19CSE205 Program Reasoning Program Verification

Dr.S.Padmavathi
CSE, Amrita School of Engineering
Coimbatore

Formal Reasoning

- Formal reasoning about program correctness using pre- and post conditions
- Syntax: *{I} P {O}*
- *I* and *O* are predicates
- *P* is a program
- Semantics
- If we start in a state where *I* is true and execute *P,* then *P* will terminate in a state where *O* is true

Valid Hoare Triple

- I: { x != 0 }
- *S :* y = x*x;
- *O:* { y > 0 }
- Valid Hoare triple :
  - If S is executed in a state where I is false,
  - O might be true or it might be false;
  - a valid Hoare triple doesn't have to promise anything either way.
- Invalid Hoare triple
  - if there can be a scenario where I is *true*, S is executed, and O is false afterwards.
- Validity Suits Implication rule

Invalid Hoare triple

- Example 1:
- I: { x > 0 } *1 > 0;* *I is satisfied*
- S: x = y; *x = -1* *(note that I says nothing about y)*
- O: { x > 0 }. *-1 < 0;* *O is not satisfied*
- Example 2:
- I: { x >= 0 } *x = 0;* *I is satisfied*
- S: y = 2*x; *y=0;*
- O: { y > x } *y = x;* *O is not satisfied*

If O is changed to y >= x; then O is satisfied. Hence Valid

Example: pre and post condition

- Formulate the precondition / post condition in the following
- {true } x := 5 { } { true } x := 5 { x=5 }
- { } x := x + 3 { x = y + 3 } { x = y } x := x + 3 { x = y + 3 }
- { } x := x * 2 + 3 { x > 1 } { x > -1 } x := x * 2 + 3 { x > 1 }
- { x=a } if (x < 0) then x := -x { } { x=a } if (x < 0) then x := -x { x=|a| }
- { false } x := 3 { } { false } x := 3 { x = 8 }
- { x < 0 } while (x!=0) x := x-1 { } { x < 0 } while (x!=0) x := x-1 {false}

“Strong” vs “Weak” conditions

A condition can be thought of as a set, i.e., the set of values that make the condition true. For example, {x >= 2} can be thought of as the set {2, 3, 4, ...} assuming x:int.

Stronger conditions yield smaller sets and weaker conditions yield larger sets. For example, we can say that {x >= 50} is a stronger condition than {x >= 20}.

The strongest condition is false, and this corresponds to the empty set. The weakest condition is true, and this corresponds to the universal set – in our example, the set of all numbers.

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Weakest Precondition

- **Given:**
- A program *S*: *y* := *x* * *x*
- A postcondition *R*: *y* >=4
- **Find:**
- The weakest precondition *Q*.
- **Solution:**
- *Q*: (*x* <=-2)V(x>= 2).
- The precondition *x* >= 2 would also guarantee that R is valid after execution.
- Even stronger preconditions like *x* >= 3, *x* = 3, etc. would be valid preconditions as well.
- However, the weakest precondition is *Q*: (*x* <=-2)V(x>= 2).

Strongest postcondition

- Check whether the following Hoare Triples are valid.
  {x = 5} x := x * 2 { true } Infinite elements
- {x = 5} x := x * 2 { x > 0 } {1,2,3,4,.....}
- {x = 5} x := x * 2 { x = 10 || x = 5 } {5,10}
- {x = 5} x := x * 2 { x = 10 } {5}
- All are true
- Identify the strongest post condition:
- x=10 is the *strongest postcondition*

Weakest Precondition

- *weakest precondition:*
  - the most *useful* because it allows us to invoke the program in the most general condition
- Check whether the following Hoare Triples are valid.
- {x = 5 && y = 10} z := x / y { z < 1 }   x={5}, y={10}
- {x < y && y > 0} z := x / y { z < 1 }
  x={..-2,-,1,0,1,2,3,..}, y={1,2,3,…}
- {y ≠ 0 && x / y < 1} z := x / y { z < 1 }
- All are true   x={..-2,-,1,0,2,3,..}, y={..-3,-2,-1,1,2,3,…}
- Identify the weakest pre condition:
- y ≠ 0 && x / y < 1 is the *weakest precondition*

Exercise

- Consider the following Hoare triples:
- A) { z = y + 1 } x := z * 2 { x = 4 }
- B) { y = 7 } x := y + 3 { x > 5 }
- C) { false } x := 2 / y { true }
- D) { y < 16 } x := y / 2 { x < 8 }
- Which of the Hoare triples above are valid?
- Considering the valid Hoare triples, for which ones can you write a stronger postcondition? (Leave the precondition unchanged, and ensure the resulting triple is still valid)
- Considering the valid Hoare triples, for which ones can you write a weaker precondition? (Leave the postcondition unchanged, and ensure the resulting triple is still valid)

Validity Checking

- Deductive method:
  - Forward reasoning: (strongest postcondition method)
    - Assumptions & axioms are logically combined by inference rules to reason towards goal
  - Backward reasoning: (weakest precondition method)
    - Inference rules are directly applied to goal generating new sub goals.
- Hoare Logic is **relational**:
  - For each O, there are many I such that {I}S{O}.
  - For each I, there are many O such that {I}S{O}.
- WP is **functional**:
  - For each O there is exactly one assertion wp(S,O).
- WP does respect Hoare Logic: {wp(S,O)} S{O} is true.

Program Verification

Objective: To prove that a program P is correct with respect to its contract which is stated as a pre-condition I and post-condition O.

The Weakest Precondition of a statement S w.r.t. a post-condition O is written as wp(S, O).

If the input condition for program P is I, then we want the following theorem to be true:

$$I \implies wp(P, O)$$

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Defining Weakest Preconditions

1. wp(x = expr, O).

2. wp(S1 ; S2, O).

3a. wp(if (B) S1 else S2, O).
3b. wp(if (B) S1, O).

4. wp(while B do S, O).

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Assignment Axiom

When S is an assigment statement, x = expr, the weakest precondition wp(x = expr, O) is defined as

$$\boxed{O\ [x \leftarrow expr]}$$

i.e., replace all occurrences of x in O by expr.

Example: If S is x = y * 5 and
O is {x >= 20}
then wp(S, O)

= {x >= 20} [x ← y * 5]

= {y * 5 >= 20}

= {y >= 4}

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Why "weakest" pre-condition?

Re-consider: S is $x = y * 5$ and O is $\{x >= 20\}$.

We derived: $wp(S, O) = \{x >= 20\} [x \leftarrow y * 5]$

$= \{y >= 4\}$

Given the above S and O, input conditions such as

$\{y = 4\}$ or

$\{y = 50\}$ or

$\{y >= 100\}$ ...

will all yield the output condition O.

However, the "weakest" (i.e., **least restrictive)** input condition is $\{y >= 4\}$

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Example: Assignment

- Compute the weakest precondition for the assignment statement a := 2 * (b - 1) given the postcondition (a > 0).
- {I}P {O}.
- {}a := 2 * (b - 1) {a > 0}.
- I= wp(S, O) = {a >0} [a ← 2 * (b - 1)]
- ={2 * (b - 1) > 0}
- ={2b - 2 > 0}
- ={2b > 2}
- {b > 1}

Sequencing

Given a statement sequence, S1 ; S2 ;

How should we define:

wp(S1 ; S2 ; , O) = wp(S1, wp(S2, O))

Weakest pre-conditions is a "backward flow" analysis, from output back to input.

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Sequencing (cont'd)

Given a sequence of statements, S1 ; S2 ;

wp(S1 ; S2 ; , O) = wp(S1, wp(S2, O))

Example:

If S is y = z – 4; x = y * 5; and
O is {x >= 20}

then wp(S, O) = wp(y = z – 4, $\boxed{\text{wp(x = y * 5, \{x >= 20\})}}$)

= wp(y = z – 4, {y >= 4})

= {z - 4 >= 4}

= $\boxed{\text{\{z >= 8\}}}$ Weakest Pre-condition

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Example: Sequencing

- Compute the weakest precondition for the following sequence of assignment statements, for the post condition given.
- { }a := 2 * b + 1; b := a – 3 {b < 0}
- { }a := 2 * b + 1;{ } b := a – 3 {b < 0}
- { }a := 2 * b + 1;{ } {a-3 < 0}
- { }a := 2 * b + 1;{ a<3} b := a – 3 {b < 0}
- { }a := 2 * b + 1; {a< 3}
- { } {2 * b + 1 < 3}
- { } { 2b < 2}
- {b < 1} a := 2 * b + 1; b := a – 3 {b < 0}

Exercise: Derive the WP

- Code Snippet 1:
  - a := 2*(b-1)-1
  - {a>0}
- Code Snippet 2:
  - a := a+2*b-1
  - {a>1}
- { I } x := 3 { x+y > 0 }
- { I } x := 3*y + z { x * y - z > 0 }
- Code Snippet 3:
  - a := 2*b+1
  - b := a-3
  - {b < 0}
- Code Snippet 4:
  - a := 3*(2*b+a)
  - b := 2*a-1
  - {b>5}

WP Derivation -Swap

- I: { }
- O: {x=n, y=m}
- P:{
-     x = x+y;
-     y = x-y;
-     x = x - y;
- }

{(x+y)-((x+y)-y) =n, ((x+y)-y)=m}

{y=n, (x+y)-y=m}

{y=n, x=m}

{x-(x-y) =n, (x-y)=m}

{y=n, x-y=m}

{x-y =n, y=m}

{x=n, y=m}

Swap WP in Alt-Ergo

Valid (syntax: I-> WP)

- goal g_1 :
- forall x,y,z,t:int.
- x=10 and y=3 ->
- ((x+y)-((x+y)-y) =3) and(((x+y)-y)=10)

- goal g_1 :
- forall x,y,m,n:int.
- x=m and y=n ->
- ((x+y)-((x+y)-y) =n) and(((x+y)-y)=m)

unknown

- goal g_1 :
- forall x,y,z,t:int.
- x=3 and y=10 ->
- ((x+y)-((x+y)-y) =3) and(((x+y)-y)=10)

- goal g_1 :
- forall x,y,m,n:int.
- x=n and y=m ->
- ((x+y)-((x+y)-y) =n) and(((x+y)-y)=m)

Longer Example

```
I: w = 5 && y = 7
O: w*w + x = y*y + z
P:{
    x = 6;
    z = 8;
    w = w*2 - 5;
    x = (x-1)*(x-1);
    y = y-2;
    z = (z+2)*(z+2);
    z = z - 75;
}
```

The example motivates need for program reasoning tool.

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

WP Derivation

- I: w = 5 && y = 7
- O: w*w + x = y*y + z
- P:{ { (w*2-5)*(w*2-5) + (6-1)*(6-1) = (y-2)*(y-2) + (8+2)*(8+2)-75}
-     x = 6; { (w*2-5)*(w*2-5) + (x-1)*(x-1) = (y-2)*(y-2) + (8+2)*(8+2)-75}
-     z = 8; { (w*2-5)*(w*2-5) + (x-1)*(x-1) = (y-2)*(y-2) + (z+2)*(z+2)-75}
-     w = w*2 - 5; { w*w + (x-1)*(x-1) = (y-2)*(y-2) + (z+2)*(z+2)-75}
-     x = (x-1)*(x-1); { w*w + x = (y-2)*(y-2) + (z+2)*(z+2)-75}
-     y = y-2; { w*w + x = y*y + (z+2)*(z+2)-75}
-     z = (z+2)*(z+2); { w*w + x = y*y + z-75}
-     z = z - 75; { w*w + x = y*y + z}
- }

Weakest Precondition method

- *WP methodology involves generating Verification Conditions* (VC).
  - Given {I}P{O} the over all VC to be proved is **I ⇒ wp(P,O)**
- These VCs are proved using theorem provers.
- VC generators:
  - an algorithm that takes a Hoare Triple and produces a set of first-order verification conditions such that the triple is derivable in Hoare logic if and only if all the conditions are valid.
- automatic theorem prover:
  - machine assistance to establish the validity of the verification conditions.
  - all such tools generically called as "proof tools" or "provers".
- Verification conditions are sometimes said to be *discharged* by a prover when successfully proved

Alt-Ergo

Frama-C: for given P,I,O internally generates the VCs which are verified by theorem provers

Alt-Ergo is one of the theorem provers underlying Frama-C. It can be used to check the validity of Verification Conditions (VCs) that arise in software verification.

It is based upon the concept of Satisfiability Modulo Theories (SMT). The theories are the domain axioms of datatypes.

Online Tool: https://alt-ergo.ocamlpro.com/try.html

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.

Longer Example

Pre-condition

Post-condition

```
@requires w = 5 && y = 7
@ensures  w*w + x = y*y + z
@program {
    x = 6;
    z = 8;
    w = w*2 - 5;
    x = (x-1)*(x-1);
    y = y-2;
    z = (z+2)*(z+2);
    z = z - 75;
}
```

The example motivates need for program reasoning tool.

Reference: Dr.Bharat Jayaraman, Unviversity of Buffalo, CSE449-459 Software verification course, Spring 2020.