CSC2/455 Software Analysis and Improvement
Introduction to Hoare Logic

Sreepathi Pai
April 24, 2023

URCS
Outline

Logics

A Logic for Proofs of Programs

Program Verification using Hoare Logic

Postscript
Outline

Logics

A Logic for Proofs of Programs

Program Verification using Hoare Logic

Postscript
Logic

- OED Definition: Reasoning conducted or assessed according to strict principles of validity.

- Particularly relevant to this lecture:
  - A particular system or codification of the principles of proof and inference.
Propositional Logic

- Recall, propositions (identified by symbols)
  - The connectives $\lor$, $\land$, $\Rightarrow$, $\iff$ and the operation $\neg$
- Tautologies
  - A formula that is always true
- Contradiction
  - A formula that is always false
- Equivalence: two formulae $A$ and $B$ are equivalent if $A \iff B$ is a tautology
- "Proof" technique in propositional logic
  - Enumerate all possible values of variables and check if the final result is always true
Equivalences

- $p \iff q$ is equivalent to $\neg q \iff \neg p$
  - contrapositive (theorem)
- $p \implies q$ is not necessarily equivalent to $q \implies p$
  - converse
Valid Arguments

\[ P_1 \]
\[ P_2 \]
\[ \ldots \]
\[ P_n \]
\[ \implies \]
\[ P_{n+1} \]

- An argument is valid if and only if
  \[ P_1 \land P_2 \land \cdots \land P_n \implies P_{n+1} \] is a tautology
  - this means that \[ P_1 \land P_2 \land \cdots \land P_n \land P_{n+1} \] is true
Rules of Inference: Modus Ponens

\[ p \rightarrow q \]

- \( (p \land (p \rightarrow q)) \rightarrow q \) is a tautology
- Example:
  - \( p \) is “it is raining”
  - \( p \rightarrow q \) is “if it is raining, roads are wet”
  - \( q \), so “roads are wet”
Rules of Inference: Modus Tollens

\[ p \implies q \]
\[ \neg q \]
\[ \therefore \neg p \]

- \((p \implies q) \land (\neg q)) \implies \neg p\) is a tautology

Example:
- \(p \implies q\) is “if \(a\) is even, \(a + 1\) is odd”
- \(\neg q\) is “\(a + 1\) is not odd”
- \(\neg p\), so “\(a\) is not even”
Invalid Rule of Inference: Affirming the Consequent

\[ p \implies q \]

\[ q \]

\[ \therefore p \]

\[ ((p \implies q) \land q) \implies p \text{ is not a tautology} \]
Proof System for Propositional Logic

- System L
- Lines of proof in this system must be
  - an axiom of L (an axiom of L is a tautology)
  - an application of Modus Ponens
  - a hypothesis (a hypothesis $G_n$ is assumed to be true)
  - a lemma (a previously proven theorem)
- The last line of a proof is a theorem
  - $G_1, G_2, ..., G_n \vdash_L A$
- This proof system is both:
  - Sound: Only tautologies can be proved
  - Complete: All tautologies can be proved

From Hirst and Hirst, A Primer for Logic and Proof.
Outline

Logics

A Logic for Proofs of Programs

Program Verification using Hoare Logic

Postscript
Floyd-Hoare Logic

Developed by Robert Floyd and Tony Hoare in the 1960s.

\{ P \} C \{ Q \}

- $P$ is a precondition
- $C$ is a statement, function or program
- $Q$ is a postcondition
- Both $P$ and $Q$ are logical statements, e.g., what you would put in an assert

Read as: If $P$ holds, and $C$ executes (and terminates), then $Q$ holds. $P$ and $Q$ are assertions, usually over program state, and usually we need to prove that $Q$ holds.
Recall: Partial and Total Correctness

- If \( C \) does not terminate, \( Q \) may or may not be true
  - This is the notion of partial correctness
- If \( C \) can be shown (formally) to terminate, then we achieve a proof of total correctness

Total correctness = Termination + Partial Correctness
Some examples of assertions

- \{X = 1\} Y := X \{Y = 1\}
- \{X = 1\} Y := X \{Y = 2\}
- \{\text{true}\} C\{Q\}
- \{P\} C\{\text{true}\}
- \{P\} C\{\text{false}\}

Note: not all of the above are valid, they are just assertions to be checked.
(informally) Proofs at the level of rigour that even a computer could understand!

Usually, each step in the proof is explicitly annotated as to how it was obtained from the previous steps

- Makes it easy to check (esp. for computers)
- Either the use of an axiom or a rule of inference

Painful to construct by hand

- Interactive proof assistants like Coq and Isabelle usually make it more fun
- (if you’ve disliked writing proofs, try them!)
The assignment axiom of Hoare Logic

• The assignment axiom states that
  - \( \vdash \{P[E/V]\} V := E \{P\} \)
• \(P[E/V]\) is read as \(P\) with all instances of \(V\) replaced by \(E\)
  - \(P\) with \(E\) for \(V\)
  - \(\{X = 1\}[Y/X]\) leads to \(\{Y = 1\}\)

• Usage example: if \(X = 6\), prove \(Y > 15\) after \(Y := X * 3\)
  - Postcondition \(P\) to prove: \(\{Y > 15\}\)
  - Use assignment axiom: \(\{X \times 3 > 15\} Y := X * 3 \{Y > 15\}\)
  - Given that \(X = 6\), so \(X \times 3 = 6 \times 3 = 18\)
  - \(X \times 3 = 18 \implies X \times 3 > 15\)
Two incorrect assignment axiom forms

- $\{P\} V := E \{P[E/V]\}$
- $\{P\} V := E \{P[V/E]\}$
Precondition strengthening

If $\vdash \{P\}' \subseteq \{Q\}$ and $P \implies P'$, then we can write $\vdash \{P\} \subseteq \{Q\}$

- $\{X + 1 = n + 1\} X := X + 1 \{X = n + 1\}$ (assignment axiom)
- $\vdash X = n \implies X + 1 = n + 1$ (from arithmetic)
- $\{X = n\} X := X + 1 \{X = n + 1\}$ (precondition strengthening)
Postcondition weakening

If $\vdash \{P\} C \{Q'\}$, and $Q' \implies Q$, then we can write $\vdash \{P\} C \{Q\}$

- $\{R = X \land 0 = 0\} Q := 0 \{R = X \land Q = 0\}$ (assignment axiom)
- $R = X \land Q = 0 \implies R = X + (Y \times Q)$
- $\{R = X\} Q := 0 \{R = X + (Y \times Q)\}$ (postcondition weakening)
Conjunctions and Disjunctions

- If $\vdash \{ P_1 \} \mathcal{C} \{ Q_1 \}$ and $\vdash \{ P_2 \} \mathcal{C} \{ Q_2 \}$, then $\vdash \{ P_1 \land P_2 \} \mathcal{C} \{ Q_1 \land Q_2 \}$

- If $\vdash \{ P_1 \} \mathcal{C} \{ Q_1 \}$ and $\vdash \{ P_2 \} \mathcal{C} \{ Q_2 \}$, then $\vdash \{ P_1 \lor P_2 \} \mathcal{C} \{ Q_1 \lor Q_2 \}$
Sequencing Rule

- If \( \vdash \{ P \} C_1 \{ Q \} \) and \( \vdash \{ Q \} C_2 \{ R \} \), then \( \vdash \{ P \} C_1 ; C_2 \{ R \} \)

- You can combine the sequencing rule and the rules of consequence (i.e. precondition strengthening and postcondition weakening) to extend this to multiple statements.
The Conditional Rule

- If $\vdash \{ P \land S \} \ C_1 \ \{ Q \}$ and $\vdash \{ P \land \neg S \} \ C_2 \ \{ Q \}$, then
- $\vdash \{ P \} \ \text{IF} \ S \ \text{THEN} \ C_1 \ \text{ELSE} \ C_2 \ \{ Q \}$
The While Rule

- If $\{ P \land S \} \subseteq \{ P \}$ then
  - $\{ P \} \text{WHILE } S \text{ DO } C \text{ ENDDO } \{ P \land \neg S \}$
- Here, $P$ is the *inductive loop invariant*, recall:
  - It is true on entry into and exit out of the loop
  - It is true after every iteration of the loop
More rules

- FOR-rule
- Handling arrays
  - variant of assignment, due to McCarthy
Outline

Logics

A Logic for Proofs of Programs

Program Verification using Hoare Logic

Postscript
Example 1

\[ X = x \land Y = y \]

\begin{align*}
R & := X; \\
X & := Y; \\
Y & := R;
\end{align*}

\[ X = y \land Y = x \]
A verification condition is a mechanically generated proof goal from the program and program specifications.

For example, suppose \( \{P\} V := E \{Q\} \) exists in the program
- \( P \) is programmer-supplied precondition (or annotation)
- \( Q \) is programmer-supplied postcondition

The verification condition for this statement is
\[ P \implies Q[E/V] \]
Why the VC for assignment works

- From Hoare Logic, we have:
  - $\vdash \{ Q[E/V] \} V := E \{ Q \}$
- If we prove $P \implies Q[E/V]$, then by precondition strengthening, we have:
  - $\vdash \{ P \} V := E \{ Q \}$
- Which is what we had to prove.

What if we can’t prove $P \implies Q[E/V]$? Does that mean $\{ P \} C \{ Q \}$ does not hold?
Sufficiency and Incompleteness

- VCs are *sufficient*, but not necessary
  - There may be other ways to prove $\{P\} C \{Q\}$
- Mechanical provers cannot prove everything
  - Gödel’s Incompleteness Theorem
Verification conditions for our example

\[
\begin{align*}
\{X = x \land Y = y\} & \quad R := X; \\
X & := Y; \\
Y & := R; \quad \{X = y \land Y = x\}
\end{align*}
\]

- The verification conditions for a sequence ending in an assignment \(\{P\} C1; V := E \{Q\}\) are those generated by:
  - \(\{P\} C1 \{Q[E/V]\}\)
Verification conditions for our example: 2

\{X = x \land Y = y\} \quad R := X;
\quad X := Y; \quad \{X = y \land R = x\}

- Because \{X = y \land Y = x\}[R/Y], following from VC for sequences ending in an assignment.
Verification conditions for our example: 3

\{X = x \land Y = y\} \quad R := X; \quad \{Y = y \land R = x\}

- \(P = \{X = x \land Y = y\}\)
- \(Q = \{Y = y \land R = x\}\)
- Using VC for assignment:
  - \(Q[E/V] = \{Y = y \land R = x\}[X/R] = \{Y = y \land X = x\}\)
- Here, \(P \implies Q[E/V]\) trivially (identical)
Example 2

\[
k \geq 0
\]

\[
x := k; \\
c := 0;
\]

\[
\text{while}(x > 0) \{
  \quad x := x - 1; \\
  \quad c := c + 1;
\}
\]

\[
x = 0 \land c = k
\]
• The verification conditions for a While statement \( \{ P \} \text{WHILE } S \text{ DO } C \{ Q \} \) are
  
  \begin{itemize}
  \item \( P \implies R \) (where \( R \) is the loop invariant)
  \item \( R \land \neg S \implies Q \)
  \item recursively, all VCs from \( \{ R \land S \} \text{C} \{ R \} \)
  \end{itemize}

• The verification conditions for a sequence not ending in an assignment \( \{ P \} \text{C1;} \text{C2;} \text{C(n-1);} \text{Cn} \{ Q \} \), assuming \( \{ R \} \text{C(n)} \{ Q \} \) are those generated by:
  
  \begin{itemize}
  \item \( \{ R \} \text{Cn} \{ Q \} \)
  \item \( \{ P \} \text{C1;} \text{C2;} \text{C(n-1)} \{ R \} \)
  \end{itemize}
Verification Conditions for While loop and body

```text
while(x > 0) {
    x := x - 1;
    c := c + 1;
}
/* Q: x = 0 \&\& c = k */
```

- loop invariant: \( x + c = k \)
- (VC1) \( x + c = k \land \neg(x > 0) \implies x = 0 \land c = k \)
  - (from \( R \land \neg S \implies Q \))
- (VC2) \( P \implies x + c = k \) (from \( P \implies R \))
- (VC3) \( x + c = k \land x > 0 \implies x - 1 + c + 1 = k \) (VC from assignment)
  - Recursively from body:
  - \( \{x + c = k \land x > 0\} x := x - 1; \ c := c + 1 \{x + c = k\} \)
  - \( \{x + c = k \land x > 0\} x := x - 1 \{x + c + 1 = k\} \) (from sequence ending with assignment)
Let's assume $P = R$, so $P$ is $x + c = k$.

\[ VC0 \quad k \geq 0 \implies k = k \]

- $\{ k \geq 0 \} x := k; \ c := 0 \{ x + c = k \}$
- $\{ k \geq 0 \} x := k; \{ x + 0 = k \}$ (from sequence ending with assignment)
- $Q[E/V]$ is $k + 0 = k$
Verification Conditions

- (VC0) $k \geqslant 0 \iff k = k$
- (VC1) $x + c = k \land \neg (x > 0) \iff x = 0 \land c = k$
- (VC2) $x + c = k \iff x + c = k$
- (VC3) $x + c = k \land x > 0 \iff x + c = k$

- We need to show that $VC_0 \land VC_1 \land VC_2 \land VC_3$ is valid.
• Implication $P \implies Q$ can be read as, assume $P$ is true, show that $Q$ is valid.
  - $P \vdash Q$

• For proving with SAT solvers, if $P \implies Q$, check whether $P \land \neg Q$ is unsatisfiable
  - $P \implies Q$ can also be written as $\neg P \lor Q$
from z3 import *

s = Solver()
x, k, c = Ints('x k c')

vc0 = Not(Implies(k >= 0, k == k))
vc1 = Not(Implies(And(x + c == k, Not(x > 0)), And(x == 0, c == k)))
vc2 = Not(Implies(x + c == k, x + c == k))
vc3 = Not(Implies(And(x + c == k, x > 0), x + c == k))

s.add(And(And(And(vc0, vc1), vc2), vc3))

if s.check() == sat:
    print("SAT", s.model())
else:
    print("UNSAT")
from z3 import *

s = Solver()
x, k, c = Ints('x k c')

vc0 = And(k >= 0, Not(k == k))
vc1 = And(And(x + c == k, Not(x > 0)), Not(And(x == 0, c == k)))
vc2 = And(x + c == k, Not(x + c == k))
vc3 = And(And(x + c == k, x > 0), Not(x + c == k))

s.add(And(vc0, And(vc1, And(vc2, vc3))))

if s.check() == sat:
    print("SAT", s.model())
else:
    print("UNSAT")

Note: No Implies in above code.
Program Verification Procedure

- Generate specifications (aka annotations or assert statements)
- Generate verification conditions
  - Usually mechanical, e.g. Dafny or CBMC
- Prove verification conditions
  - By hand or
  - Automated Theorem Prover
More stuff

- Generating VCs for other statements in language
- Soundness?
- Completeness?
- Decidability?
- Pointers: Separation logic
Outline

Logics

A Logic for Proofs of Programs

Program Verification using Hoare Logic

Postscript
Sources, further reading and links

- Background Reading on Hoare Logic, by Mike Gordon
  - The reference for this lecture
- Textbooks
  - Software Foundations: Vol 1: Logical Foundations,
  - Software Foundations: Vol 2: Programming Language Foundations
  - Concrete Semantics