Verification

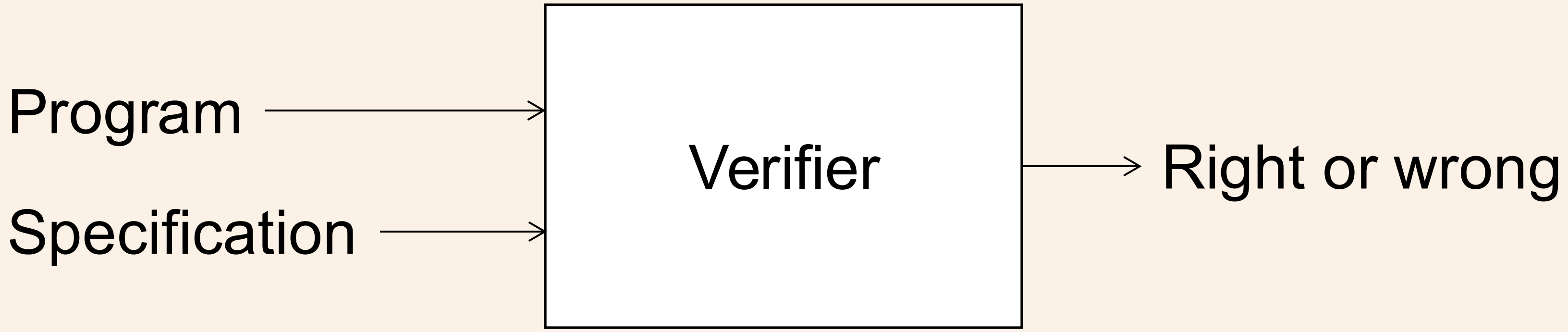
Sergey Mechtaev

Peking University

mechtaev@pku.edu.cn

Formal Verification

Definition. The act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property.



Applications

- Safety-critical systems (e.g. medical software, nuclear reactor controllers, autonomous vehicles)
- Core system components (e.g. device drivers)
- Security (e.g. ATM software, cryptographic algorithms)
- Hardware verification (e.g. processors)

Hoare Triple

- C is a program
- P and Q are predicates over program variables
- Hoare triple: given a state that satisfies preconditions P, executing a program C results in a state that satisfies postconditions Q:
$$\{P\}\, C\, \{Q\}$$
- Example: $\{x \geq 0\}\, x := x + 1\, \{x > 0\}$

Partial Correctness

- The meaning of $\{P\}\ C\ \{Q\}$ is as follows:
  - If we begin executing c in a state satisfying P,
  - and if C terminates,
  - then its final state will satisfy Q
- The specification says nothing about:
  - Executions that do not terminate (i.e., diverge)
  - Executions that do not begin in P
- Goal of verification: prove that $\{P\}\ C\ \{Q\}$ is valid

Total Correctness

- Total correctness is a stronger statement, written:

$$[P]C\ [Q]$$

- The meaning of $[P]C\ [Q]$ is:
  - If we begin executing C in a state satisfying P,
  - then c terminates,
  - and its final state will satisfy Q
- Total correctness introduces another obligation for verification

Examples

- $\{true\}\ C\ \{Q\}$ – if C terminates, then Q holds
- $[true]\ C\ [Q]$ – C terminates, and Q always holds after
- $[P]\ C\ [true]$ – if C starts in P, then C terminates
- $[true]\ C\ [true]$ – C terminates
- $\{x > 0\}\ \textbf{while}\ 0 < x\ \textbf{do}\ x := x + 1\ \{false\}$ – C does not terminate when starting in x > 0
- $\{true\}\ C\ \{false\}$ – C does not terminate

Poll: Is the following a valid Hoare triple?

- $\{x = 2\}\textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\{x = 0\}$
- $\{x = 0 \land y = 1\}\ x := x + 1\ \{x = 1\ \land\ y = 2\}$
- $\{true\}\ \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1\ \{x \leq 0\}$
- $[true]\textbf{while}\ x > 0\ \textbf{do}\ x := x + 1\ [x \leq 0]$

Hoare Logic: Assignments

- Hoare Logic is a logic for deriving new triples from existing ones
- The rule for assignment statements:

$$\text{ASGN}\ \frac{}{\{Q[a/x]\}\ x := \ a\{Q\}}$$

- Read $Q[a/x]$ as “Q with a substituted for x”
- Examples:
  - $\{1 = y\}\ x := \ 1\ \{x = y\}$
  - $\{x + 1 = n\}\ x := \ x + 1\ \{x = n\}$

Hoare Logic: Strengthening

- The rule for precondition strengthening:

$$\text{PRE}\ \frac{\{P'\}C\{Q\} \quad P \Rightarrow P'}{\{P\}C\{Q\}}$$

- Example: proving that $\{true\}\ x := 1\ \{x = 1\}$

$$\text{PRE}\ \frac{\text{ASGN}\ \dfrac{}{\{1 = 1\}x := 1\{x = 1\}} \qquad true \Rightarrow 1 = 1}{\{true\}x := 1\{x = 1\}}$$

Hoare Logic: Weakening

- The rule for weakening the postcondition:

$$\text{POST}\ \frac{\{P\}C\{Q'\} \quad Q' \Rightarrow Q}{\{P\}C\{Q\}}$$

- The rule for consequence (combines PRE and POST):

$$\text{CONSEQ}\ \frac{P \Rightarrow P' \quad \{P'\}C\{Q'\} \quad Q' \Rightarrow Q}{\{P\}C\{Q\}}$$

Hoare Logic: Composition

- Given triples for C_1 and C_2 , this gives us one for $C_1; C_2$:

$$\text{SEQ}\ \frac{\{P\}C_1\{P'\} \qquad \{P'\}C_2\{Q\}}{\{P\}C_1; C_2\{Q\}}$$

Example: proving correctness of swap

- ASGN: $\{x = x' \land y = y'\}\, t := x\, \{t = x' \land y = y'\}$
- ASGN: $\{t = x' \land y = y'\}\, x := y\, \{t = x' \land x = y'\}$
- ASGN: $\{t = x' \land x = y'\}\, y := t\, \{y = x' \land x = y'\}$
- SEQ (1,2):

$$\begin{array}{l} \{x = x' \land y = y'\} \\ t := x; x := y \\ \{t = x' \land x = y'\} \end{array}$$

- SEQ (3,4):

$$\begin{array}{l} \{x = x' \land y = y'\} \\ t := x; x := y; y := t \\ \{y = x' \land x = y'\} \end{array}$$

Hoare Logic: Conditional

- Proving: $\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2$
- At the beginning of the true branch, we know that B holds, in the false branch, ¬B must hold:

$$\text{IF}\ \frac{\{P \wedge B\}C_1\{Q\} \qquad \{P \wedge \neg B\}C_2\{Q\}}{\{P\}\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2\{Q\}}$$

Hoare Logic: While Loop

- To prove triples for loops, we need **loop invariant** (condition that holds before the loop, and is preserved by each iteration)

$$\text{WHILE}\ \frac{\{P \land B\}\ C\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{P \land \neg B\}}$$

Example: loop invariant

- Want to prove that

$$\{true\}$$
$$r := x; q := 0;$$
$$\textbf{while } y \leq r \textbf{ do } r := r - y; q := q + 1$$
$$\{r < y \land x = r + (q \times y)\}$$

- Guess loop invariant:

$$P:\ x = r + (q \times y)$$

Example: loop invariant

- We are obligated to show that
$$\{x = r + (q \times y) \wedge y \leq r\}$$
$$r \coloneqq r - y; q \coloneqq q + 1$$
$$\{x = r + (q \times y)\}$$
- $\{x = r + (q + 1) \times y\} q \coloneqq q + 1 \{x = r + q \times y\}$
- $\{x = r - y + (q + 1) \times y\} r := r - y \{x = r + (q + 1) \times y\}$
- $x = r + (q \times y) \wedge y \leq r \Rightarrow$
$$x = r - y + (q + 1) \times y$$

Java Modeling Language (JML)

```
//@ requires 0 < amount && amount + balance < MAX_
BALANCE;
//@ ensures balance == \old(balance) + amount;
public void credit(final int amount) {
    this.balance += amount;
}

//@ requires 0 < amount && amount <= balance;
//@ ensures balance == \old(balance) - amount;
public void debit(final int amount) {
    this.balance -= amount;
}
```

Weakest Precondition

- Given an assertion Q and program C, we'll describe a function:
  - That is a predicate transformer: produces another assertion
  - Assertion for the corresponding precondition P for C
  - P guaranteed to be the weakest such assertion
- This is the weakest precondition predicate transformer $wp(C, Q)$:
  - The triple $\{wp(C, Q)\}\ C\ \{Q\}$ is valid
  - For any P where $\{P\}\ c\ \{Q\}$ is valid, $P \Rightarrow wp(c, Q)$

Strongest Postcondition

- The strongest postcondition predicate transformer $\mathrm{sp}(C, P)$:
  - The triple $\{P\}\, C\, \{sp(C, P)\}$ is valid
  - For any Q where $\{P\}\, C\, \{Q\}$ is valid, $sp(C, P) \Rightarrow Q$
- Can be computed using symbolic execution (e.g. start symbolic execution with path condition P)

Verification Condition

Definition. A logical formula such that its validity means some aspect of program correctness.

To check $\{P\}C\{Q\}$, weakest precondition allows us to:

- Start with a desired postcondition
- Propagate backwards to precondition P that must hold
- Verify that $P \Rightarrow wp(C, Q)$

Weakest Precondition Computation

- Assignment
$$wp(x := a, Q) = Q[a/x]$$
- Sequence:
$$wp(C_1; C_2, Q) = wp(C_1, wp(C_2, Q))$$
- Conditional:
$$wp(\textbf{if } b \textbf{ then } C_1 \text{ else } C_2, Q) =$$
$$(b \Rightarrow wp(C_1, Q)) \wedge (\neg b \Rightarrow wp(C_2, Q))$$

Weakest Precondition for Loops

- Equivalent:
$\textbf{while}\ b\ \textbf{do}\ c\ \equiv\ \textbf{if}\ b\ \textbf{then}\ c;\ \textbf{while}\ b\ \textbf{do}\ c\ \textbf{else}\ \textbf{skip}$
- $wp(\textbf{if}\ b\ \textbf{then}\ c;\ \textbf{while}\ b\ \textbf{do}\ c\ \textbf{else}\ \textbf{skip}, Q) = \big(b\ \rightarrow\ wp(c;\ \textbf{while}\ b\ \textbf{do}\ c, Q)\big) \wedge\ (\neg b\ \rightarrow\ Q) = \Big(b\ \rightarrow\ wp\big(c, wp(\textbf{while}\ b\ \textbf{do}\ c, Q)\big)\Big) \wedge\ (\neg b\ \rightarrow\ Q) = (b \rightarrow\ wp(c, wp(if\ b\ \textbf{then}\ c;\ \textbf{while}\ b\ \textbf{do}\ c\ \textbf{else}\ \textbf{skip}, Q)))\ \wedge\ (\neg b\ \rightarrow\ Q)$
- Infinite unrolling…

Approximate Weakest Precondition

- In general, we can't always compute wp for loops
- Instead, we'll approximate it with help from annotations
- Now we'll assume loops have the syntax:

$$\textbf{while}\ b\ \textbf{do}\ \{I\}\ c$$

- I is a loop invariant provided by the programmer
- The approximate wp for while will still be a valid precondition
- But it may not be the weakest precondition: even if $\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{Q\}$ is valid, it might not be that: $P \;\Rightarrow\; wp(\textbf{while}\ \{I\}\ b\ \textbf{do}\ c, Q)$

Approximate wp: While Loop

- If we define

$$wp(\textbf{while}\ \{I\}\ b\ \textbf{do}\ C, Q)\ =\ I$$

- Then we still need to show that
  - $I\ \land\ \neg b$ establishes Q
  - I is a loop invariant

Checking Verification Condition

$$\{P\}\ C\ \{Q\}$$

Verify that for all inputs $P \Rightarrow wp(C, Q)$

Check satisfiability of VC: $\neg(P \Rightarrow wp(C, Q))$