CSC2/455 Software Analysis and Improvement
Program Analysis III – Deductive Techniques

Sreepathi Pai

URCS

April 29, 2019

Outline

Outline

Proving Programs Correct

How do we:

- specify the behaviour of programs?
- prove that an implementation matches its specification?
- check that the proof is sound?

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. Therefore, P and Q are assertions, usually over program state.

Partial and Total Correctness

- Note that 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

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

Outline

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

- $P[E/V]$ is read as P with all instances of V replaced by E
  - P with E for V
- $\{X = 1\}[Y/X]$ leads to $\{Y = 1\}$
- Considering proving this:
  - $\{X = 1\}$ `Y := X` $\{Y = 1\}$
- We can do this using the *assignment axiom*
  - $\vdash \{P[E/V]\}$ `V := E` $\{P\}$

Two incorrect assignment axiom forms

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

Precondition strengthening

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

- $\{X + 1 = n + 1\}$ `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 $\vdash \{P\}C\{Q\}$

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

Conjunctions and Disjunctions

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

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 $\vdash \{P \wedge S\} C_1 \{Q\}$ and $\vdash \{P \wedge \neg S\} C_2 \{Q\}$, then
  - $\{P\}$ `IF` S `THEN` C_1 `ELSE` C_2 $\{Q\}$

The While Rule

- If $\{P \wedge S\} C \{P\}$ then
  - $\{P\}$ `WHILE` S `DO` C `ENDDO` $\{P \wedge \neg S\}$
- Here, P is called a `inductive loop invariant`
  - It is true on entry and exit into loop
  - It is true after every iteration of the loop

More rules

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

Outline

Example

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

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

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

Summary of steps

- Add assertions/specifications that must hold at points in the program
  - called annotations
- Generate a set of *verification conditions* (VCs) from the program + specification
- Prove the verification conditions
  - These imply the annotations are true

Generating VCs for assignment

- The verification condition for a statement $\{P\}V := E\{Q\}$ is:
  - $P \implies Q[E/V]$ (assignment verification condition)
- How does showing this is true prove $\vdash \{P\}V := E\{Q\}$?

Why the VC for assignment works

- From Hoare Logic, we have:
  - $\{Q[E/V]\}V := E\{Q\}$
- If we prove $P \implies Q[E/V]$, then by precondition strengthening, we have:
  - $\{P\}V := E\{Q\}$
- 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

More complicated example: Integer Division

Source material, page 45

Can machines really do this?

Dafny

Summary

- Annotations are inserted by programmer
- Verification conditions are generated by compiler/verifier
- Verification conditions are proved by theorem prover
  - Cannot always be automated

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 Dafny Project at Microsoft Research
  - Try it in your browser: dafny at rise4fun (work through the Dafny tutorial)
  - More reading (including 4-part video lectures)
- IEEE CSDL, Accessible Software Verification with Dafny
- Textbooks
  - Software Foundations: Vol 1: Logical Foundations,
  - Software Foundations: Vol 2: Programming Language Foundations