, 20

CS 245 - Logic and Computation **Fall 2016**

Lecture 19, 20: November 15 - 17, 2016

Lecturer: Kevin Lanctot *Notes By: Harsh Mistry*

19.1 Program Verification

Formal Verification is when you state a specification and prove a program satisfies the specification for all inputs. We formally specify and verify programs to :

- Reduce Bugs
- To test safety critical software or important components
- To document individual component.
- To adopt a better development and verification process .

19.1.1 Framework for Software Verification

The steps of formal verification:

1. Convert teh informal description R of requirements for an application domain into an "equivalent" formula ϕ_R of some symbolic logic.
2. Write a program P which is meant to realise ϕ_R in some given programming environment, and
3. Prove that the program P satisfies the formula ϕ_R

Note : Only part 3 is covered in CS245

19.1.2 Hoare Triples

Our assertions about programs will have the form

- (P) - Precondition
- C - Program or code
- (Q) - Postcondition

The meaning of the triple (P) C (Q) is : If program C is run starting in a state that satisfies P, then the resulting state after execution of C will satisfy Q.

An assertion (P) C (Q) is called a Hoare Triple.

19.1.2.1 Specification of a Program

A specification of a program C is a Hoare triple with C as the second component.

Note : Specification is not behaviour.

19.1.3 Partial Correctness

A triple (P) C (Q) is satisfied under partial correctness, denoted

$$\models_{par} (P)C(Q),$$

if and only if , for every state s that satisfies condition P, if execution of C starting from state s terminates then the resulting state s' satisfies condition Q

Note : Partial correctness **does not imply termination.**

19.1.4 Total correctness

A triple (P) C (Q) is satisfied under total correctness denoted

$$\models_{par} (P)C(Q)$$

if and only if, for every state S that satisfies condition P, execution of C starting from state S terminates, and the resulting state s' satisfies Q.

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

19.1.5 Logical variables

Sometimes the preconditions and postconditions require additional variables that do not appear in the program. These are called logical variables.

The logical variable allows the postcondition to be expressed in terms of the value of at the beginning of the code.

19.1.6 Proving Correctness

Our Goal: Prove total correctness

Our Approach: We usually prove it by proving partial correctness and termination separately.

- For partial correctness: we shall introduce sound inference rules.
- For total correctness: we shall use ad hoc reasoning, which suffices for our examples.
- In general, total correctness is undecidable. i.e. it cannot be decided in an automated fashion that always classifiies programs correctly.

19.1.7 Inference Rules

- Rule for Assignment

$$\frac{}{(Q[E/x]) \quad X = E \quad (Q)}$$

- Inference Rule of Precondition Strengthening

$$\frac{P \to P' \quad (P')C(Q)}{(P)C(Q)}$$

- Inference Rule of Postcondition Weakening

$$\frac{(P)C(Q') \quad Q' \to Q}{(P)C(Q)}$$

- Inference Rule of Composition

$$\frac{(P)C_1(Q), \quad (Q)C_2(R)}{(P)C_1; C_2(R)}$$

- Inference Rule for if-then-else :

$$\frac{(P \wedge B)C_1(Q) \quad (P \wedge \neg B)C_2(Q)}{(P) \ \mathit{if} \ \{B\} \ C_1 \ \mathit{else} \ C_2(Q)}$$

- Inference Rule for if-then (without else)

$$\frac{(P \wedge B)C(Q) \quad (P \wedge \neg B) \to Q}{(P) \ \mathit{if} \ \{B\} \ C \ (Q)}$$

19.1.8 The Steps of Creating a Proof

1. First **annotate the code** using the appropriate inference rules.
2. Then **move bottom-up in the proof** : adding an assertion before each assignment statement, based on the assertion after the assignment (as we have always done for assignments).
3. Finally prove any justifications labelled implied:
   - Annotations from (1) above containing implications
   - Adjacent assertions created in step (2).

Proofs here can use predicate logic, basic arithmetic, or other appropriate reasoning