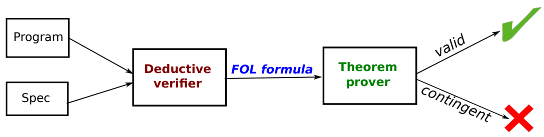
Introduction to Deductive Program Verification

Işıl Dillig

Hoare Logic I



- Example specs: safety (no crashes), absence of arithmetic overflow, complex behavioral property (e.g., "sorts an array")
- Verification condition: An SMT formula ϕ s.t. program is correct iff ϕ is valid

Hoare Logic

- Hoare logic forms the basis of all deductive verification techniques
- Named after Tony Hoare: inventor of quick sort, father of formal verification, 1980 Turing award winner

- Logic is also known as Floyd-Hoare logic: some ideas introduced by Robert Floyd in 1967 paper "Assigning Meaning to Programs"

Simple Imperative Programming Language

- To illustrate Hoare logic, we'll consider a small imperative programming language IMP
- In IMP, we distinguish three program constructs: expressions, conditionals, and statements
- Expression $E := Z \mid V \mid e_1 + e_2 \mid e_1 \times e_2$
- Conditional $C := \texttt{true} \mid \texttt{false} \mid e_1 = e_2 \mid e_1 \leq e_2$

$$\begin{array}{llll} \text{Statement } S := & V := E & & \text{(Assignment)} \\ & S_1; S_2 & & \text{(Composition)} \\ & \texttt{if } C \texttt{ then } S_1 \texttt{ else } S_2 & & \text{(If)} \\ & \texttt{while } C \texttt{ do } S & & \text{(While)} \end{array}$$

Partial Correctness Specification

- In Hoare logic, we specify partial correctness of programs using Hoare triples:

$$\{P\}\ S\ \{Q\}$$

- Here, S is a statement in programming language IMP
- P and Q are SMT formulas
- P is called precondition and Q is called post-condition

Meaning of Hoare Triples

- Meaning of Hoare triple $\{P\}S\{Q\}$:
  - If S is executed in state satisfying P
  - and if execution of S terminates
  - then the program state after S terminates satisfies Q
- Is $\{x = 0\}\ \texttt{x} := \texttt{x} + 1\ \{x = 1\}$ valid Hoare triple?
- What about $\{x = 0 \land y = 1\}\ \texttt{x} := \texttt{x} + 1\ \{x = 1 \land y = 2\}$?
- What about $\{x = 0\}\ \texttt{x} := \texttt{x} + 1\ \{x = 1 \lor y = 2\}$?
- What about $\{x = 0\}\ \texttt{while true do x} := 0 \{x = 1\}$?

Partial vs. Total Correctness

- The specification $\{P\}S\{Q\}$ called partial correctness spec. b/c doesn't require S to terminate
- There is also a stronger requirement called total correctness
- Total correctness specification written $[P]S[Q]$
- Meaning of $[P]S[Q]$:
  - If S is executed in state satisfying P
  - then the execution of S terminates
  - and program state after S terminates satisfies Q
- Is $[x = 0]$ `while true do x :=` $0[x = 1]$ valid?

Example Specifications

- What does $\{true\}S\{Q\}$ say?
- What about $\{P\}S\{true\}$?
- What about $[P]S[true]$?
- When does $\{true\}S\{false\}$ hold?
- When does $\{false\}S\{Q\}$ hold?
- We'll only focus on only partial correctness (safety)
- Total correctness = Partial correctness + termination

More Examples

Valid or invalid?

- $\{i = 0\}$ `while i<n do i++;` $\{i = n\}$
- $\{i = 0\}$ `while i<n do i++;` $\{i \geq n\}$
- $\{i = 0 \land j = 0\}$ `while i<n do i++; j+=i` $\{2j = n(n + 1)\}$
- How can we strengthen the precondition so it's valid?

Proving Partial Correctness

- Key problem: How to prove valid Hoare triples?
- If a Hoare triple is valid, written $\models \{P\}\ S\{Q\}$, we want a proof system to prove its validity
- Use notation $\vdash \{P\}S\{Q\}$ to indicate that we can prove validity of Hoare triple
- Hoare also gave a sound and (relatively-) complete proof system that allows semi-mechanizing correctness proofs
- Soundness: If $\vdash \{P\}S\{Q\}$, then $\models \{P\}S\{Q\}$
- Completeness: If $\models \{P\}S\{Q\}$, then $\vdash \{P\}S\{Q\}$

Inference Rules

- Proof rules in Hoare logic are written as inference rules:

$$\frac{\vdash \{P_1\}S_1\{Q_1\} \ldots \vdash \{P_n\}S_n\{Q_n\}}{\vdash \{P\}S\{Q\}}$$

- Says if Hoare triples $\{P_1\}S_1\{Q_1\}, \ldots, \{P_n\}S_n\{Q_n\}$ are provable in our proof system, then $\{P\}S\{Q\}$ is also provable.
- Not all rules have hypotheses: these correspond to bases cases in the proof
- Rules with hypotheses correspond to inductive cases in proof
- One inference rule for every statement in the IMP language

Understanding Proof Rule for Assignment

- Consider the assignment $x := y$ and post-condition $x > 2$
- What do we need before the assignment so that $x > 2$ holds afterwards?
- Consider $i := i + 1$ and post-condition $i > 10$
- What do we need to know before the assignment so that $i > 10$ holds afterwards?

Proof Rule for Assignment

$$\vdash \{Q[E/x]\}\ x := E\ \{Q\}$$

- To prove Q holds after assignment $x := E$, sufficient to show that Q with E substituted for x holds before the assignment.
- Using this rule, which of these are provable?
  - $\{y = 4\}\ x := 4\ \{y = x\}$
  - $\{x + 1 = n\}\ x := x + 1\ \{x = n\}$
  - $\{y = x\}\ y := 2\ \{y = x\}$
  - $\{z = 3\}\ y := x\ \{z = 3\}$

Exercise

- Your friend suggests the following proof rule for assignment:

$$\vdash \{(x = E) \rightarrow Q\}\ x := E\ \{Q\}$$

- Is the proposed proof rule correct?
-
-

Motivation for Precondition Strengthening

- Is the Hoare triple $\{z = 2\} y := x \{y = x\}$ valid?
- Is this Hoare triple provable using our assignment rule?
- Instantiating the assignment rule, we get:

$$\{y = x[x/y]x = xtrue\} y = x \{y = x\}$$

- But intuitively, if we can prove $y = x$ w/o any assumptions, we should also be able to prove it if we do make assumptions!

Proof Rule for Precondition Strengthening

$$\frac{\vdash \{P'\}S\{Q\} \qquad P \Rightarrow P'}{\vdash \{P\}S\{Q\}}$$

- Recall: $P \Rightarrow P'$ means the formula $P \rightarrow P'$ is valid
- Hence, need to use SMT solver every time we use precondition strengthening!

Example

- Using this rule and rule for assignment, we can now prove $\{z = 2\} y = x \{y = x\}$
- Proof:

$$\cfrac{\cfrac{\vdash \{y = x[x/y]\} y = x \{y = x\}}{\vdash \{true\} y := x \{y = x\}} \qquad z = 2 \Rightarrow true}{\vdash \{z = 2\} y := x \{y = x\}}$$

Proof Rule for Post-Condition Weakening

- We also need a dual rule for post-conditions called post-condition weakening:

$$\frac{\vdash \{P\}S\{Q'\} \qquad Q' \Rightarrow Q}{\vdash \{P\}S\{Q\}}$$

- If we can prove post-condition Q', we can always relax it to something weaker
- Again, need to use SMT solver when applying post-condition weakening

Post-condition Weakening Examples

- Suppose we can prove $\{true\}S\{x = y \wedge z = 2\}$.
- Using post-condition weakening, which of these can we prove?
  - $\{true\}S\{x = y\}$
  - $\{true\}S\{z = 2\}$
  - $\{true\}S\{z > 0\}$
  - $\{true\}S\{\forall y.\ x = y\}$
  - $\{true\}S\{\exists y.\ x = y\}$

Proof Rule for Composition

$$\frac{\vdash \{P\}S_1\{Q\} \quad \vdash \{Q\}S_2\{R\}}{\vdash \{P\}S_1; S_2\{R\}}$$

- Using this rule, let's prove validity of Hoare triple:

$$\{true\}\ x = 2; y = x\ \{y = 2 \wedge x = 2\}$$

- What is appropriate Q?

$$\frac{\dfrac{\{x = 2[2/x]\}x = 2\{x = 2\}}{\{true\}\ x = 2\ \{x = 2\}} \quad \dfrac{\{x = 2 \wedge y = 2[x/y]\}\ y = x\ \{x = 2 \wedge y = 2\}}{\{x = 2\}\ y = x\ \{x = 2 \wedge y = 2\}}}{\vdash \{true\}\ x = 2; y = x\ \{y = 2 \wedge x = 2\}}$$

Proof Rule for If Statements

$$\frac{\begin{array}{l}\vdash \{P \wedge C\}\ S_1\ \{Q\} \\ \vdash \{P \wedge \neg C\}\ S_2\ \{Q\}\end{array}}{\vdash \{P\}\ \texttt{if}\ C\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{Q\}}$$

- Suppose we know P holds before `if` statement and want to show Q holds afterwards.
- At beginning of then branch, what facts do we know?
- Thus, in the then branch, we want to show $\{P \wedge C\}S_1\{Q\}$
- At beginning of else branch, what facts do we know?
- What do we need to show in else branch?

Example

Prove the correctness of this Hoare triple:

$$\{true\}\ \texttt{if x > 0 then y := x else y := -x}\ \{y \geq 0\}$$

Exercise

- Your friend suggests the following proof rule:

$$\frac{\begin{array}{c}\{P \wedge C\}\ S_1; S_3\ \{Q\} \\ \{P \wedge \neg C\}\ S_2; S_3\ \{Q\}\end{array}}{\{P\}\ (\texttt{if}\ C\ \texttt{then}\ S_1\ \texttt{else}\ S_2);\ S_3\ \{Q\}}$$

- Is this proof rule correct? If so, prove your answer. Otherwise, give a counterexample.

Exercise, cont

- Yes, this rule can be derived from existing rules.
- From premises, we know (1) $\{P \wedge C\}\ S_1; S_3\ \{Q\}$ and (2) $\{P \wedge \neg C\}\ S_2; S_3\ \{Q\}$
- Let Q' be the weakest precondition we need for Q to hold after executing S_3, i.e., (3) $\{Q'\}S_3\{Q\}$
- Using premises, this means $\{P \wedge C\}S_1\{Q'\}$ and $\{P \wedge \neg C\}S_2\{Q'\}$
- From these and existing if rule, we can derive:

$$\{P\}\ (\texttt{if}\ C\ \texttt{then}\ S_1\ \texttt{else}\ S_2)\ \{Q'\}$$

- Conclusion follows from these and (3) using Seq

Proof Rule for While and Loop Invariants

- Last proof rule of Hoare logic is that for `while` loops.
- But to understand proof rule for while, we first need concept of a loop invariant
- A loop invariant I has following properties:
  1. I holds initially before the loop
  2. I holds after each iteration of the loop

Examples

- Consider the following code

  $\texttt{i} := 0;\ \texttt{j} := 0;\ \texttt{n} := 10;\ \texttt{while i} < \texttt{n do i} := \texttt{i} + 1;\ \texttt{j} := \texttt{i} + \texttt{j}$

- Which of the following are loop invariants?
  - $i \leq n$
  - $i < n$
  - $j \geq 0$
- Suppose I is a loop invariant. Does I also hold after loop terminates?
-

Proof Rule for While

- Consider the statement $\texttt{while } C \texttt{ do } S$
- Suppose I is a loop invariant for this loop. What is guaranteed to hold after loop terminates? $I \wedge \neg C$
- Putting all this together, proof rule for while is:

$$\frac{\vdash \{P \wedge C\} S \{P\}}{\vdash \{P\} \texttt{while } C \texttt{ do } S \{P \wedge \neg C\}}$$

- This rule simply says "If P is a loop invariant, then $P \wedge \neg C$ must hold after loop terminates"
- Based on this rule, why is P a loop invariant?
-

Example

- Consider the statement $S = \texttt{while } x < n \texttt{ do } x = x + 1$
- Let's prove validity of $\{x \leq n\} S \{x \geq n\}$
- What is appropriate loop invariant?
- First, let's prove $x \leq n$ is loop invariant. What do we need to show?
- What proof rules do we need to use to show this?

$$\frac{\vdash \{x \leq n[x+1/x]\} x = x + 1 \{x \leq n\} \vdash \{x + 1 \leq n\} x = x + 1 \{x}{\vdash \{x \leq n \wedge x < n\} x = x + 1 \{x \leq}$$

Example, cont

- Ok, we've shown $x \leq n$ is loop invariant, now let's instantiate proof rule for while with this loop invariant:

$$\frac{\vdash \{x \leq n \wedge x < n\} S' \{x \leq n\}}{\vdash \{x \leq n\} \texttt{while } x < n \texttt{ do } S' \{x \leq n \wedge \neg(x < n)\}}$$

- Recall: We wanted to prove the Hoare triple $\{x \leq n\} S \{x \geq n\}$
- In addition to proof rule for while, what other rule do we need?

Example, cont.

The full proof:

$$\cfrac{\cfrac{\cfrac{\vdash \{x+1 \leq n\} x = x + 1 \{x \leq n\} \quad x \leq n \wedge x < n \Rightarrow x + 1 \leq n}{\vdash \{x \leq n \wedge x < n\} x = x + 1 \{x \leq n\}}}{\vdash \{x \leq n\} S \{x \leq n \wedge \neg(x < n)\}} \quad x \leq n \wedge \neg(x < n) \Rightarrow x \geq n}{\{x \leq n\} S \{x \geq n\}}$$

Invariant vs. Inductive Invariant

- Suppose I is a loop invariant for `while C do S`.
- Does it always satisfy $\{I \wedge C\}S\{I\}$?
- Counterexample: Consider $I = j \geq 1$ and the code:

  $\texttt{i} := 1;\ \texttt{j} := 1;\ \texttt{while i} < \texttt{n do } \{\texttt{j} := \texttt{j} + \texttt{i};\ \texttt{i} := \texttt{i} + 1\}$

- But strengthened invariant $j \geq 1 \wedge i \geq 1$ does satisfy it
- Such invariants are called inductive invariants, and they are the only invariants that we can prove
- Key challenge in verification is finding inductive loop invariants

Exercise

Find inductive loop invariant to prove the following Hoare triple:

$$\{i = 0 \wedge j = 0 \wedge n = 5\}$$
$$\texttt{while i} < \texttt{n do i} := \texttt{i} + 1;\ \texttt{j} := \texttt{j} + \texttt{i}$$
$$\{j = 15\}$$

- Inductive loop invariant I:
- Weakest precondition P w.r.t loop body:

  $$2j = i(i+1) \wedge i + 1 \leq n \wedge n = 5$$

- Since $I \wedge C \Rightarrow P$, I is inductive.

Another Exercise

- Suppose we add a for loop construct to IMP:

  $$\texttt{for v} := \texttt{e}_1 \texttt{ until e}_2 \texttt{ do S}$$

- Initializes v to e_1, increments v by 1 in each iteration and terminates when $v > e_2$
- Write a proof rule for this for loop construct
- We can de-sugar into while loop:

  $$\texttt{v} := \texttt{e}_1; \texttt{while v} \leq \texttt{e}_2 \texttt{ do } \{\texttt{S};\ \texttt{v} := \texttt{v} + 1\}$$

Exercise, cont.

$$\texttt{v} := \texttt{e}_1; \texttt{while v} \leq \texttt{e}_2 \texttt{ do } \{\texttt{S};\ \texttt{v} := \texttt{v} + 1\}$$

- Suppose I is the inductive invariant of while loop
- First, I must hold at the beginning:

  $$\{P\}\ \texttt{v} := \texttt{e}_1\ \{I\}$$

- Next, I must be inductive:

  $$\{I \wedge v \leq e_2\}\ \texttt{S}; \texttt{v} := \texttt{v} + 1\ \{I\}$$

Exercise, cont.

- Putting all this together, we get the following proof rule:

$$\frac{\{P\}\ \texttt{v} := \texttt{e}_1\ \{I\} \quad \{I \wedge v \leq e_2\}\ \texttt{S}; \texttt{v} := \texttt{v} + 1\ \{I\}}{\{P\}\ \texttt{for v} := \texttt{e}_1 \texttt{ until e}_2 \texttt{ do S}\ \{I \wedge v > e_2\}}$$

Arrays

- Let's add arrays to our IMP language:

  $$\texttt{v}[\texttt{e}_1] := \texttt{e}_2$$

- What is the proof rule for this statement?
- Idea 1: Treat array write just like assignment:

  $$\frac{}{\{Q[e_2/v[e_1]]\}\ \texttt{v}[\texttt{e}_1] := \texttt{e}_2\ \{Q\}}$$

- Is this rule correct?

Counterexample

- No, counterexample:

$$\{i = 1\}\ \mathtt{v}[\mathtt{i}] := 3; \mathtt{v}[1] := 2\ \{v[i] = 3\}$$

- What is the value of `v[i]` after this code?
- But using previous "proof rule", we can "prove" this Hoare triple
- Clearly, this rule is unsound

Correct Proof Rule for Arrays

- The correct proof rule:

$$\frac{}{\{Q[v\langle e_1 \triangleleft e_2\rangle/v]\}\ \mathtt{v}[\mathtt{e}_1] := \mathtt{e}_2\ \{Q\}}$$

- Effectively assigns $\mathtt{v}$ to a new array that is the same as $\mathtt{v}$ except at index e_1
- We now require theory of arrays

Array Example

- Consider again this example:

$$\{i = 1\}\ \mathtt{v}[\mathtt{i}] := 3; \mathtt{v}[1] := 2\ \{v[i] = 3\}$$

- Applying the array write rule, we obtain:

$$\{v\langle 1 \triangleleft 2\rangle[i] = 3\}\ \mathtt{v}[1] := 2\ \{v[i] = 3\}$$

- Use composition and apply array rule to first statement:

$$\{(v\langle i \triangleleft 3\rangle)\langle 1 \triangleleft 2\rangle[i] = 3\}\ \mathtt{v}[\mathtt{i}] := 3\ \{v\langle 1 \triangleleft 2\rangle[i] = 3\}$$

- But the following implication is not valid:

$$i = 1 \Rightarrow (v\langle i \triangleleft 3\rangle)\langle 1 \triangleleft 2\rangle[i] = 3$$

Example with Arrays and Loops

- Consider the following code snippet:

$$\mathtt{while}\ \mathtt{i} < \mathtt{n}\ \mathtt{do}\ \{\mathtt{a}[\mathtt{i}] := 0; \mathtt{i} := \mathtt{i} + 1;\}$$

- Suppose the precondition is $i = 0 \land n > 0$ and the postcondition is:

$$\forall j.\ 0 \leq j < n \rightarrow a[j] = 0$$

- Find an inductive loop invariant and show the correctness proof
- Inductive invariant:

Summary of Proof Rules

1. $\vdash \{Q[E/x]\}\ x = E\ \{Q\}$ (Assignment)
2. $\dfrac{\vdash \{P'\}S\{Q\} \quad P \Rightarrow P'}{\vdash \{P\}S\{Q\}}$ (Strengthen P)
3. $\dfrac{\vdash \{P\}S\{Q'\} \quad Q' \Rightarrow Q}{\vdash \{P\}S\{Q\}}$ (Weaken Q)
4. $\dfrac{\vdash \{P\}C_1\{Q\} \quad \vdash \{Q\}C_2\{R\}}{\vdash \{P\}C_1; C_2\{R\}}$ (Composition)
5. $\dfrac{\begin{array}{l}\vdash \{P \land C\}\ S_1\ \{Q\} \\ \vdash \{P \land \neg C\}\ S_2\ \{Q\}\end{array}}{\vdash \{P\}\ \mathtt{if}\ C\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2\ \{Q\}}$ (If)
6. $\dfrac{\vdash \{P \land C\}S\{P\}}{\vdash \{P\}\mathtt{while}\ C\ \mathtt{do}\ S\{P \land \neg C\}}$ (While)

Meta-theory: Soundness of Proof Rules

- It can be show that the proof rules for Hoare logic are sound:

$$\text{If } \vdash \{P\}S\{Q\}, \text{ then } \models \{P\}S\{Q\}$$

- That is, if a Hoare triple $\{P\}S\{Q\}$ is provable using the proof rules, then $\{P\}S\{Q\}$ is indeed valid
- Completeness of proof rules means that if $\{P\}S\{Q\}$ is a valid Hoare triple, then it can be proven using our proof rules, i.e.,

$$\text{If } \models \{P\}S\{Q\}, \text{ then } \vdash \{P\}S\{Q\}$$

- Unfortunately, completeness does not hold!

Meta-theory: Relative Completeness

- Recall: Rules for precondition strengthening and postcondition weakening require checking $A \Rightarrow B$
- In general, these formulas belong to Peano arithmetic
- Since PA is incomplete, there are implications that are valid but cannot be proven
- However, Hoare's proof rules still have important goodness guarantee: relative completeness
- If we have an oracle for deciding whether an implication $A \Rightarrow B$ holds, then any valid Hoare triple can be proven using our proof rules

Automating Reasoning in Hoare Logic

- Manually proving correctness is tedious, so we'd like to automate the tedious parts of program verification
- Idea: Assume an oracle gives loop invariants, but automate the rest of the reasoning
- This oracle can either be a human or a static analysis tool (e.g., abstract interpretation)

Basic Idea Behind Program Verification

- Automating Hoare logic is based on generating verification conditions (VC)
- A verification condition is a formula ϕ such that program is correct iff ϕ is valid
- Deductive verification has two components:
  1. Generate VC's from source code
  2. Use theorem prover to check validity of formulas from step 1

Generating VCs: Forwards vs. Backwards

- Two ways to generate verification conditions: forwards or backwards
- A forwards analysis starts from precondition and generates formulas to prove postcondition
- Forwards technique computes strongest postconditions (sp)
- In contrast, backwards analysis starts from postcondition and tries to prove precondition
- Backwards technique computes weakest preconditions (wp)
- We'll use the backwards method

Weakest Preconditions

- Idea: Suppose we want to verify Hoare triple $\{P\}S\{Q\}$
- We'll start with Q and going backwards, compute formula $wp(S, Q)$ called weakest precondition of Q w.r.t. to S
- $wp(S, Q)$ has the property that it is the weakest condition that guarantees Q will hold after S in any execution
- Thus, Hoare triple $\{P\}S\{Q\}$ is valid iff:

$$P \Rightarrow wp(S, Q)$$

Defining Weakest Preconditions

- Weakest preconditions are defined inductively and follow Hoare's proof rules
- $wp(x := E, Q) = Q[E/x]$
- $wp(s_1; s_2, Q) = wp(s_1, wp(s_2, Q))$
- $wp(\texttt{if } C \texttt{ then } s_1 \texttt{ else } s_2, Q) =$
  $C \rightarrow wp(s_1, Q) \wedge \neg C \rightarrow wp(s_2, Q)$
- This says "If C holds, wp of then branch must hold; otherwise, wp of else branch must hold"

Example

- Consider the following code S:

$$\texttt{x} := \texttt{y}+1;\ \texttt{if x} > 0 \texttt{ then z} := 1 \texttt{ else z} := -1$$

- What is $\text{wp}(S,\ z > 0)$?
- What is $\text{wp}(S,\ z \leq 0)$?
- Can we prove post-condition $z = 1$ if precondition is $y \geq -1$?
- What if precondition is $y > -1$?

Weakest Preconditions for Loops

- Unfortunately, we can't compute weakest preconditions for loops exactly...
- Idea: approximate it using $awp(S, Q)$
- $awp(S, Q)$ may be stronger than $wp(S, Q)$ but not weaker
- To verify $\{P\}S\{Q\}$, show $P \Rightarrow awp(S, Q)$
- Hope is that $awp(S, Q)$ is weak enough to be implied by P although it may not be the weakest

Approximate Weakest Preconditionsr loops, we will rely on loop invariants provided by oracle (human or static ana

- For all statements except for while loops, computation of $awp(S, Q)$ same as $wp(S, Q)$
- To compute, $awp(S, Q)$ for loops, we will rely on loop invariants provided by oracle (human or static analysis)
- Assume all loops are annotated with invariants $\texttt{while } C \texttt{ do } [I]\ S$
- Now, we'll just define $awp(\texttt{while } C \texttt{ do } [I]\ S, Q) \equiv I$
- Why is this sound?

Verification with Approximate Weakest Preconditions

- If $P \Rightarrow awp(S, Q)$, does this mean $\{P\}S\{Q\}$ is valid?
- No, two problems with $awp(\texttt{while } C \texttt{ do } [I]\ S, Q)$
  1. We haven't checked I is an actual loop invariant
  2. We also haven't made sure $I \land \neg C$ is sufficient to establish Q!
- For each statement S, generate verification condition $VC(S, Q)$ that encodes additional conditions to prove

Generating Verification Conditions

- Most interesting VC generation rule is for loops:

$$VC(\texttt{while } C \texttt{ do } [I]\ S, Q) = ?$$

- To ensure Q is satisfied after loop, what condition must hold? $I \land \neg C \Rightarrow Q$
- Assuming I holds initially, need to check I is loop invariant
- i.e., need to prove $\{I \land C\}S\{I\}$
- How can we prove this? check validity of $I \land C \Rightarrow awp(S, I)\ \land VC(S, I)$

Verification Condition for Loops

- To summarize, to show I is preserved in loop, need:

$$I \land C \Rightarrow awp(S, I)\ \land VC(S, I)$$

- To show I is strong enough to establish Q, need:

$$I \land \neg C \Rightarrow Q$$

- Putting this together, verification condition for a while loop $S' = \texttt{while } C \texttt{ do } \{I\}\ S$ is:

$$VC(S', Q) = (I \land C \Rightarrow awp(S, I) \land VC(S, I)) \land (I \land \neg C \Rightarrow Q)$$

Verification Condition for Other Statements

- We also need rules to generate VC's for other statements because there might be loops nested in them
- $VC(x := E, Q) = true$
- $VC(s_1; s_2, Q) = VC(s_2, Q) \land VC(s_1, awp(s_2, Q))$
- $VC(\texttt{if } C \texttt{ then } s_1 \texttt{ else } s_2, Q) = VC(s_1, Q) \land VC(s_2, Q)$

Verification of Hoare Triple

- Thus, to show validity of $\{P\}S\{Q\}$, need to do following:
  1. Compute $awp(S, Q)$
  2. Compute $VC(S, Q)$
- Theorem: $\{P\}S\{Q\}$ is valid if following formula is valid:

$$VC(S, Q) \ \land \ P \rightarrow awp(S, Q) \quad (*)$$

- Thus, if we can prove of validity of $(*)$, we have shown that program obeys specification

Discussion

Theorem: $\{P\}S\{Q\}$ is valid if following formula is valid:

$$VC(S, Q) \ \land \ P \rightarrow awp(S, Q) \quad (*)$$

- Question: If $\{P\}S\{Q\}$ is valid, is $(*)$ valid?
- No, for two reasons:
  1. Loop invariant might not be strong enough
  2. Loop invariant might be bogus
- Thus, even if program obeys specification, might not be able to prove it b/c loop invariants we use are not strong enough

Example

- Consider the following code:

```
i := 1; sum := 0;
while i ≤ n do [sum ≥ 0] {
    j := 1;
    while j ≤ i do [sum ≥ 0 ∧ j ≥ 0]
       sum := sum + j; j := j + 1
    i := i + 1
}
```

- Show the VC's generated for this program for post-condition $sum \geq 0$ – can it be verified?
- What is the post-condition we need to show for inner loop? $sum \geq 0$

Example, cont.

- Generate VC's for inner loop:

$$(1) \quad (sum \geq 0 \land j \geq 0 \land j > i) \Rightarrow sum \geq 0$$
$$(2) \quad (j \leq i \land sum \geq 0 \land j \geq 0) \Rightarrow (sum + j \geq 0 \land j + 1 \geq 0))$$

- Now, generate VC's for outer loop:

$$(3) \quad (i \leq n \land sum \geq 0) \Rightarrow (sum \geq 0 \land 1 \geq 0)$$
$$(4) \quad (i > n \land sum \geq 0) \Rightarrow sum \geq 0$$

- Finally, compute awp for outer loop: $(5)\ 0 \geq 0$
- Feed the formula $(1) \land (2) \land (3) \land (4) \land (5)$ to SMT solver
- It's valid; hence program is verified!

Example: Variant

- Suppose annotated invariant for inner loop was $sum \geq 0$ instead of $sum \geq 0 \land j \geq 0$
- Could the program be verified then? no, because loop invariant not strong enough
- While VC generation handles many tedious aspects of the proof, user must still come up with loop invariants (more on this in next few lectures)