CSC2/455 Software Analysis and Improvement
Introduction to Hoare Logic

Sreepathi Pai

May 3, 2021

URCS

Outline

Outline

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 $\vee, \wedge, \implies, \iff$ 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

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

Valid Arguments

$$
\begin{array}{r}
P_1 \\
P_2 \\
\cdots \\
P_n \\
\hline
P_{n+1}
\end{array}
$$

- An argument is valid if and only if $P_1 \wedge P_2 \wedge \cdots \wedge P_n \implies P_{n+1}$ is a tautology
  - this means that $P_1 \wedge P_2 \wedge \cdots \wedge P_n \wedge P_{n+1}$ is true

Rules of Inference: Modus Ponens

$$\begin{array}{r} p \\ p \implies q \\ \hline q \end{array}$$

- $(p \wedge (p \implies q)) \implies q$ is a tautology
- Example:
  - p is "it is raining"
  - $p \implies q$ is "if it is raining, roads are wet"
  - q, so "roads are wet"

Rules of Inference: Modus Tollens

$$\begin{array}{r} p \implies q \\ \neg q \\ \hline \neg p \end{array}$$

- $((p \implies q) \wedge (\neg q)) \implies \neg p$ 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

$$\begin{array}{rl} p & \Longrightarrow q \\ & q \\ \hline & p \end{array}$$

- $((p \implies q) \wedge q) \implies p$ 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

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*

$$\text{Total correctness} = \text{Termination} + \text{Partial Correctness}$$

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.

Formal Proof

- (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
  - $\vdash \{P[E/V]\}\,\texttt{V := E}\,\{P\}$
- $P[E/V]$ 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: $\{X \times 3 > 15\}\,\texttt{Y := X * 3}\,\{Y > 15\}$
  - Given that $X = 6$, so $X \times 3 = 6 \times 3 = 18$
  - $X * 3 = 18 \implies X * 3 > 15$

Two incorrect assignment axiom forms

- $\{P\}\,\mathtt{V} := \mathtt{E}\,\{P[E/V]\}$
- $\{P\}\,\mathtt{V} := \mathtt{E}\,\{P[V/E]\}$

Precondition strengthening

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

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

Postcondition weakening

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

- $\{R = X \wedge 0 = 0\}\, \mathtt{Q} := 0\, \{R = X \wedge Q = 0\}$ (assignment axiom)
- $R = X \wedge Q = 0 \implies R = X + (Y \times Q)$
- $\{R = X\}\, \mathtt{Q} := 0\, \{R = X + (Y \times Q)\}$ (postcondition weakening)

Conjunctions and Disjunctions

- If $\vdash \{P_1\}\, \mathtt{C}\, \{Q_1\}$ and $\vdash \{P_2\}\, \mathtt{C}\, \{Q_2\}$, then $\vdash \{P_1 \wedge P_2\}\, \mathtt{C}\, \{Q_1 \wedge Q_2\}$
- If $\vdash \{P_1\}\, \mathtt{C}\, \{Q_1\}$ and $\vdash \{P_2\}\, \mathtt{C}\, \{Q_2\}$, then $\vdash \{P_1 \vee P_2\}\, \mathtt{C}\, \{Q_1 \vee Q_2\}$

Sequencing Rule

- If $\vdash \{P\}\, \texttt{C1}\, \{Q\}$ and $\vdash \{Q\}\, \texttt{C2}\, \{R\}$, then $\vdash \{P\}\, \texttt{C1; C2}\, \{R\}$
- 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 $\vdash \{P \wedge S\}\, \texttt{C1}\, \{Q\}$ and $\vdash \{P \wedge \neg S\}\, \texttt{C2}\, \{Q\}$, then
  - $\vdash \{P\}\, \texttt{IF S THEN C1 ELSE C2}\, \{Q\}$

The While Rule

- If $\{P \wedge S\}\, \mathtt{C}\, \{P\}$ then
  - $\vdash \{P\}\, \mathtt{WHILE\ S\ DO\ C\ ENDDO}\, \{P \wedge \neg S\}$
- 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

Example 1

$$X = x \wedge Y = y$$

```
R := X;
X := Y;
Y := R;
```

$$X = y \wedge Y = x$$

Generating Verification Conditions

- A verification condition is a mechanically generated proof goal from the program and program specifications.
- For example, suppose $\{P\}\,\texttt{V := E}\,\{Q\}$ 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:
  - $\vdash \{Q[E/V]\}\, \texttt{V := E}\, \{Q\}$
- If we prove $P \implies Q[E/V]$, then by precondition strengthening, we have:
  - $\vdash \{P\}\, \texttt{V := E}\, \{Q\}$
- 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

$$\{X = x \wedge Y = y\} \quad \mathsf{R} := \mathsf{X};$$
$$\mathsf{X} := \mathsf{Y};$$
$$\mathsf{Y} := \mathsf{R}; \quad \{X = y \wedge Y = x\}$$

- The verification conditions for a sequence ending in an assignment $\{P\}\,\texttt{C1; V := E}\,\{Q\}$ are those generated by:
  - $\{P\}\,\texttt{C1}\,\{Q[E/V]\}$

Verification conditions for our example: 2

$$\{X = x \wedge Y = y\} \quad \mathsf{R} := \mathsf{X};$$
$$\mathsf{X} := \mathsf{Y}; \quad \{X = y \wedge R = x\}$$

- Because $\{X = y \wedge Y = x\}[R/Y]$, following from VC for sequences ending in an assignment.

Verification conditions for our example: 3

$\{X = x \wedge Y = y\} \quad \mathsf{R} := \mathsf{X}; \quad \{Y = y \wedge R = x\}$

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

Example 2

$k >= 0$

```
x := k;
c := 0;

while(x > 0) {
   x := x - 1;
   c := c + 1;
}
```

$x = 0 \wedge c = k$

Verification conditions for While and Sequences

- The verification conditions for a While statement $\{P\}$ `WHILE S DO C` $\{Q\}$ are
  - $P \implies R$ (where R is the loop invariant)
  - $R \wedge \neg S \implies Q$
  - recursively, all VCs from $\{R \wedge S\}$ `C` $\{R\}$
- The verification conditions for a sequence not ending in an assignment $\{P\}$ `C1; C2; C(n-1); Cn` $\{Q\}$, assuming $\{R\}$ `C(n)` $\{Q\}$ are those generated by:
  - $\{R\}$ `Cn` $\{Q\}$
  - $\{P\}$ `C1; C2; C(n-1)` $\{R\}$

Verification Conditions for While loop and body

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

- loop invariant: $x + c = k$
- (VC1) $x + c = k \wedge \neg(x > 0) \implies x = 0 \wedge c = k$
  - (from $R \wedge \neg S \implies Q$)
- (VC2) $P \implies x + c = k$ (from $P \implies R$)
- (VC3) $x + c = k \wedge x > 0 \implies x - 1 + c + 1 = k$ (VC from assignment)
  - Recursively from body:
  - $\{x + c = k \wedge x > 0\}$ `x := x - 1; c := c + 1` $\{x + c = k\}$
  - $\{x + c = k \wedge x > 0\}$ `x := x - 1` $\{x + c + 1 = k\}$ (from sequence ending with assignment)

Verification Conditions for Initialization

```
/* k >= 0 */
x := k;
c := 0;
/* P */
```

- Let's assume $P = R$, so P is $x + c = k$
- (VC0) $k >= 0 \implies k = k$
  - $\{k >= 0\}$ `x := k; c := 0` $\{x + c = k\}$
  - $\{k >= 0\}$ `x := k;` $\{x + 0 = k\}$ (from sequence ending with assignment)
  - $Q[E/V]$ is $k + 0 = k$

Verification Conditions

- (VC0) $k >= 0 \implies k = k$
- (VC1) $x + c = k \wedge \neg(x > 0) \implies x = 0 \wedge c = k$
- (VC2) $x + c = k \implies x + c = k$
- (VC3) $x + c = k \wedge x > 0 \implies x + c = k$

- We need to show that $VC_0 \wedge VC_1 \wedge VC_2 \wedge VC_3$ is true.
- Are there values x, c, k that simultaneously make all true?

SMT to the rescue

```
from z3 import *

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

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


SAT [c = 0, k = 0, x = 0]
```

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

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