)$, so $\{\varphi_1\}$ `j := 1` $\{j = 1 + 2.k\}$ holds.
Applying rule for strengthening precedent:

$$\frac{(n > 0) \Rightarrow \text{true} \qquad \{\text{true}\} \ \texttt{k := 0} \ \{\varphi_1 : k = 0\}}{\{n > 0\} \ \texttt{k := 0} \ \{\varphi_1 : k = 0\}}$$

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

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 :

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

Soundness of Hoare Logic

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

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

- 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 :

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

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

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

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)

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)
- The result uses *weakest pre-condition*

Outline

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 ψ. ($wp(P, \psi)$)

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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, if $\models \{\phi\}P\{\psi\}$, then $\models \phi \Rightarrow wp(P, \psi)$

- Weakest precondition:
  - exists
  - is unique, upto 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, 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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, 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 $\vdash \{wp(P, \psi)\}\ P\ \{\psi\}$

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 ψ. ($wp(P, \psi)$)

Definition (Weakest Precondition)

Given program P and postcondition ψ, 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 ϕ, 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 $\vdash \{wp(P, \psi)\}\ P\ \{\psi\}$

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:

$$wp(\texttt{if } B \texttt{ then } P_1 \texttt{ else } P_2, \psi) = (B \wedge wp(P_1, \psi)) \vee (\neg B \wedge wp(P_2, \psi))$$

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:

$$wp(\texttt{if } B \texttt{ then } P_1 \texttt{ else } P_2, \psi) = (B \wedge wp(P_1, \psi)) \vee (\neg B \wedge wp(P_2, \psi))$$

- Loops:

$$wp(\texttt{while } B\ P, \psi) = \bigwedge_{k \geq 0} \phi_k$$

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:

$$wp(\texttt{if } B \texttt{ then } P_1 \texttt{ else } P_2, \psi) = (B \wedge wp(P_1, \psi)) \vee (\neg B \wedge wp(P_2, \psi))$$

- Loops:

$$wp(\texttt{while } B\ P, \psi) = \bigwedge_{k \geq 0} \phi_k$$

  - $\phi_0 = \texttt{true}$
  - $\phi_{k+1} = (B \wedge wp(P, \phi_k)) \vee (\neg B \wedge \psi)$

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:

$$wp(\texttt{if } B \texttt{ then } P_1 \texttt{ else } P_2, \psi) = (B \wedge wp(P_1, \psi)) \vee (\neg B \wedge wp(P_2, \psi))$$

- Loops:

$$wp(\texttt{while } B\ P, \psi) = \bigwedge_{k \geq 0} \phi_k$$

  - $\phi_0 = \texttt{true}$
  - $\phi_{k+1} = (B \wedge wp(P, \phi_k)) \vee (\neg B \wedge \psi)$
  - Can be expressed as an assertion (Gödel's β function)

Back to Relative Completeness

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

Back to Relative Completeness

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

- Existence and provability of weakest pre-condition:

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

Back to Relative Completeness

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

- 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

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

- 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

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

- 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:

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

Back to Relative Completeness

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

- 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:

$$\dfrac{\dfrac{\vdots}{\phi \Rightarrow \{wp(P,\psi)\}P\{\psi\}} \qquad \dfrac{\vdots}{\{wp(P,\psi)\}P\{\psi\}}}{\{\phi\}P\{\psi\}}$$

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 ϕ, postcondition ψ, 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 ϕ, postcondition ψ, 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 ϕ, postcondition ψ, 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 ϕ, postcondition ψ, 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

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 ϕ, postcondition ψ, 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 ϕ, postcondition ψ, 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 I


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


Robert Floyd
Assigning Meanings to Programs
Program Verification, Studies in Cognitive Systems (1993-01-01)


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