CS780: Automated Logical Reasoning

Lecture 19: Hoare Logic

İşıl Dillig
Overview

▶ Today’s lecture: Applications of logic in formally proving correctness of computer programs

▶ Specifically, we’ll look at Hoare logic: a set of logical inference rules for reasoning about program correctness

▶ Hoare logic is also basis of axiomatics semantics of programming languages

▶ 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”
Motivation for Verifying Programs

- Typical way to ensure software quality: testing

- Testing is good at finding presence of bugs

- But in general, testing cannot guarantee absence of bugs

- Goal of program verification is to mathematically prove program obeys specification

- Thus, verification useful for guaranteeing absence of certain classes of errors

- Especially useful for applications where correctness particularly important: car braking systems, medical equipment, voting machines, ...
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 \( C := \text{true} \mid \text{false} \mid e_1 = e_2 \mid e_1 \leq e_2 \)

  Statement \( S := \)
  
  \( V := E \quad \text{(Assignment)} \)
  
  \( S_1; S_2 \quad \text{(Composition)} \)
  
  if \( C \) then \( S_1 \) else \( S_2 \) \quad \text{(If)}
  
  while \( C \) do \( S \) \quad \text{(While)}
Partial Correctness Specification

In Hoare logic, we specify **partial correctness** of programs using specifications of the form:

\[
\{ P \} S \{ Q \}
\]

- Here, \( S \) is a statement in programming language IMP
- \( P \) and \( Q \) are first-order logic formulas over program variables
- \( P \) is called **precondition** and \( Q \) is called **post-condition**
- Partial correctness spec \( \{ P \} S \{ Q \} \) is called **Hoare triple**
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 $\{x = 0\} x = x + 1 \{x = 1\}$ valid Hoare triple? yes

- What about $\{x = 0 \land y = 1\} x = x + 1 \{x = 1 \land y = 2\}$? no

- What about $\{x = 0\} x = x + 1 \{x = 1 \lor y = 2\}$? yes

- What about $\{x = 0\} \text{while true do } x = 0 \{x = 1\}$? yes
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 spec. 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 $[x = 0]\text{while true do } x = 0[x = 0]$ valid? no
Examples, Safety, Liveness

- What does $\{true\}S\{Q\}$ say? If $S$ terminates, then $Q$ holds

- What about $\{P\}S\{true\}$? holds for any $P$ and any $S$

- What about $[P]S[true]$? If $P$ holds, then $S$ terminates

- When does $\{true\}S\{false\}$ hold? If $S$ does not terminate

- In the rest of lecture, we’ll focus on only partial correctness

- Partial correctness specs also known as safety properties

- Total correctness = Partial correctness + termination

- Termination is liveness property

- Proving safety is easier than liveness
Hoare triples are just partial correctness specifications, but not very useful on their own!

We would like to prove program obeys given partial correctness specification

Hoare also gave a sound proof system that allows semi-mechanizing correctness proofs

Proof rules in Hoare logic are written as inference rules
Inference Rules

- Inferences rules are of the following form:

\[ \vdash \{ P_1 \} S_1 \{ Q_1 \} \ldots \vdash \{ P_n \} S_n \{ Q_n \} \]
\[ \vdash \{ P \} S \{ Q \} \]

- 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
Proof Rule for Assignment

⊢ \{ Q[E/x] \} x = E \{ Q \}

- This is probably the most difficult to understand proof rule

- 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?
  
  - $\{ y = 4 \} x = 4 \{ y = x \}$ yes
  
  - $\{ x + 1 = n \} x = x + 1 \{ x = n \}$ yes
  
  - $\{ y = x \} y = 2 \{ y = x \}$ no
  
  - $\{ z = 3 \} y = x \{ z = 3 \}$ yes
Motivation for Precondition Strengthening

- Is the Hoare triple \( \{ z = 2 \} y = x \{ y = x \} \) valid? \( \text{yes} \)

- Is this Hoare triple provable using our assignment rule? \( \text{no} \)

- Instantiating the assignment rule, we get:

\[
\{ y = x[x/y] x = x \text{true} \} y = x \{ y = x \}
\]

- Thus, although this Hoare triple is perfectly valid, assignment rule only allows us to prove \( \{ \text{true} \} y = x \{ y = x \} \)

- But intuitively, if we can prove \( y = x \) w/o any assumptions (i.e., precondition is true), we should also be able to prove it if we do make assumptions!

- This motivates need for precondition strengthening
Proof Rule for Precondition Strengthening

\[
\frac{
\vdash \{P'\} S\{Q\} \quad P \Rightarrow P'
}{
\vdash \{P\} S\{Q\}
}
\]

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

- Proof:

\[
\begin{align*}
\vdash \{y = x[x/y]\} y = x \{y = x\} \\
\vdash \{\text{true}\} y = x \{y = x\} \\
\vdash \{z = 2\} y = x \{y = x\}
\end{align*}
\]
We also need a dual rule for post-conditions called **post-condition weakening**:

\[
\begin{align*}
\vdash \{ P \} S \{ Q' \} & \quad Q' \Rightarrow Q \\
\vdash \{ P \} S \{ Q \}
\end{align*}
\]

Suppose we can prove \( \{ \text{true} \} S \{ x = y \land z = 2 \} \).

Using post-condition weakening, which of these can we prove?

- \( \{ \text{true} \} S \{ x = y \} \) yes
- \( \{ \text{true} \} S \{ z = 2 \} \) yes
- \( \{ \text{true} \} S \{ z > 0 \} \) yes
Proof Rule for Composition

\[
\begin{array}{c}
\vdash \{P\} S_1 \{Q\} \quad \vdash \{Q\} S_2 \{R\} \\
\vdash \{P\} S_1 ; S_2 \{R\}
\end{array}
\]

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

\[
\{\text{true}\} \ x = 2; \ y = x \ \{y = 2 \land x = 2\}
\]

- What is appropriate \( Q? \) \( x = 2 \)

\[
\begin{array}{c}
\{x = 2[2/x]\} x = 2 \{x = 2\} \\
\{x = 2 \land y = 2[x/y]\} y = x \ {x = 2 \land y = 2}\end{array}
\]

\[
\begin{array}{c}
\{\text{true}\} x = 2 \ {x = 2}\end{array}
\]

\[
\begin{array}{c}
\{x = 2\} y = x \ {x = 2 \land y = 2}\end{array}
\]

\[
\vdash \{\text{true}\} x = 2; \ y = x \ \{y = 2 \land x = 2\}
\]
Proof Rule for If Statements

\[
\begin{align*}
\vdash \ & \{P \land C\} & S_1 \ & \{Q\} \\
\vdash \ & \{P \land \neg C\} & S_2 \ & \{Q\}
\end{align*}
\]

\[
\vdash \{P\} \text{ if } C \text{ then } S_1 \text{ else } S_2 \ & \{Q\}
\]

- 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? \(P \land C\)

- Thus, in the then branch, we want to show \(\{P \land C\} S_1 \{Q\}\)

- At beginning of else branch, what facts do we know? \(P \land \neg C\)

- What do we need to show in else branch? \(\{P \land \neg C\} S_2 \{Q\}\)
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.

- **Question:** Suppose $I$ is a loop invariant. Does $I$ also hold after loop terminates?

- **Yes** because, by definition, $I$ holds after every loop iteration, including after the last one.
Proof Rule for While

- Consider the statement \( \text{while } C \text{ do } S \)

- Suppose \( I \) is a loop invariant for this loop. What is guaranteed to hold after loop terminates? \( I \land \neg C \)

- Putting all this together, proof rule for while is:

\[
\begin{align*}
\vdash & \{P \land C\} S\{P\} \\
\vdash & \{P\} \text{while } C \text{ do } S\{P \land \neg C\}
\end{align*}
\]

- This rule simply says "If \( P \) is a loop invariant, then \( P \land \neg C \) must hold after loop terminates"

- Based on this rule, why is \( P \) a loop invariant?

- Because \( P \) holds initially and is preserved after each iteration
Example

Consider the statement $S = \text{while } x < n \text{ do } x = x + 1$

Let’s prove validity of $\{x \leq n\}S\{x \geq n\}$

What is appropriate loop invariant? $x \leq n$

First, let’s prove $x \leq n$ is loop invariant. What do we need to show? $\{x \leq n \land x < n\}x = x + 1\{x \leq n\}$

What proof rules do we need to use to show this? assignment, precondition strengthening

$$\vdash \{x \leq n \upharpoonright x + 1/x\}x = x + 1\{x \leq n\} \vdash \{x + 1 \leq n\}x = x + 1\{x \leq n\} \vdash \{x \leq n \land x < n\}x = x + 1\{x \leq n\}$$
Example, cont

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

\[
\begin{align*}
\vdash \{x \leq n \land x < n\} S' \{x \leq n\} \\
\vdash \{x \leq n\} & \text{while } x < n \text{ do } S' \{x \leq n \land \neg (x < n)\}
\end{align*}
\]

- 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? postcondition weakening
Example, cont.

The full proof:

\[
\begin{align*}
\vdash \{ x + 1 \leq n \} x &= x + 1 \{ x \leq n \} \\
x &\leq n \land x < n \Rightarrow x + 1 < n \\
\vdash \{ x \leq n \land x < n \} x &= x + 1 \{ x \leq n \} \\
\vdash \{ x \leq n \} S \{ x \leq n \land \neg (x < n) \} &\quad x \leq n \land \neg (x < n) \Rightarrow x \geq n \\
\vdash \{ x \leq n \} S \{ x \geq n \}
\end{align*}
\]
Summary of Proof Rules

1. \( \vdash \{ Q[E/x] \} \ x = E \ \{ Q \} \) (Assignment)

2. \( \vdash \{ P' \} S \{ Q \} \quad P \Rightarrow P' \)
   \[ \vdash \{ P \} S \{ Q \} \]
   (Strengthen P)

3. \( \vdash \{ P \} S \{ Q' \} \quad Q' \Rightarrow Q \)
   \[ \vdash \{ P \} S \{ Q \} \]
   (Weaken Q)

4. \( \vdash \{ P \} C_1 \{ Q \} \quad \vdash \{ Q \} C_2 \{ R \} \)
   \[ \vdash \{ P \} C_1; C_2 \{ R \} \]
   (Composition)

5. \( \vdash \{ P \} \text{if} \ C \text{then} \ S_1 \text{ else} \ S_2 \ \{ Q \} \)
   (If)

6. \( \vdash \{ P \} S \{ P \} \)
   \[ \vdash \{ P \} \text{while} \ C \text{ do} \ S \{ P \land \neg C \} \]
   (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

- Won’t do the proof in class, but to prove this claim, roughly need to do following:
  1. Give operational semantics for language IMP
  2. Prove equivalence between Hoare proof rules and operational semantics by structural induction
Meta-theory: Relative Completeness

- 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!

- **Recall**: Rules for precondition strengthening and postcondition weakening require checking \( A \Rightarrow B \)

- In general, these formulas belong to Peano arithmetic

- As we saw earlier, Godel’s incompleteness implies Peano arithmetic is incomplete

- Thus, there are implications that are valid but we cannot show to be valid using any proof system
Meta-theory: Relative Completeness

- 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

- However, this relative completeness result only holds for very simple languages like IMP

- Result of Ed Clarke: For languages combining certain constructs (e.g., higher-order functions, recursion) not possible to give a sound and relatively complete proof system
Automating Reasoning in Hoare Logic

- Hoare’s proof rules provide a sound system to show properties about programs

- However, these proof rules are not immediately amenable to automation

- Number of places in proof system that require insight:
  1. When do we apply precondition strengthenning?
  2. When do we apply postcondition weakening?
  3. Most difficult: How do we come up with loop invariants to use in rule for while?
Loop Invariants

- In rest of lecture, we’ll assume some oracle gives us appropriate invariants for each program

- This oracle can either be a human or a static analysis tool

- Static analysis techniques, such as abstract interpretation, can automatically infer loop invariants

- However, we won’t discuss how to automate loop invariant discovery in this lecture
Basic Idea Behind Program Verification

- Automating Hoare logic is based on generating verification conditions (VC)

- A verification condition is a formula $\phi$ generated automatically from source code and annotated loop invariants

- Furthermore, the program obeys specification if $\phi$ is valid

- In this model, program verification has two components:
  
  1. Generate VC’s from source code
  
  2. Use theorem prover to check validity of formulas from step 1

- We’ve seen how to do (2) all semester, so our focus today is generating VC’s
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
- Also useful if we know precondition and want to compute what postcondition program guarantees
- Forwards technique based on computing strongest postconditions (sp)
In contrast, backwards analysis starts from postcondition and tries to prove precondition.

Especially useful if we have assertion to prove and want to know which precondition is needed.

Backwards technique based on computing weakest preconditions (wp).

Today, we’ll focus on backwards, wp-based method for generating verification conditions.
Weakest Preconditions

- **Idea:** Suppose we want to verify Hoare triple \( \{ P \} S \{ Q \} \)

- We’ll start with \( Q \) and going backwards, compute formula \( \text{wp}(S, Q) \) called **weakest precondition** of \( Q \) w.r.t. to \( S \)

- \( \text{wp}(S, Q) \) 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:
  \[
P \Rightarrow \text{wp}(S, Q)
  \]

- **Why?** Because if triple \( \{ P' \} S \{ Q \} \) is valid and \( P \Rightarrow P' \), then \( \{ P \} S \{ Q \} \) is also valid

- However, \( P \) cannot be weaker than \( \text{wp}(S, Q) \); otherwise, \( Q \) may not hold in some execution
Defining Weakest Preconditions

- Weakest preconditions are defined inductively and follow Hoare's proof rules

- \( wp(x = E, Q) = Q[E/x] \)

- \( wp(s_1; s_2, Q) = wp(s_1, wp(s_2, Q)) \)

- \( wp(\text{if } C \text{ then } s_1 \text{ else } s_2, Q) = \)
  \( C \rightarrow wp(s_1, Q) \land \neg C \rightarrow wp(s_2, Q) \)

- This says "If \( C \) holds, \( wp \) of then branch must hold; otherwise, \( wp \) of else branch must hold"
Defining Weakest Preconditions, cont.

- We have one case left for while loops: \( wp(\text{while } C \text{ do } s, Q) \)

- To define \( wp \) for loops, we will first use the following equivalence:

\[
\text{while } C \text{ do } s \equiv \text{if } C \text{ then } s; \text{while } C \text{ do } s \text{ else } \text{skip}
\]

- Let \( W = wp(\text{while } C \text{ do } s, Q) \)

- Now, using the definition of \( wp \) for \text{if}, we have,

\[
W = (C \rightarrow wp(s; \text{while } C \text{ do } s, Q) \land \neg C \rightarrow Q)
\]
Defining Weakest Preconditions, cont.

\[ W = (C \rightarrow wp(s; while \ C \ do \ s, Q) \land \neg C \rightarrow Q) \]

- **Observe:** \( \wp(s; while \ C \ do \ s, Q) \) is equivalent to:
  \[ \wp(s, \wp(\text{while } C \ do \ s, Q)) \]

- **Recall:** \( \wp(\text{while } C \ do \ s, Q) \) is just \( W \)!

- Thus, whole RHS can be written as:
  \[ W = (C \rightarrow wp(s, W) \land \neg C \rightarrow Q) \]

- This is a recursive equation!

- In general, not possible to compute weakest preconditions for loops exactly
Approximating Weakest Preconditions

- Since we can’t compute weakest preconditions exactly, we will approximate them to generate verification conditions.

- Suppose we want to verify Hoare triple \( \{P\} S \{Q\} \).

- Now, instead of computing exact \( wp(S, Q) \), we will approximate it using \( awp(S, Q) \).

- \( awp(S, Q) \) may be stronger than \( wp(S, Q) \) but not weaker.

- Thus, to verify \( \{P\} S \{Q\} \), necessary to 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 Preconditions

- 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

- “Oracle” can be human or any automated loop invariant generation technique

- Thus, we’ll assume all loops are of the form
  $$\text{while } C \text{ do } \{I\} \ S$$
  where $I$ is a loop invariant

- Now, we’ll just define $awp(\text{while } C \text{ do } \{I\} \ S, Q) \equiv I$

- This is ok because for $I$ to be loop invariant, it must hold before the loop
Verification with Approximate Weakest Preconditions

- Now suppose we want to verify \( \{P\} S \{Q\} \)

- If \( P \Rightarrow awp(S, Q) \), does this mean \( \{P\} S \{Q\} \) is valid?

- No, two problems with \( awp(\text{while } C \text{ do } \{I\} S, Q) \)
  
  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 \( Q \)!

- Thus, we’ll generate additional verification conditions to ensure these two conditions hold!

- For each statement \( S \), we’ll define \( VC(S, Q) \) that encodes additional conditions that must be checked
Generating Verification Conditions

- Most interesting VC generation rule is for loops:

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

- 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 $S' = \text{while } C \text{ do } \{I\} \ S$ 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.

- Thus, VC generation rules for other statement simply propagate VC’s for nested loops up.

- \( VC(x = E, Q) = true \)

- \( VC(s_1; s_2, Q) = VC(s_2, Q) \land VC(s_1, awp(s_2, Q)) \)

- \( VC(\text{if } C \text{ then } s_1 \text{ else } s_2, Q) = VC(s_1, Q) \land VC(s_2, Q) \)
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:

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

- Question: If $\{P\} S \{Q\}$ is valid, is $(\ast)$ 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
Discussion, cont.

- Generating verification conditions using weakest preconditions gives a way to automate reasoning in Hoare logic.

- **But** assuming either a human or some other tool gives us appropriate loop invariants.

- In general, finding inductive loop invariants not easy even for humans – requires intuition, understanding of program.

- Static analysis techniques can automate task of finding loop invariants.
Loop Invariants and Other Complications

- Unfortunately, static analysis tools can only infer a certain "class" of invariants

- There could be very complicated loop invariants that no automated static analysis will be able to infer

- Thus, impossible to avoid false alarms in general

- Also, considered verifying programs in simple IMP language

- In reality, verification more involved b/c real languages have many more features: pointers, unbounded data structures, dynamic memory allocation, function calls, concurrency, . . .

- If you want to find out more, take Tom’s verification/static analysis class next fall!