

Specifications and Hoare logic

From contracts to program proofs

CMPUT 664 • Formal Verification

Today

1. Specifications, states, and Hoare triples
2. Proof rules for loop-free commands
3. Soundness of Hoare-logic proofs

Specifications

A specification states what behavior a system must exhibit. Different kinds of behavior call for different forms of specification.

Form	Example
Pre/postcondition	If $x \geq 0$ before <code>y := x + 1</code> , then $y > 0$ afterward
Trace	An allowed event sequence is <code>login</code> , zero or more <code>read</code> events, then <code>logout</code>
Temporal behavior	Every request is eventually followed by a response
Information flow	Changing a secret input does not change the public output
Probability	A message arrives within two seconds with probability at least 0.999

The specification form studied today

Hoare logic describes pre/postcondition relationships for commands:

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

- P describes the initial states in which execution is permitted
- S is the command being specified
- Q describes the property required of every terminating final state

We first define states, assertions, and commands; then we can state this relationship precisely.

Program states

For integer variables, a state is a mapping

$$\sigma : \text{Var} \rightarrow \mathbb{Z}.$$

An assertion is a logical formula interpreted over a state.

$$\sigma \models x > 0 \wedge y = x + 1$$

means that the values assigned by σ satisfy the formula.

Assertions will serve as preconditions and postconditions.

A small command language

We use commands generated by

$$S ::= \mathbf{skip} \mid x := E \mid S_1; S_2 \\ \mid \mathbf{if } B \mathbf{ then } S_1 \mathbf{ else } S_2.$$

Expressions and guards are evaluated in the current state.

Today we restrict attention to loop-free commands. Loops require an additional assertion and come next.

Hoare triples

A partial-correctness specification has the form

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

- P is the precondition
- S is the command
- Q is the postcondition

P and Q are assertions about program states.

Meaning of partial correctness

The triple

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

is valid when, for every initial state satisfying P :

if execution of S terminates, its final state satisfies Q .

Partial and total correctness

Partial correctness

If execution terminates, Q holds

Nontermination does not falsify the triple

Safety-style guarantee

Total correctness

Execution terminates and Q holds

Nontermination falsifies the specification

Safety plus termination

Every command in today's loop-free language terminates, assuming expression evaluation terminates. The distinction matters when we add loops.

Are these triples valid?

1. $\{x = 0\} \ x := x + 1 \ \{x = 1\}$
2. $\{x = 0 \wedge y = 1\} \ x := x + 1 \ \{x = 1 \wedge y = 2\}$
3. $\{x = 0\} \ x := x + 1 \ \{x = 1 \vee y = 2\}$
4. $\{\text{true}\} \ x := x + 1 \ \{x > 0\}$

Preconditions and postconditions at the extremes

What do these specifications require?

$\{\text{true}\} S \{Q\}$ $\{P\} S \{\text{true}\}$

$\{\text{false}\} S \{Q\}$ $\{\text{true}\} S \{\text{false}\}$

A proof system

From semantic validity to derivations

Validity and provability

Semantic validity:

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

means the specification holds for program executions.

Syntactic provability:

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

means the triple has a derivation using the Hoare-logic rules.

The proof system should derive only semantically valid triples.

Inference rules

An inference rule has premises above a line and a conclusion below it:

$$\frac{\vdash \{P_1\}S_1\{Q_1\} \quad \cdots \quad \vdash \{P_n\}S_n\{Q_n\}}{\vdash \{P\}S\{Q\}}.$$

A derivation is a tree whose leaves are rules without Hoare-triple premises or logical side conditions.

The outermost command determines the main proof rule:

Command	Main rule
skip	Skip
$x := E$	Assign
$S_1; S_2$	Seq
if	If

Proof rules do not fix a reasoning direction

An inference rule is not an arrow describing program execution. It states that if its premises hold, then its conclusion holds.

We can use the same proof system in two ways:

Backward proof construction	Forward proof construction
Start with the triple we want to prove	Start with facts or triples already established
Match the goal against a rule's conclusion	Apply a rule to derive a new conclusion
Generate proof obligations for the premises	Propagate known information through the program
Often uses weakest preconditions	Often uses strongest postconditions

This lecture mainly constructs proofs backward. Later, weakest preconditions make that choice a mechanical calculation. Hoare logic itself permits either style.

Skip rule

$$\frac{}{\vdash \{P\} \textbf{skip} \{P\}} \text{Skip}$$

Because **skip** does not change the state, every assertion true before it remains true afterward.

Assignment: reason backward from the postcondition

Consider

$$x := y + 1$$

with desired postcondition

$$x > 5.$$

Before the assignment, it is sufficient that

$$y + 1 > 5.$$

Assignment rule

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

$Q[E/x]$ denotes capture-avoiding substitution of E for the free occurrences of x in Q .

The substitution describes the value that x will have after the assignment.

Apply the assignment rule

Which triples follow directly from the assignment rule?

1. $\{y = 4\} x := 4 \{y = x\}$
2. $\{x + 1 = n\} x := x + 1 \{x = n\}$
3. $\{y = x\} y := 2 \{y = x\}$
4. $\{z = 3\} y := x \{z = 3\}$

The assignment rule gives a specific precondition

For the postcondition $y = x$, the assignment rule gives

$$\vdash \{\text{true}\} y := x \{y = x\}.$$

The triple

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

is also valid, but it is not the literal instance produced by substitution. We need a rule for adapting preconditions and postconditions.

Strength of assertions

P is stronger than P' when

$$\models P \rightarrow P'.$$

Examples:

$x = 3$ is stronger than $x > 0$,

$x > 0 \wedge y = 2$ is stronger than $x > 0$.

A stronger assertion describes fewer states.

This ordering lets us reuse a proof under stronger assumptions or for a weaker requested result.

Rule of consequence

$$\frac{\models P \rightarrow P' \quad \vdash \{P'\}S\{Q'\} \quad \models Q' \rightarrow Q}{\vdash \{P\}S\{Q\}} \text{Consequence}$$

We may strengthen the required precondition and weaken the guaranteed postcondition.

Precondition strengthening

$$\frac{\models P \rightarrow P' \quad \vdash \{P'\}S\{Q\}}{\vdash \{P\}S\{Q\}} \text{ Pre-Strengthen}$$

If S works for every state satisfying P' , it also works when callers satisfy the stronger assertion P .

For example, $x = 3 \Rightarrow x > 0$, so a proof from $x > 0$ also applies to callers with $x = 3$.

Postcondition weakening

$$\frac{\vdash \{P\}S\{Q'\} \quad \models Q' \rightarrow Q}{\vdash \{P\}S\{Q\}} \text{ Post-Weaken}$$

If S establishes Q' , it also establishes any weaker assertion Q .

For example, $x = 3 \Rightarrow x > 0$, so a proof that S establishes $x = 3$ also proves that it establishes $x > 0$.

Logical side conditions

Applications of consequence create logical obligations such as

$$\models P \rightarrow P' \quad \text{or} \quad \models Q' \rightarrow Q.$$

An SMT solver can check validity by searching for a model of

$$P \wedge \neg P' \quad \text{or} \quad Q' \wedge \neg Q.$$

The remaining rules describe how proofs follow the structure of compound commands.

Sequential composition

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

R describes the intermediate state between the two commands.

To use the rule, the postcondition of S_1 must match the precondition of S_2 .

Choose the intermediate assertion

Consider

$$x := x + 1; \quad y := 2x$$

with postcondition $y > 10$.

Before the second assignment, substitution gives

$$2x > 10.$$

This becomes the intermediate assertion used by the sequence rule.

Continue backward

Before $x := x + 1$, substitute $x + 1$ into $2x > 10$:

$$2(x + 1) > 10.$$

Thus one derivable triple is

$$\{2(x + 1) > 10\} \ x := x + 1; \ y := 2x \ \{y > 10\}.$$

Here, arithmetic simplification gives the logically equivalent precondition $x > 4$.

The same backward reasoning applies separately to each branch of a conditional.

Conditional rule

$$\frac{\vdash \{P \wedge B\} S_1 \{Q\} \quad \vdash \{P \wedge \neg B\} S_2 \{Q\}}{\vdash \{P\} \text{ if } B \text{ then } S_1 \text{ else } S_2 \{Q\}} \text{ If}$$

Each branch is verified under the condition that selects it.

Example: absolute value

$\{\text{true}\} \text{ if } x \geq 0 \text{ then } y := x \text{ else } y := -x \{y \geq 0\}.$

The rule creates two branch triples:

$\{x \geq 0\} y := x \{y \geq 0\}$

and

$\{x < 0\} y := -x \{y \geq 0\}.$

Then branch

Assignment requires the precondition

$$x \geq 0.$$

Therefore

$$\vdash \{x \geq 0\} y := x \{y \geq 0\}$$

follows directly from *Assign*.

Else branch

Assignment requires

$$-x \geq 0.$$

The branch assumption is $x < 0$, so consequence requires the valid implication

$$x < 0 \rightarrow -x \geq 0.$$

This proves the else-branch triple.

Completed conditional proof

Both premises of If are derivable:

$$\vdash \{x \geq 0\} y := x \{y \geq 0\},$$

$$\vdash \{x < 0\} y := -x \{y \geq 0\}.$$

Hence the original triple is derivable for every initial integer value of x .

The proof followed the syntax of the command. Its intermediate assertions and logical side conditions were determined systematically.

Soundness and relative completeness

Soundness:

$$\vdash \{P\}S\{Q\} \quad \Rightarrow \quad \models \{P\}S\{Q\}.$$

For the loop-free language, Hoare logic is relatively complete when the assertion language can express the needed intermediate assertions and all valid logical side conditions can be proved.

Manual derivations expose the structure of the proof. The next lecture uses these rules to generate proof obligations mechanically.

Why is Hoare logic a logic?

We can write preconditions and postconditions without Hoare logic. Hoare logic provides a system for proving that commands satisfy them.

It has the usual parts of a logic:

- **Judgments:** Hoare triples such as $\{P\} S \{Q\}$
- **Semantics:** a definition of when a triple is valid
- **Inference rules:** ways to derive valid triples from simpler ones

The assertions P and Q describe states. Hoare logic connects those assertions to program behavior.

Its proof rules let us reason about a program from its structure.

Next: verification conditions and loop invariants

How can the structural steps in a Hoare proof be calculated from a program and its specification?

The next lecture uses weakest preconditions to generate verification conditions, then studies the invariants needed for loops.