Hoare Logic & Weakest Preconditions

Klaus v. Gleissenthall

Logic:

- Propositional
- First order
- Theories

→

Programs to Logic:

- Hoare Logic
- VCGen

→

Automation

- Horn clauses
- Predicate Abstraction

→

Security

- Information Flow Control
- Side-Channels

Precondition Strengthening

- Is the following Hoare triple valid?

$$\{z = 2\}\ y := x\ \{y = x\}$$

- Can we prove it with the assignment rule?

- Intuitively, we should be able to prove it <u>without</u> any assumptions; we should also be able to prove it <u>if we do</u> have assumptions!

Precondition Strengthening

- We write $P \Rightarrow P'$ to mean that formula $P \rightarrow P'$ is valid, i.e., $\vDash P \rightarrow P'$

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

- To check $P \Rightarrow P'$, we need to call the SMT solver
- Let's now prove $\{z = 2\}\ y := x\ \{y = x\}$

Postcondition Weakening

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

$$\frac{\vdash \{P\}\ s\ \{Q'\} \quad Q' \Rightarrow Q}{\vdash \{P\}\ s\ \{Q\}}$$

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

Postcondition Weakening

Quiz:

- Suppose we can prove $\{\top\}$ s $\{x = y \land z = 2\}$
- Using post-condition weakening, which of these can we prove?
  - $\{\top\}$ s $\{x = y\}$
  - $\{\top\}$ s $\{z = 2\}$
  - $\{\top\}$ s $\{z > 0\}$
  - $\{\top\}$ s $\{x > 1\}$
  - $\{\top\}$ s $\{\exists y.\ x = y\}$

Composition

$$\frac{\vdash \{P\}\ s_1\ \{Q\} \qquad \vdash \{Q\}\ s_2\ \{R\}}{\vdash \{P\}\ s_1;\ s_2\ \{R\}}$$

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

$$\{\top\}\ x := 2;\ y := x\ \ \{y = 2 \land x = 2\}$$

If Statements

$$\frac{\vdash \{P \land b\}\ s_1\ \{Q\} \qquad \{P \land \neg b\}\ s_2\ \{Q\}}{\vdash \{P\}\ \text{if } b \text{ then } s_1 \text{ else } s_2\ \{Q\}}$$

- Suppose P holds before the if-statement
- When executing the then-branch, what do we know?
- What about the else-branch?

If Statements

Example: Prove the correctness of this Hoare triple

$\{\top\}$ if x > 0 then y := x else y := −x $\{y \geq 0\}$

While Loops

- The last rule we're missing is that for while-loops

- This is the trickiest part, and we first need to understand the concept of **loop invariants**

- A loop invariant I has the following properties:

    - I holds *before* the loop

    - It still holds *after* each loop iteration

While Loops: Example

Quiz:

- Consider the following code

  i := 0; j := 0; n := 10; while i < n do i := i + 1; j := i + j

- Which of the following is a loop invariant?

  - $i \leq n$
  - $i < n$
  - $j \geq n$

- Suppose *I* is a loop invariant. Does it also hold after the loop terminates?

While Loops

- Consider the following code   while b do S

- Say I is an invariant for this loop. What holds after the loop terminates?

$$\frac{\vdash \{I \wedge b\}\ \mathrm{s}\ \{I\}}{\vdash \{I\}\ \text{while } b \text{ do } s\ \{I \wedge \neg b\}}$$

- This rule says, if I is a loop invariant, then $I \wedge \neg b$ must hold after the loop terminates

- How does this rule ensure that I is a loop invariant?

While Loops

Example:

- Let's consider the statement W: while $x < n$ do $x := x + 1$
- Let's prove validity of $\vdash \{x \leq n\}\ W\ \{x \geq n\}$
- What's a good loop invariant? Let's use $I \triangleq x \leq n$

$$\frac{\vdash \{I \wedge b\}\ s\ \{I\}}{\vdash \{I\}\ \text{while } b \text{ do } s\ \{I \wedge \neg b\}}$$

Quiz:

- Would $\top$ also have worked as loop invariant?
- What if we changed the post-condition to $x = n$?

Recap: While Loops

- We saw the proof rule for while loops:

$$\frac{\vdash \{I \wedge b\}\ \mathrm{s}\ \{I\}}{\vdash \{I\}\ \text{while}\ b\ \text{do}\ s\ \{I \wedge \neg b\}}$$

- The rule requires us to provide loop invariant I with the following properties:
    - I holds *before* the loop
    - It still holds *after* each loop iteration

Inductive Invariants

Quiz:

- Consider the following program:

$$\vdash \{I \wedge b\}\ s\ \{I\}$$
$$\overline{\vdash \{I\}\ \text{while } b \text{ do } s\ \{I \wedge \neg b\}}$$

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

- Let us pick I = j≥1. Is I a loop invariant?

- Can we prove the Hoare triple $\{j \geq 1 \wedge i < n\}$ j := j + i; i := i + 1 $\{j \geq 1\}$?

- What about the strengthened invariant $I \triangleq$ j≥1 ∧ i≥1 ?

Inductive Invariants

$$\frac{\vdash \{I \wedge b\}\ s\ \{I\}}{\vdash \{I\}\ \text{while}\ b\ \text{do}\ s\ \{I \wedge \neg b\}}$$

- Not all invariants can be used in the proof rule for while!
- Invariants like $I \triangleq j \geq 1 \wedge i \geq 1$ are called <u>inductive invariants</u>
- Only inductive invariants can be used to prove program correctness
- The Hoare proof rule requires us to supply an inductive invariant to prove correctness
- A key challenge in verification is finding those inductive invariants

Inductive Invariants

Example:

$$\frac{\vdash \{I \wedge b\}\ s\ \{I\}}{\vdash \{I\}\ \text{while}\ b\ \text{do}\ s\ \{I \wedge \neg b\}}$$

- Consider the following statement W:

  while x < n do x:=y; y:=x+1

- We want to prove the following Hoare triple:

  $\{x=0 \wedge y=1\}\ \mathrm{W}\ \{x \geq 0\}$

- What is an inductive invariant I that allows us to prove the triple?

Inductive Invariants

<u>Example:</u>

$$\frac{\vdash \{I \wedge b\}\ \mathrm{s}\ \{I\}}{\vdash \{I\}\ \text{while } b \text{ do } s\ \{I \wedge \neg b\}}$$

- Let's try $x \geq 0$
- We have to prove $\vdash \{x \geq 0 \wedge x < n\}$ x:=y; y:=x+1 $\{x \geq 0\}$
- Can we prove this triple? No, I is <u>not inductive</u>. What information is missing?
- Let us instead try invariant $I \triangleq x \geq 0 \wedge y = x + 1$
- We have to prove $\vdash \{x \geq 0 \wedge y = x + 1\}$ x:=y; y:=x+1 $\{x \geq 0 \wedge y = x + 1\}$
- This invariant is inductive and concludes the proof

Arrays

- What if we add arrays to Nano?

$$a[e_1] := e_2$$

- What proof rule should we add for this statement?

- Let's try treating arrays like assignments

$$\frac{}{\vdash \{Q[e_2/a[e_1]]\}\ a[e_1] := e_2\ \{Q\}}$$

- Is this rule correct?

Arrays

- No! Let's look at a counterexample!

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

$$\frac{}{\vdash \{Q[e_2/a[e_1]]\}\ a[e_1] := e_2\ \{Q\}}$$

- What's the value of a[i] after the program?
- But we're able to prove post-condition $a[i] = 3$ using to proof rule
- Clearly this rule is unsound!

Arrays

- Here is the correct proof rule:

$$\frac{}{\vdash \{Q[a\langle e_1 \triangleleft\ e_2\rangle/a]\}\ a[e_1] := e_2\ \{Q\}}$$

- Substitute a by <u>the array</u> where position e_1 is set to e_2

- Reasoning about this requires the theory of arrays

Arrays

Example:

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad}$$

$$\vdash \{Q[a\langle e_1 \triangleleft e_2\rangle/a]\}\ a[e_1] := e_2\ \{Q\}$$

- Let's look again at our example

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

$$\frac{}{\vdash \{Q[a\langle e_1 \lhd e_2\rangle / a]\}\ a[e_1] := e_2\ \{Q\}}$$

Arrays

Example:

- Let's consider the following loop

$$\overline{\vdash \{Q[a\langle e_1 \triangleleft e_2\rangle/a]\}\ a[e_1] := e_2\ \{Q\}}$$

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

- Suppose our precondition is $i = 0 \wedge n > 0$ and postcondition is $\forall j.\ 0 \leq j < n \rightarrow a[j] = 0$

- What's an inductive invariant that shows correctness?

$$\vdash \{Q[a\langle e_1 \lhd e_2\rangle/a]\}\ a[e_1] := e_2\ \{Q\}$$

Soundness & Completeness

- One can show that the proof rules for Hoare logic are **sound**:
  - If $\vdash \{P\}$ s $\{Q\}$ then $\vDash \{P\}$ s $\{Q\}$
- That is, if we can prove triple $\{P\}$ s $\{Q\}$, then it is indeed valid
- Soundness ensures that we cannot prove invalid Hoare triples
- But, are there valid Hoare triples we cannot prove?
- Completeness: If $\vDash \{P\}$ s $\{Q\}$ then $\vdash \{P\}$ s $\{Q\}$
- Completeness only holds with an important caveat

Relative Completeness

- Completeness: If $\models \{P\}\ \mathrm{s}\ \{Q\}$ then $\vdash \{P\}\ \mathrm{s}\ \{Q\}$

- Completeness only holds with an important <u>caveat</u>
- Rules for precondition strengthening and postcondition weakening require checking A $\Rightarrow$ B

- Thus, we need to be able to prove such assertions, but for undecidable logics (Peano arithmetic, arrays), this might not always be possible

- Thus, Hoare's proof rules guarantee **relative completeness**

- Given an oracle for deciding A $\Rightarrow$ B, any valid Hoare triple can be proven using our rules

- In particular, if we stick to decidable logics, our rules are complete

Relative Completeness

- Completeness: If $\models \{P\}\ s\ \{Q\}$ then $\vdash \{P\}\ s\ \{Q\}$

- You can find a full proof of relative completeness here: https://web.eecs.umich.edu/~weimerw/2006-615/reading/Necula-Axiomatic-Complete.pdf

- For now, we move on to the question of how to automate proofs using Hoare logic

Automating Proofs in Hoare Logic

- Writing out proofs for programs is tedious, so we want to automate it

- Idea: we're given loop invariants, but automate the remaining reasoning steps

- For now, we assume that loop invariants are given by the user (of the verification tool)

- We will show how to automate finding them later

Automating Proofs in Hoare Logic

- 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:
  - Generate VC's from source code
  - Use SMT solver to check validity of formulas

Weakest Preconditions

- Idea: Suppose we want to verify Hoare triple $\{P\}$ s $\{Q\}$

- We'll start with Q and, going backwards, compute a 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, the triple $\{P\}$ s $\{Q\}$ is valid, iff $P \Rightarrow$ **wp**(s, Q)

- This check can be automated by the SMT solver

Weakest Preconditions

- What's the weakest precondition for $\{?\}$ x := e $\{Q\}$

$$wp(x := e, Q) \triangleq Q[e/x]$$

- For composition s_1; s_2 we can compute the weakest precondition as

$$wp(s_1; s_2, Q) \triangleq wp(s_1, wp(s_2, Q))$$

- For an if-statement $\{?\}$ if b then s_1 else s_2 $\{Q\}$ we get

$$wp(\text{if b then } s_1 \text{ else } s_2, Q) \triangleq (b \rightarrow \quad wp(s_1, Q)) \quad \wedge (\neg b \rightarrow wp(s_2, Q))$$

Weakest Preconditions

Quiz:

- Consider the statement s:

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

- What is $wp(s, z>0)$?

- What is $wp(s, z\leq 0)$?

- Can we prove $\{-1 \leq y\}\ s\ \{z>0\}$?

- What about $\{y > -1\}\ s\ \{z>0\}$?

Weakest Preconditions: Loops

- What's the weakest precondition for a loop: W $\triangleq$ while b do s
- From our semantics, we know that we can unwind the loop as follows

  if b then s else skip; while b do s

- Then, we can derive
  - wp$(W, Q) = \ (b \rightarrow$ wp$(W, Q)) \wedge \quad (\neg b \rightarrow Q)$
- But that's a recursive equation, so we're not really any further
- Idea: our Hoare logic proofs used invariants. Let's compute wp wrt. a given invariant

Weakest Preconditions: Loops

- What's the weakest precondition for a loop?
- Say all our loops are annotated with a loop invariant I

$$\mathrm{W} \triangleq \text{while } [I]\ b \text{ do } s$$

Quiz:

- Is it sound to let wp$(W, Q) \triangleq I$?

Weakest Preconditions: Loops

- What's the weakest precondition for a loop?
- Say all our loops are annotated with a loop invariant I

$$\text{W} \triangleq \text{while } [I]\ b \text{ do } s$$

Quiz:

- Is it sound to let wp$(W, Q) \triangleq I$?
- No! We need to check that I implies post-condition Q
- We need to check that I is actually a loop invariant!
- Define function vc(s) that encodes these additional conditions

Verification Conditions

- How should we define $\text{vc}(\text{while } [I]\ b \text{ do } s, Q)$?
- We need to ensure that Q holds *after* the loop, that is $(I \wedge \neg b) \Rightarrow Q$
- I needs to be a loop invariant, that is, needs to be preserved under s, i.e.,

$$\{I \wedge b\}\ \text{s}\ \{I\}$$

- We can prove this by showing $I \wedge b \Rightarrow \text{wp}(s, I) \wedge \text{vc}(\text{s}, I)$
- This means, we can define

$$\text{vc}(\text{while } I\ b \text{ do } s, Q) \triangleq (I \wedge b \Rightarrow \text{wp}(s, I)) \wedge \text{vc}(\text{s}, I) \wedge (I \wedge \neg b) \Rightarrow Q$$

Verification Conditions

Quiz: $\text{vc}(\text{while } [I]\ b \text{ do } s, Q) \triangleq (I \wedge b \Rightarrow \text{wp}(s, I)) \wedge \text{vc}(s, I) \wedge (I \wedge \neg b) \Rightarrow Q$

- Let W $\triangleq$ while $[x \leq 6]\ x \leq 5$ do $x := x+1$; we want to prove $\{x \leq 0\}$ W $\{x = 6\}$
- What do we get for vc(W, $x=6$)?

Verification of Hoare Triples

- To show validity of a Hoare triple $\{P\}$ s $\{Q\}$, we thus need to
  - Compute $\mathrm{wp}(s, Q)$
  - Compute $\mathrm{vc}(s, Q)$
- Then $\{P\}$ s $\{Q\}$ is valid, if the following formula is valid

$$\mathrm{vc}(s, Q) \wedge (P \rightarrow \mathrm{wp}(s, Q)) \quad (*)$$

- Thus, if we prove (*), we have shown that the program conforms to its specification

Verification of Hoare Triples

Quiz:

- Is our method complete, that is if $\{P\}$ s $\{Q\}$, then the following formula is valid

$$\mathrm{vc}(s, Q) \wedge (P \rightarrow \mathrm{wp}(s, Q)) \quad (*)$$

Verification of Hoare Triples

Quiz:

```
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? $sum \geq 0$?

$$\mathrm{vc}(\mathrm{while}\ [I]\ b\ \mathrm{do}\ s, Q) \triangleq (I \land b \Rightarrow \mathrm{wp}(s, I)) \land \mathrm{vc}(\mathrm{s}, I) \land (I \land \neg b) \Rightarrow Q$$

Verification of Hoare Triples

- For the assignment, you will implement a function

vcgen :: Nano.Stmt -> Logic.Base -> (Logic.Base, [Logic.Base])

wp

List of extra VCs for invariants

- Takes as inputs Nano programs annotated with invariants

- Check that VCs hold

Extensions & Plan

- Nano is missing many features of real programming languages
- In the next lecture, we will look at two extensions:
    - Functions
    - Pointers
- After that, we'll look at techniques to discover loop invariants (semi-) automatically