19CSE205 – Program Reasoning

Jevitha KP

Lecture 16 –
 Hoare Triples, WP vs Hoare triples,
 Partial vs Total Correctness,
 WP for skip / abort / break / continue

Hoare Triples

- Three Components of Hoare Triples
  - Precondition (P)
  - Code fragment (S)
  - Postcondition (Q)
- The precondition is an ***assertion*** saying something of interest about the ***state before*** the code is executed.
- The postcondition is an ***assertion*** saying something of interest about the ***state after*** the code is executed
- A **state** is determined by the values given to the program variables
- **Assertions** about the state will be built out of variables, numbers, and basic arithmetic relations and combined with propositional logic operators

Hoare Triples

- The Hoare triple: {P} S {Q}
- Hoare triple {P} S {Q} is valid iff:

  For all states where P holds, executing S always produces a state where Q holds

  - "If P is true in the initial state before S, and S terminates, then Q will hold in the final state."

Hoare Triples

- Valid or invalid? - Assume all variables are integers without overflow
  - {x != 0} y = x*x; {y > 0} - Valid
  - {z != 1} y = z*z; {y != z} - Invalid (z=0)
  - {x > 0} y = 2*x; {y > x} - Valid (Invalid for precondition x>=0)
  - {true} if (x > 7) { y=4; } else { y=3; } {y < 5} - Valid
  - {true} x = y; z = x; {y=z} - Valid

Hoare Triples

- Valid or invalid?
  - {x != 0} y = x*x; {y > 0} - Valid
  - {z != 1} y = z*z; {y != z} - Valid
  - {x >= 0} y = 2*x; {y > x} - Invalid
  - {true} if (x > 7) { y=4; } else { y=3; } {y < 5} - Valid
  - {true} x = y; z = x; {y=z} - Valid

WP vs Hoare Logic

- Dijkstra's Weakest Precondition Calculus is another technique for proving properties of imperative programs.
- **Hoare Logic presents *logic* problems:**
  - Given a precondition P, a code fragment S and a postcondition Q, is {P} S {Q} true?
- **WP is about evaluating a *function*:**
  - Given a code fragment S and post condition Q, find the unique P which is the weakest precondition for S and Q.

WP Function

- If S is a code fragment and Q is an assertion about states, then the ***weakest precondition*** for S with respect to Q is an assertion, that is true for precisely those initial states from which:
  - S must terminate, and
  - Executing S must produce a state satisfying Q.
- That is, the weakest precondition P is a function of S and Q: **P=wp(S,Q)**
- (wp is sometimes called a ***predicate transformer***, and the Weakest Precondition Calculus is sometimes called ***Predicate Transformer Semantics.***)

WP Relationship with Hoare logic

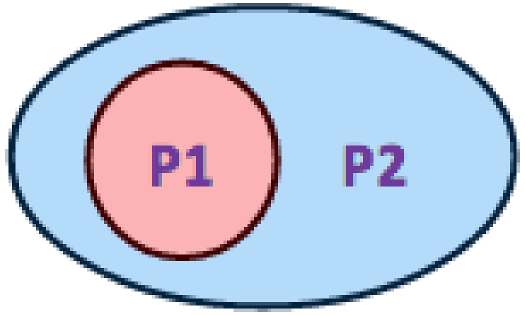
- Hoare Logic is relational:
  - For each Q, there are many P such that {P} S {Q} holds.
  - For each P, there are many Q such that {P} S {Q} holds.
- WP is functional:
  - For each Q, there is exactly one assertion **wp(S,Q)**.
- WP does respect Hoare Logic:
  - **{wp(S,Q)}** S **{Q}** is true

WP vs Hoare Logic

- Weakest Precondition WP:
  - For a statement S and a postcondition Q, a weakest precondition is a predicate WP such that for any precondition P , { P } S { Q } if and only if $P \Rightarrow WP$.
  - In other words, it is the "loosest" or least restrictive requirement needed to guarantee that Q holds after S.

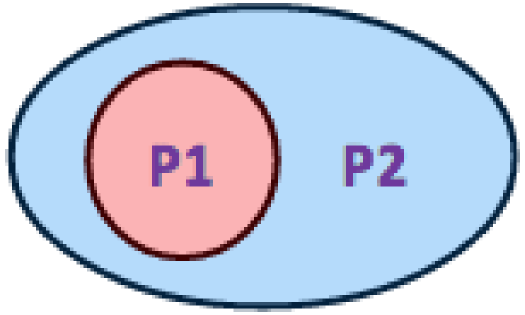
Stronger vs Weaker Assertions

- If P1 => P2 then:
  - P1 is stronger than P2
  - P2 is weaker than P1
- Whenever P1 holds, P2 is guaranteed to hold



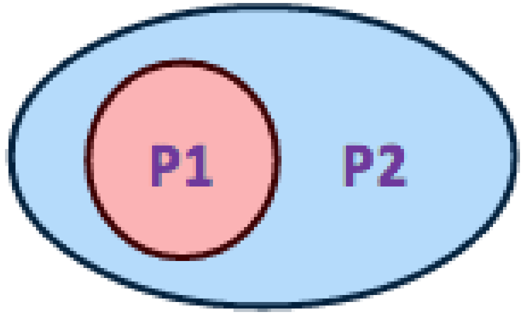
Stronger vs Weaker Assertions

- x > 0 => x >= 0?
- x = 1 => x > 0?
- x > 0 => true?
- x > 0 => x != 1?



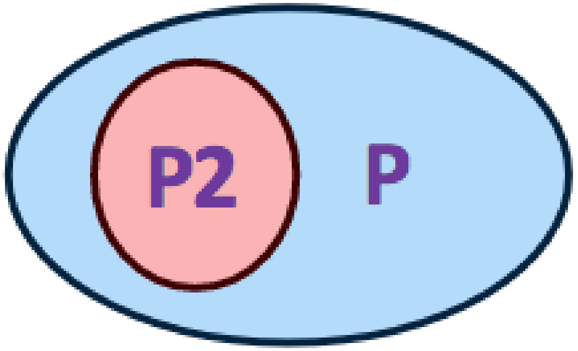
Stronger vs Weaker Assertions

- x > 0 => x >= 0 - True
- x = 1 => x > 0 - True
- x > 0 => true - True
- x > 0 => x != 1 - False



Weakest Precondition (WP)

- Given S and Q, find P such that {P} S {Q}.
- But which P?
  - { x > 0 } y = x*x { y > 0 }
  - { x != 0 } y = x*x { y > 0 }
- Weakest precondition P: {P} S {Q}
  - For all P2, if {P2} S {Q} then P2 => P.
- Most relaxed requirements on program state.



Partial Correctness vs Total Correctness

- **Partial Correctness**
  - Weak requirement where the **program satisfies the specification** even if it **does not terminate**
  - Hoare Logic is about partial correctness
  - (We do not go into termination proofs when doing WP for loops, we stop with partial correctness proof)

Partial Correctness vs Total Correctness

- **Total correctness**
  - Requires that the program terminates in order to satisfy a specification
  - WP is about total correctness
  - Total Correctness = Partial Correctness + Termination

Note on Termination

- Some loops should run forever Eg: a web server,
- But most of the loops we write should normally terminate.
- So, a full proof that a loop is correct involves showing that it terminates as well as that it computes the right result.
- Hoare logic only says that *if* a loop terminates then the postcondition holds.
- We have ignored termination to focus on the correctness issues.

Proving Termination – Loop Variant

- There are various ways to prove termination, but a common strategy is the following:
  - Map the state of the computation to a natural number somehow (only used "in the proof")
  - Prove that the natural number decreases on every iteration
  - Prove that the loop test is false by the time the natural number gets to 0, if not sooner

Proving Termination

- For many examples, the appropriate number is the size of the unprocessed input:
  - The quantity *n* - *k* in when we sum the numbers from 1 to *n*
  - The size of the part of the array that has not yet been examined / unprocessed array in finding max number in an array , etc
- In these cases, the number decreases by 1 or more (for binary search) on each iteration of the loop.
- Eventually the number reaches 0 and the loop must terminate at that point.

WP for Skip, Abort

- **Skip :** { ??} skip ; {O}

  WP(skip, O) = O

- **Abort (exit) :** {??} abort {O}

  WP(abort, O) = false

Example

- Find WP : If($x<y$) $x=y$ else skip end, O : $x>= y$

Example

- Find WP : If(x<y) x=y else skip end, O : x>= y

- WP(S1,O) = WP(x=y, x>=y)

= x>= y {x=y} = y >= y

- WP(S2,O) = WP (skip, x>=y) = x>= y

- B && wp (S1,O) || not(B) && wp(S2,O)

= (x<y) && (y>=y) || not(x<y) && (x>=y)

= (x<y) && True || not(x<y) && not(x<y)

= (x<y) || not(x<y)

= True

WP for Break, continue

- Break:

  @invariant I **while (B) { … ; break ; … }** O …

  - WP is the post-condition, O, of the while-loop.

- Continue :

  @invariant I **while (B) { … ; continue; … }** O …

  - WP is the loop invariant, I, of the while-loop.

Other WP facts

- WP(S, false) = false, for any program S.
- WP(S, true) = true, for any program S, if S contains no abort and all repetitions are assured of termination