CPSC320

Tutorials

Robert Pringle

October 29, 2007

- Method used to prove things regarding programs such as the correctness of the program.
- Floyd-Hoare logic statements have the form $\vdash [P]C[Q]$ where P is the precondition, C is the set of commands that make up the program and Q is the postcondition.

Partial and Complete Correctness

- Proofs can be to determine partial or complete correctness.
  - Partial correctness is represented by encasing the precondition and postconditions of a logic statement inside $\{\ \}$ giving us a statement of the form $\vdash \{P\}C\{Q\}$.
  - For $\vdash \{P\}C\{Q\}$ the postcondition only needs to hold if C terminates, which is not guaranteed.
  - Total correctness is represented by encasing the precondition and postconditions of a logic statement inside [ ] giving us a statement of the form $\vdash [P]C[Q]$.
  - For $\vdash [P]C[Q]$ the termination of C is guaranteed for the if the precondition P is satisfied.

Floyd-Hoare Logic Axioms and Rules

- There are a number of axioms and rules that have been previously proved and that can be used to build a correctness proof.
- These include the assignment axiom, precondition strengthening, postcondition weakening, specification conjunction and disjunction, sequencing rule, derived sequencing rule, block rule, derived block rule, conditional rule, while rule and for rule.

Floyd-Hoare Logic Proofs

- Usually all statements of your proof will be numbered so if a rule or axiom needs to be applied between multiple statements it will be obvious what statements you are referring to.
- As well as the number and the actual statement you need to state what axiom or rule was used with which previous statements to valid the current statement.
- A line usually divides the final statement of your proof or the statement you have proved from the intermediate statements/steps of the proof.

Floyd-Hoare Logic Proof Example

Lets prove the following floyd-hoare statement:

$$\vdash \{even(X)\}X := X + 1; X := X + 1\{even(X)\}$$

(1)	$\vdash even(X) \Rightarrow \neg even(X+1)$	Math
(2)	$\vdash \neg even(X) \Rightarrow even(X+1)$	Math
(3)	$\vdash \{\neg even(X+1)\}X := X + 1\{\neg even(X)\}$	Assignment Axiom
(4)	$\vdash \{even(X)\}X := X + 1\{\neg even(X)\}$	Precondition Strengthening 1,3
(5)	$\vdash \{even(X+1)\}X := X + 1\{even(X)\}$	Assignment Axiom
(6)	$\vdash \{\neg even(X)\}X := X + 1\{even(X)\}$	Precondition Strengthening 2,5
(7)	$\vdash \{even(X)\}X := X + 1; X := X + 1; \{even(X)\}$	Sequencing Rule with 4,6

Prove the following statements using Floyd-Hoare logic.

1. $\vdash \{TRUE\}$
   IF $\neg even(X)$ THEN $X := X + 1;$ ENDIF
   $\{(XModulus2) = 0\}$

2. $\vdash \{even(X) \wedge odd(Y)\}$
   $X := X * Y; Y := X + Y;$
   $\{odd(Y)\}$

3. $\vdash \{X = a \wedge Y = b \wedge N = 0\}$
   WHILE $X > 0$ DO
   BEGIN $X := X - 1; N := N + Y;$ END
   $\{N = b * a\}$

Domain Equations

- Used to show the elements and mappings for a particular variable or function domain.
- Combinations of elements within a given domain are often represented by the cross product, $\times$, as the resulting domain will contain such combinations.
- Mappings from on domain of elements to another is often represented by $\rightarrow$.
- Domains that contain elements of either one domain or another are usually represented by the union of the two domains $+$

Domain Equation Example

Consider a C++ array, A, that contains C++ structures of type S. Let the C++ integer domain be represented by I, the float domain by F and the char domain by C and S be defined as:

```
struct {
  float F1;
  char C1;
  union {
    int I1;
    char C2;
  }
}
```

The domain equation for this would then be defined as:
$A : I \rightarrow F \times C \times (I + C)$

Domain Equation Practice

For each of the following questions we are given the domain of integers, I, positive integers, I_p, strings, S, floats, F and booleans, B. Write the domain equations for following entities:

1. A hash table with string keys and integer values.
2. An n-dimensional array of strings.
3. A hash table with string or integer keys and n-dimensional arrays, hash tables, integers or strings for values.
4. The domain equation for the following structure:

```
struct {
  float F1;
  bool B1;
  char C1;
  union {
    int I1;
    char C2;
  }
}
```

Domain Equation Practice

For each of the following questions assume that the domain of integers, I, strings, S, floats, F and booleans, B, are already defined. Write the C++ structure corresponding to the following domain equations:

1. $I \times B \times C$
2. $F + B + I$
3. $F \times B \times I \times (I + C + B)$
4. $F + B + (C \times B \times I)$

Type Equivalence

- Type equivalence is concerned with whether two types can be considered the same or not.
- The two primary methods used to determine type equivalence are structural equivalance, name equivalence and declaration equivalence.
- In structural equivalence two types are considered the same if they share the same structure.
  - For user-defined types that utilize other non-simple types structural equivalence can be determined by replacing a typename with its structure though this becomes problematic for recursive types.
- In name equivalence two types are considered the same only if they share the same name.
  - How name equivalence is applied to anonymous types is not always clear and can be implementation dependant.
- Declaration equivalence is where types constructed from another types (subrange, derived classes, etc.) are considered equivalent to the base type.

Type Equivalence Practice

Given the C floating point type, float, determine if the following types could be considered equivalent to this simple type and if so in which way would it be considered equivalent.

1. float fltarr[10];
2. typedef FPN float;
3. struct FPN { float f; }
4. struct { float f; } FPN;
5. union { float f; double d; }
6. float f;

Type Equivalence Practice

Given the C structure **struct STR** { **double d; float f; char c;** } give an analog structure that is.

1. Only structurally equivalent.
2. Declaritevly equivalent.
3. Name equivalent.