Program verification using Hoare Logic¹
Automated Reasoning - Guest Lecture

Petros Papapanagiotou
pe.p@ed.ac.uk

Part 1 of 2

¹Contains material from Mike Gordon’s slides: http://www.cl.cam.ac.uk/~mjcg/HL
A simple “while” programming language

- Sequence: a ; b
- Skip (do nothing): SKIP
- Variable assignment: X := 0
- Conditional: IF cond THEN a ELSE b FI
- Loop: WHILE cond DO c OD
Given some $X$

$$Y := 1 ;$$
$$Z := 0 ;$$
$$\text{WHILE } Z \neq X \text{ DO}$$
$$\qquad Z := Z + 1 ;$$
$$\qquad Y := Y \times Z$$
$$\text{OD}$$

$$\{ Y = X! \}$$

*How do you know for sure?*
Formal Methods

- **Formal Specification:**
  - Use mathematical notation to give a precise description of what a program should do

- **Formal Verification:**
  - Use logical rules to mathematically prove that a program satisfies a formal specification

- **Not a panacea:**
  - Formally verified programs may still not work!
  - Must be combined with testing
Modern use

Some use cases:

- 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)
Formal Verification

Requires programming language *semantics*

*What does it mean to execute a command C?*

*How does it affect the State?*

(State = map of memory locations to values)
Formal Verification

- **Denotational** semantics: construct *mathematical objects* that describe the meaning
  - Programs = functions: $[[C]] : State \rightarrow State$

- **Operational** semantics: describe the *steps of computation* during program execution
  - Small-step (only one transition): $\langle C, \sigma \rangle \rightarrow \langle C', \sigma' \rangle$
  - Big-step (entire transition to final value): $\langle C, \sigma \rangle \downarrow \sigma'$

- **Axiomatic** semantics: define *axioms and rules of some logic* of programs
  - Hoare Logic $\{P\} \ C \ {Q}$
Floyd-Hoare Logic and Partial Correctness Specification

By Charles Antony (“Tony”) Richard Hoare with original ideas from Robert Floyd - 1969

- **Specification**: Given a state that satisfies *preconditions P*, executing a *program C* (and assuming it terminates) results in a state that satisfies *postconditions Q*
- “Hoare triple”:

\[ \{P\} C \{Q\} \]

e.g.:

\[ \{X = 1\} \ x := x + 1 \{X = 2\} \]
Correctness

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

Partial correctness + termination = Total correctness
Trivial Specifications

\{P\} \ C \ \{T\}

\{F\} \ C \ \{Q\}
Formal specification can be tricky!

- Specification for the maximum of two variables:

\[
\{ T \} \ C \{ Y = \text{max}(X, Y) \}
\]

- \( C \) could be:

\[
\text{IF } X \geq Y \text{ THEN } Y := X \text{ ELSE SKIP FI}
\]

- \textit{But} \( C \) could also be:

\[
\text{IF } X \geq Y \text{ THEN } X := Y \text{ ELSE SKIP FI}
\]

- Or even:

\[
Y := X
\]

- Better use “auxiliary” variables (i.e. not program variables) \( x \) and \( y \):

\[
\{ X = x \land Y = y \} \ C \{ Y = \text{max}(x, y) \}
\]
Hoare Logic

- A deductive proof system for Hoare triples \( \{P\} \ C \ \{Q\} \)
- Can be used for verification with forward or backward chaining
  - Conditions \( P \) and \( Q \) are described using FOL
  - Verification Conditions (VCs): What needs to be proven so that \( \{P\} \ C \ \{Q\} \) is true?
  - Proof obligations or simply proof subgoals: Working our way through proving the VCs
Hoare Logic Rules

- Similar to FOL inference rules
- One for each programming language construct:
  - Assignment
  - Sequence
  - Skip
  - Conditional
  - While
- Rules of consequence:
  - Precondition strengthening
  - Postcondition weakening
Assignment Axiom

\[
\{Q[E/V]\} \ V := E \ \{Q\}
\]

- Example:

\[
\{X + 1 = n + 1\} \ X := X + 1 \ \{X = n + 1\}
\]

- Backwards!?
  - Why not \(\{P\} \ V := E \ \{P[V/E]\}\)?
    - because then: \(\{X = 0\} \ X := 1 \ \{X = 0\}\)
  - Why not \(\{P\} \ V := E \ \{P[E/V]\}\)?
    - because then: \(\{X = 0\} \ X := 1 \ \{1 = 0\}\)
Sequencing Rule

\[
\begin{array}{c}
\{P\} \ C_1 \ \{Q\} \\
\{Q\} \ C_2 \ \{R\}
\end{array}
\]

\[
\{P\} \ C_1 \ ; \ C_2 \ \{R\}
\]

Example (Swap \(X, Y\)):
\[
S := X ; \ X := Y ; \ Y := S
\]

\[
\{X = x \land Y = y\} \ S := X \ \{S = x \land Y = y\} \quad (1)
\]

\[
\{S = x \land Y = y\} \ X := Y \ \{S = x \land X = y\} \quad (2)
\]

\[
\{S = x \land X = y\} \ Y := S \ \{Y = x \land X = y\} \quad (3)
\]

\[
\begin{array}{c}
\{X = x \land Y = y\} \ S := X ; \ X := Y \ \{S = x \land X = y\} \\
\{X = x \land Y = y\} \ S := X ; \ X := Y ; \ Y := S \ \{Y = x \land X = y\}
\end{array}
\]
Skip Axiom

\{P\} \text{SKIP} \{P\}
Conditional Rule

\[
\{ P \land S \} \quad C_1 \quad \{ Q \} \quad \{ P \land \neg S \} \quad C_2 \quad \{ Q \} \\
\{ P \} \quad \text{IF} \quad S \quad \text{THEN} \quad C_1 \quad \text{ELSE} \quad C_2 \quad \text{FI} \quad \{ Q \} 
\]

- Example (Max X Y):

\[
\begin{align*}
\{ X \geq X \land X \geq Y \} & \quad \text{MAX} := X \quad \{ \text{MAX} \geq X \land \text{MAX} \geq Y \} \\
\{ T \land X \geq Y \} & \quad \text{MAX} := X \quad \{ \text{MAX} \geq X \land \text{MAX} \geq Y \} \quad \text{(4)} \\
\{ Y \geq X \land Y \geq Y \} & \quad \text{MAX} := Y \quad \{ \text{MAX} \geq X \land \text{MAX} \geq Y \} \\
\{ T \land \neg(X \geq Y) \} & \quad \text{MAX} := Y \quad \{ \text{MAX} \geq X \land \text{MAX} \geq Y \} \quad \text{(5)}
\end{align*}
\]

\[
\{ T \} \quad \text{IF} \quad X \geq Y \quad \text{THEN} \quad \text{MAX} := X \quad \text{ELSE} \quad \text{MAX} := Y \quad \text{FI} \quad \{ \text{MAX} \geq X \land \text{MAX} \geq Y \} \quad \text{(6)}
\]
Summary

- **Formal Verification**: Use logical rules to mathematically prove that a program satisfies a formal specification
- **Programming language semantics**
  - *denotational, operational, axiomatic*
- **Specification using Hoare triples** $\{P\} \; C \; \{Q\}$
  - Preconditions $P$
  - Program $C$
  - Postconditions $Q$
- **Hoare Logic**: A deductive proof system for Hoare triples
- **Logical Rules**:
  - One for each program construct
- **Partial correctness + termination = Total correctness**
- Precondition strengthening
- Postcondition weakening
- WHILE loops + invariants

To be continued...
Recommended reading

Theory:

▶ Mike Gordon, *Background Reading on Hoare Logic*,
▶ Huth & Ryan, Sections 4.1-4.3 (pp. 256-292)
▶ Nipkow & Klein, Section 12.2.1 (pp. 191-199)

Practice:

▶ Isabelle’s Hoare Logic library: http://isabelle.in.tum.de/dist/library/HOL/HOL-Hoare
▶ Tutorial exercise