Programming Language Semantics and Compiler Design

(Sémantique des Langages de Programmation et Compilation)

Axiomatic Semantics - Hoare Logic

Frédéric Lang & Laurent Mounier
firstname.lastname@univ-grenoble-alpes.fr
Univ. Grenoble Alpes, Inria,
Laboratoire d'Informatique de Grenoble & Verimag

Master of Sciences in Informatics at Grenoble (MoSIG)
Master 1 info

Univ. Grenoble Alpes - UFR IM^{2}AG
www.univ-grenoble-alpes.fr — im2ag.univ-grenoble-alpes.fr

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

Axiomatic Semantics for Total Correctness

Summary

Outline - Axiomatic Semantics - Hoare Logic

- Introduction
  - Partial Correctness and Total Correctness
  - Verifying Properties with NOS
- Axiomatic Semantics for Partial Correctness
- Axiomatic Semantics for Total Correctness
- Summary

Outline - Axiomatic Semantics - Hoare Logic

- Introduction
  - Partial Correctness and Total Correctness
  - Verifying Properties with NOS
- Axiomatic Semantics for Partial Correctness
- Axiomatic Semantics for Total Correctness
- Summary

Partial correctness and total correctness

Goal: specify an " **input** / **output** " relationship that the program must satisfy.

Example 1 (Program Fact)

$$y := 1;$$
$$\text{while } \neg(x = 1) \text{ do } y := y * x; x := x - 1 \text{ od}$$

- **Partial Correctness:**
  If the initial value of x is $n > 0$ and if the program terminates **then** the final value of y is $n!$
- **Total Correctness:**
  If the initial value of x is $n > 0$ **then** the program terminates and the final value of y is $n!$

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Partial Correctness and Total Correctness

Verifying Properties with NOS

Axiomatic Semantics for Partial Correctness

Axiomatic Semantics for Total Correctness

Summary

Verifying semantic properties - a motivating example

How can we prove the (partial) correctness of Fact using NOS?

```
Fact:
y := 1;
while ¬(x = 1) do y := y ∗ x; x := x − 1 od
```

Formalization:

Let n be the initial value of x

$$n > 0 \text{ and } \sigma_0(x) = n \text{ and } (\text{Fact}, \sigma_0) \to \sigma' \text{ implies } \sigma'(y) = n!$$

Stage 0 Correctness of the initialization.

Stage 1 Correctness of the loop body.

Stage 2 Correctness of the loop.

Stage 3 Correctness of the program.

$\hookrightarrow$ Study of the derivation tree.

Verifying semantic properties - a motivating example (continued)

Correctness of Fact

Use of a loop invariant $I(\sigma) \overset{\text{def}}{=} \sigma(y) = \dfrac{n!}{\sigma(x)!}$

Stage 0 The initialization $y := 1$ satisfies for all σ, σ':

$$\text{if } n > 0 \text{ and } \sigma(x) = n \text{ and } (y := 1, \sigma) \to \sigma' \text{ then } I(\sigma') \text{ and } \sigma'(x) > 0$$

Stage 1 The loop body satisfies for all σ, σ':

$$\text{if } I(\sigma) \text{ and } \sigma(x) > 0 \text{ and } (y := y * x; x := x - 1, \sigma) \to \sigma' \text{ then } I(\sigma')$$

Stage 2 The loop satisfies for all σ, σ':

$$\begin{array}{rl} \text{if} & I(\sigma) \text{ and } \sigma(x) > 0 \text{ and } (\text{while } \neg(x = 1) \text{ do } \dots \text{ od}, \sigma) \to \sigma' \\ \text{then} & I(\sigma') \text{ and } \sigma'(x) = 1 \end{array}$$

Note that $I(\sigma')$ and $\sigma'(x) = 1$ implies $\sigma'(y) = n!$

Stage 3 Partial correctness of the program:

$$\text{if } n > 0 \text{ and } \sigma(x) = n \text{ and } (\text{Fact}, \sigma) \to \sigma' \text{ then } \sigma'(y) = n!$$

Lessons learned from the motivating example

Proving semantic properties about programs could be done using OS.

But: This does not scale:

- tedious,
- long,
- not practical,
- too closely connected to the operational semantics (and how computation is performed)

We want to focus on the essential properties we want to prove.

We will exhibit the essential properties of the language constructs.

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

- Hoare Triples
- The Assertion Language
- The inference system - Hoare Calculus
- Properties of the Semantics: soundness and completeness
- Building proofs automatically?
- Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

- Hoare Triples
- The Assertion Language
- The inference system - Hoare Calculus
- Properties of the Semantics: soundness and completeness
- Building proofs automatically?
- Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

Hoare Triple

Idea: specify properties of programs as relations between properties of its inputs and properties of its outputs via Hoare triples.

Definition 1 (Hoare Triple)

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

- S a statement
- P an assertion, called precondition
- Q an assertion, called postcondition

Meaning:

if P holds in the initial state (before executing S)
and the execution of S on that state terminates,
then Q will hold in the state in which S terminates.

If we can prove this, we say that $\{P\}\ S\ \{Q\}$ holds.

Remark It is not necessary that S terminates for $\{P\}\ S\ \{Q\}$ to hold (we will see examples). □

Hoare Triple on a example

Example 2 (Hoare triple for Fact)

$\{x = n \wedge n > 0\}$
$y := 1;$
while $\neg(x = 1)$ do $y := y * x; x := x - 1$ od
$\{y = n!\}$

- Precondition: $\{x = n \wedge n > 0\}$
- Post-condition: $\{y = n!\}$.

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

- Hoare Triples
- The Assertion Language
- The inference system - Hoare Calculus
- Properties of the Semantics: soundness and completeness
- Building proofs automatically?
- Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

The assertion language

Allows to express pre and postconditions.

Example 3 (Program Fact)

$$\begin{array}{l}\{x = n \wedge n > 0\} \\ y := 1; \\ \text{while } \neg(x = 1) \text{ do } y := y * x; x := x - 1 \text{ od} \\ \{y = n!\}\end{array}$$

Remark Specifying an "input/output" relation, we could not replace $\{y = n!\}$ by $\{y = x!\}$ □

n is a logical variable:

- does not appear in the program,
- used to "remember the initial values of program variables".

Two kinds of variables:

- program variables (**Var**),
- logical variables.

The assertion language: predicates

Intuition: a Boolean expression b defines a predicate $\mathcal{B}[b]$: **State** $\mapsto \{\mathbf{tt}, \mathbf{ff}\}$

Definition 2 (Predicate)

A predicate is a partial function from **State** to $\{\mathbf{tt}, \mathbf{ff}\}$ denoted using the syntactic category **Bexp** extended with logical variables.

Notation for predicates

For a predicate P:

- we write $P(\sigma) \in \{\mathbf{tt}, \mathbf{ff}\}$ (instead of $\mathcal{B}[P](\sigma)$) for the evaluation of P on σ;
- when $P(\sigma) = \mathbf{tt}$, we say that P holds on σ.

Example 4 (Predicate)

- $P_1 \overset{\text{def}}{=} (x = n)$
- $P_2 \overset{\text{def}}{=} (n > 0 \wedge x = n!)$
- $P_3 \overset{\text{def}}{=} (x = n \wedge y = n!)$
- $P_1(\sigma_1) = \mathbf{tt}$
- $P_2(\sigma_2) = \mathbf{ff}$

- $\sigma_1 = [x \mapsto n]$
- $\sigma_2 = [x \mapsto n + 1]$
- $\sigma_3 = [x \mapsto 3, y \mapsto 6]$

$$P_3(\sigma_3) = \begin{cases} \mathbf{tt} & \text{if } n = 3 \\ \mathbf{ff} & \text{otherwise} \end{cases}$$

The assertion language: predicates (continued)

Reminder: The Boolean domain $\{\mathbf{tt}, \mathbf{ff}\}$ is endowed with usual Boolean operators, noted $\wedge$, $\vee$, $\neg$, and $\implies$, and their usual semantics.

Notations (Boolean operators on predicates)

For all predicates P_0, P_1, P_2 and all state $\sigma \in \mathbf{State}$:

- $P_1 \wedge P_2$ denotes the function defined by $(P_1 \wedge P_2)(\sigma) \overset{\text{def}}{=} P_1(\sigma) \wedge P_2(\sigma)$,
- $P_1 \vee P_2$ denotes the function defined by $(P_1 \vee P_2)(\sigma) \overset{\text{def}}{=} P_1(\sigma) \vee P_2(\sigma)$,
- $\neg P_0$ denotes the function defined by $(\neg P_0)(\sigma) \overset{\text{def}}{=} \neg(P_0(\sigma))$,
- $P_1 \implies P_2$ denotes the function defined by $(P_1 \implies P_2)(\sigma) \overset{\text{def}}{=} P_1(\sigma) \implies P_2(\sigma)$,

Example 5 (Predicate)

Recall that $P_1 \overset{\text{def}}{=} (x = n)$ and $P_2 \overset{\text{def}}{=} (x = n!)$:

$$(P_1 \wedge P_2)([x \mapsto 2]) = \begin{cases} \mathbf{tt} & \text{if } n = 2 \\ \mathbf{ff} & \text{otherwise} \end{cases}$$

If n is unknown and $P_1 \wedge P_2$ is assumed to hold, then it implies that $n = 2$.

The assertion language: logical equivalence and syntactic substitution

Definition 3 (Logical equivalence)

Two predicates P_1 and P_2 are logically equivalent iff for all σ such that $\text{vars}(P_1) \subseteq \text{dom}(\sigma)$ and $\text{vars}(P_2) \subseteq \text{dom}(\sigma)$, $P_1(\sigma) = P_2(\sigma)$.

Remark If P_1 and P_2 are logically equivalent, then they may be freely interchanged (e.g., predicate simplification). □

Definition 4 (Substitution)

For $x \in \mathbf{Var}$ and $a \in \mathbf{Aexp}$, $P[a/x]$ is a predicate obtained by replacing (syntactically) each occurrence of x by a in P.

Example 6 (Substitution)

Recall that $P_1 \overset{\text{def}}{=} (x = n)$:

- $P_1[y + 2/x] \overset{\text{def}}{=} (y + 2 = n)$

Remark Logical equivalence is closed under substitution: If P_1 and P_2 are logically equivalent, then for all $x \in \mathbf{Var}, a \in \mathbf{Aexp}$, $P_1[a/x]$ and $P_2[a/x]$ are logically equivalent. □

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

- Hoare Triples
- The Assertion Language
- The inference system - Hoare Calculus
- Properties of the Semantics: soundness and completeness
- Building proofs automatically?
- Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

The inference system - Hoare Calculus

Recall **logical derivation** with rules:

$$\frac{\mathit{Premisse}_1 \quad \ldots \quad \mathit{Premisse}_n}{\mathit{Conclusion}}$$

- forward interpretation/reading: if we have proved $\mathit{Premisse}_1$ and . . . and $\mathit{Premisse}_n$, then we have proved *Conclusion*;
- backward interpretation/reading: to prove *Conclusion*, it suffices to prove $\mathit{Premisse}_1$ and . . . and $\mathit{Premisse}_n$.

Partial correctness assertions will be specified by an inference system (axioms and rules) that will allow us to write inference trees.

Intuitively, an inference tree says how properties "propagate".

(Similar to derivation trees in Natural Operational Semantics).

Formulas are Hoare triples, of the form $\{P\}\ S\ \{Q\}$:

- $S \in$ **Stm**: a statement in language **While**.
- $\{P\}$ and $\{Q\}$ are predicates.

The inference system: skip

Definition 5 (Axiom of skip)

$$\{P\}\ \text{skip}\ \{P\}$$

- skip does not change the state;
- if P holds before the execution of skip, then it holds afterwards as well.

Example 7 (Applications of the skip axiom)

- $\{x > 0\}$ skip $\{x > 0\}$
- $\{\text{true}\}$ skip $\{\text{true}\}$
- $\{\text{false}\}$ skip $\{\text{false}\}$

The inference system: assignment

Definition 6 (Axiom of assignment)

$$\{P[a/x]\}\ x := a\ \{P\}$$

- $P[a/x]$ is P where every occurrence of x is replaced by a.
- For P to hold after the assignment of a to x, it suffices to show that P holds for a before the assignment.

The rules are “schemes” that:

- need to be instantiated for a particular choice of P, and
- have an implicit universal quantification on x, a and P.

Example 8 (Applications of the assignment axiom)

- $\{y > 42\}\ x := 20\ \{x > 0 \land y > x + 22\}$
  using an implicit simplification
  $(x > 0 \land y > x + 22)[20/x] = (20 > 0 \land y > 20 + 22) = (y > 42)$
- $\{z = 2 * x * y\}\ x := x * 2\ \{z = x * y\}$

The inference system: sequential composition

Definition 7 (Inference Rule for sequential composition)

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

- To show that $\{P\}\ S\ \{Q\}$, it suffices to find an intermediate predicate Q such that $\{P\}\ S_1\ \{Q\}$ and $\{Q\}\ S_2\ \{R\}$
- Q is the postcondition of S_1 and the precondition of S_2

The inference system: sequential composition (continued)

Rule for sequential composition

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

Example 9 (Inference rule for sequential composition)

$$\frac{\overbrace{\{x>0\}}^{P}\ \text{skip}\ \overbrace{\{x>0\}}^{Q} \quad \overbrace{\{x>0\}}^{Q}\ \text{skip}\ \overbrace{\{x>0\}}^{R}}{\underbrace{\{x>0\}}_{P}\ \text{skip};\text{skip}\ \underbrace{\{x>0\}}_{R}}$$

$$\frac{\begin{array}{c}\overbrace{\{z+2>20\wedge 20>z\}}^{P}\\ y:=20\\ \underbrace{\{z+2>y\wedge y>z\}}_{Q}\end{array} \quad \begin{array}{c}\overbrace{\{z+2>y\wedge y>z\}}^{Q}\\ x:=z+2\\ \underbrace{\{x>y\wedge y>z\}}_{R}\end{array}}{\underbrace{\{z+2>20\wedge 20>z\}}_{P}\ y:=20; x:=z+2\ \underbrace{\{x>y\wedge y>z\}}_{R}}$$

The inference system: if-then-else

Definition 8 (Inference rule for if-then-else)

$$\frac{\{b \land P\}\ S_1\ \{Q\} \qquad \{\neg b \land P\}\ S_2\ \{Q\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}\ \{Q\}}$$

To show that $\{P\}\ S\ \{Q\}$, it suffices to show that Q is a postcondition of each branch (S_1 and S_2), taking as precondition P and the condition for the branch to be executed (b and $\neg b$, respectively).

Remark "if b then S_1 fi" can be handled as "if b then S_1 else skip fi". □

The inference system: if-then-else

Rule for the if-then-else

$$\frac{\{b \land P\}\ S_1\ \{Q\} \qquad \{\neg b \land P\}\ S_2\ \{Q\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}\ \{Q\}}$$

Example 10 (Inference rule for if-then-else)

$$\frac{\begin{array}{c}\{\overbrace{x > 0}^{b} \land \overbrace{y > 0 \land x \neq 0}^{P}\}\\ \text{skip}\\ \underbrace{\{x > 0 \land y > 0 \land x \neq 0\}}_{Q}\end{array} \qquad \begin{array}{c}\{\overbrace{x \leq 0}^{\neg b} \land \overbrace{y > 0 \land -x \neq 0}^{P}\}\\ x := -x\\ \underbrace{\{x > 0 \land y > 0 \land x \neq 0\}}_{Q}\end{array}}{\underbrace{\{y > 0 \land x \neq 0\}}_{P}\ \text{if } x > 0 \text{ then skip else } x := -x \text{ fi}\ \underbrace{\{x > 0 \land y > 0 \land x \neq 0\}}_{Q}}$$

We use the following equivalence under the hood: $x \neq 0 \iff -x \neq 0$

The inference system: while loop

Definition 9 (Inference rule for while loop)

$$\frac{\{b \wedge P\}\ S\ \{P\}}{\{P\}\ \text{while } b \text{ do } S \text{ od } \{\neg b \wedge P\}}$$

P is a loop invariant, which must hold before and after each execution of the loop-body.

Example 11 (Inference rule for while loop)

$$\dfrac{\dfrac{\begin{array}{c}\{x > 0 \wedge I\}\\ x := x - 1\\ \{x \geq 0 \wedge y + z = z * (x_0 - x) \wedge z = z_0\}\end{array} \qquad \begin{array}{c}\{x \geq 0 \wedge y + z = z * (x_0 - x) \wedge z = z_0\}\\ y := y + z\\ \{I\}\end{array}}{\{x > 0 \wedge I\}\ x := x - 1; y := y + z\ \{I\}}}{\{I\}\ \text{while } x > 0 \text{ do } x := x - 1; y := y + z \text{ od } \{x \leq 0 \wedge I\}}$$

where $I \overset{\text{def}}{=} x \geq 0 \wedge y = z * (x_0 - x) \wedge z = z_0$

Inference system

We now have a rule for each statement of **While**.
Is it sufficient?

Exercise 1

Prove $\{x > 0\}\ x := x + 1\ \{x > 0\}$.

This is not possible so far...
Using the axiom of assignment, we can only build either proof tree:

$$\frac{}{\{x > 0\}\ x := x + 1\ \{x > 1\}} \quad \text{or} \quad \frac{}{\{x \geq 0\}\ x := x + 1\ \{x > 0\}}$$

Can we replace the postcondition $x > 1$ of the first Hoare triple or the precondition $x \geq 0$ of the second Hoare triple by $x > 0$?

Yes, because $x > 1 \implies x > 0$ and $x > 0 \implies x \geq 0$.
This must be allowed by a rule: the rule of consequence.

Stronger/weaker predicate

If $P \implies Q$, we say that:

- P is stronger than Q (i.e., P sets more constraints on the state than Q)
- Q is weaker than P.

Reminder: $P \implies Q$ means "if P holds then Q holds". If P does not hold, then it sets no constraint on the state. It is equivalent to $\neg P \vee Q$.

For all P, we have both:

- $P \implies \text{true}$
  true is the weakest predicate, setting no constraint at all
- $\text{false} \implies P$
  false is the strongest predicate, setting a constraint that cannot be fulfilled

Example 12

$x > 0$ is stronger than $x \geq 0$, because if if $x > 0$ then necessarily $x \geq 0$.
Neither of $x > 0$ and $x < 3$ is stronger or weaker than the other.

The inference system: consequence

The following rule allows the precondition to be weakened and/or the postcondition to be strengthened.

Definition 10 (Inference rule for consequence)

If $P \implies P'$ and $Q' \implies Q$, then:

$$\frac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$$

Example 13 (Inference rule for consequence)

$$\frac{\{x \geq 0\}\ y := x + 1\ \{y > 0\}}{\{x > 42\}\ y := x + 1\ \{y \neq 0\}}$$

Building proofs using Hoare logic

Remark

- In the NOS of **While**, the derivation tree built to achieve a given goal $(S, \sigma) \to \sigma'$ is unique, because at each step at most one inference rule applies, determined by the program syntax and the value of Boolean conditions.
- Building proofs using Hoare logic is less deterministic, as there may exist several proofs of the same Hoare triple. For instance, the rule of consequence is not guided by the program syntax.

□

Example 14

Here are two proofs of the same Hoare triple $\{x > 0\}\ x := x + 1\ \{x > 0\}$:

$$\frac{\{x > 0\}\ x := x + 1\ \{x > 1\}}{\{x > 0\}\ x := x + 1\ \{x > 0\}} \qquad \frac{\{x \geq 0\}\ x := x + 1\ \{x > 0\}}{\{x > 0\}\ x := x + 1\ \{x > 0\}}$$

The complete inference system

Rule name	**rule**
Skip	$\{P\}\ \text{skip}\ \{P\}$
Assignment	$\{P[a/x]\}\ x := a\ \{P\}$
Sequential	$\dfrac{\{P\}\ S_1\ \{Q\} \qquad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$
Conditional	$\dfrac{\{b \wedge P\}\ S_1\ \{Q\} \qquad \{\neg b \wedge P\}\ S_2\ \{Q\}}{\{P\}\ \text{if}\ b\ \text{then}\ S_1\ \text{else}\ S_2\ \text{fi}\ \{Q\}}$
Iterative	$\dfrac{\{b \wedge P\}\ S\ \{P\}}{\{P\}\ \text{while}\ b\ \text{do}\ S\ \text{od}\ \{\neg b \wedge P\}}$
Consequence	If $P \Rightarrow P'$ and $Q' \Rightarrow Q$, then: $\dfrac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$

Comparison with Natural Operational Semantics

We have defined a set of rules and axioms.

Natural Operational Semantics	Axiomatic Semantics
Axioms	Axioms
Inference rules	Inference rules
Derivation trees = description/proof of a computation expressed at the root	Inference trees = proof of a property expressed at the root
Leaves = Instance of axioms	Leaves = instance of axioms
Internal nodes = instances of rules	Internal nodes = instances of rules

Proving properties using the inference system

An inference tree gives a proof of the property expressed at its root.

Notation

When inferring $\{P\}\ S\ \{Q\}$ (with rules and axioms), we note:

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

Remark Writing $\vdash_p$ *assertion* says that we can deduce *assertion* with the rules, axioms, and other assertions. □

Example 15 (Proving properties)

- $\vdash_p \{x = 0\}\ x := x + 1; x := x + 1\ \{x = 2\}$
- $\vdash_p \{x > 0\}\ y := 1\ \{x = x * y\}$
- $\vdash_p \{\text{true}\}$ while true do skip od $\{\text{true}\}$

Exercises: a solution

Proof of $\vdash_p \{x = 0\}\ x := x + 1; x := x + 1\ \{x = 2\}$

$$\dfrac{\dfrac{}{\{x = 0\}\ x := x + 1\ \{x = 1\}}\,[\mathtt{ass}] \qquad \dfrac{}{\{x = 1\}\ x := x + 1\ \{x = 2\}}\,[\mathtt{ass}]}{\{x = 0\}\ x := x + 1; x := x + 1\ \{x = 2\}}\,[\mathtt{comp}]$$

because $(x = 1)[x + 1/x] = (x + 1 = 1) = (x = 0)$
$(x = 2)[x + 1/x] = (x + 1 = 2) = (x = 1)$

Proof of $\vdash_p \{x > 0\}\ y := 1\ \{x = x * y\}$

$$\dfrac{\dfrac{}{\{\text{true}\}\ y := 1\ \{x = x * y\}}\,[\mathtt{ass}]}{\{x > 0\}\ y := 1\ \{x = x * y\}}\,[\mathtt{conseq}]$$

because $(x = x * y)[1/y] = (x = x) = \text{true}$
$x > 0 \implies \text{true}$

Exercises: a solution

Proof of $\vdash_p \{\text{true}\}$ while true do skip od $\{\text{true}\}$

$$\dfrac{\dfrac{\dfrac{}{\{\text{true}\}\ \text{skip}\ \{\text{true}\}}\,[\mathtt{skip}]}{\{\text{true}\}\ \text{while true do skip od}\ \{\text{false}\}}\,[\mathtt{while}]}{\{\text{true}\}\ \text{while true do skip od}\ \{\text{true}\}}\,[\mathtt{conseq}]$$

because $\neg\text{true} \wedge \text{true} = \text{false}$
$\text{false} \implies \text{true}$

Remark If $\{P\}\ S\ \{Q\}$ and S is started in a state satisfying P, we **cannot** claim that S will terminate in a state satisfying Q □

Example of the factorial program

$$y := 1; \text{while } \neg(x = 1) \text{ do } y := y * x; x := x - 1 \text{ od}$$

Let n be the initial value of x, $S_0 \overset{\text{def}}{=} y := 1$, $S_1 \overset{\text{def}}{=} y := y * x; x := x - 1$, and the invariant $I \overset{\text{def}}{=} x > 0 \land y = \dfrac{n!}{x!}$.

$$\dfrac{\dfrac{\dfrac{}{\{x > 0 \land x! = n!\}\ S_0\ \{I\}}\ (3)}{\{x > 0 \land x = n\}\ S_0\ \{I\}}\ (2) \qquad \dfrac{\text{continued below}}{\{I\}\ \text{while } \neg(x = 1) \text{ do } S_1 \text{ od}\ \{y = n!\}}}{\{x > 0 \land x = n\}\ S_0; \text{while } \neg(x = 1) \text{ do } S_1 \text{ od}\ \{y = n!\}}\ (1)$$

Continued:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{I \land \neg(x = 1)\}\ y := y * x\ \{I[x - 1/x]\}}\ (7) \qquad \dfrac{}{\{I[x - 1/x]\}\ x := x - 1\ \{I\}}\ (8)}{\{I \land \neg(x = 1)\}\ S_1\ \{I\}}\ (6)}{\{I\}\ \text{while } \neg(x = 1) \text{ do } S_1 \text{ od}\ \{x = 1 \land y = n!\}}\ (5)}{\{I\}\ \text{while } \neg(x = 1) \text{ do } S_1 \text{ od}\ \{y = n!\}}\ (4)$$

Example of the factorial program (continued)

Justifications of the previous inference tree:

(1) Sequential composition.

(2) Consequence: $x > 0 \land x = n \implies x > 0 \land x! = n!$.

(3) Assignment: $I[1/y]$ is equivalent to $x > 0 \land x! = n!$.

(4) Consequence: $x = 1 \land y = n! \implies y = n!$.

(5) While loop: $I \land \neg\neg(x = 1)$ is equivalent to $x = 1 \land y = n!$.

(6) Sequential composition.

(7) Assignment: $I[x - 1/x][y * x/y]$ is equal to $x - 1 > 0 \land y * x = \dfrac{n!}{(x - 1)!}$, which is equivalent to $I \land \neg(x = 1)$.

(8) Assignment.

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

- Hoare Triples
- The Assertion Language
- The inference system - Hoare Calculus
- Properties of the Semantics: soundness and completeness
- Building proofs automatically?
- Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

Properties of the semantics

Definition 11 (Semantic equivalence between programs)

S_1 and S_2 are provably equivalent according to the axiomatic semantics (for partial correctness) if

- for all preconditions P,
- for all postconditions Q:

$$\vdash_p \{P\}\ S_1\ \{Q\} \text{ iff } \vdash_p \{P\}\ S_2\ \{Q\}$$

Soundness and completeness of Hoare logic for partial correctness

Definition 12 (Validity of a Hoare triple)

Triple $\{P\}\ S\ \{Q\}$ is valid, noted

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

iff for all states $\sigma, \sigma' \in \textbf{State}$:

- if $P(\sigma)$ holds and $(S, \sigma) \to \sigma'$
- then $Q(\sigma')$ holds.

We say that S is partially correct wrt. P and Q.

Soundness (We can infer only valid triples)

$$\text{If } \vdash_p \{P\}\ S\ \{Q\} \text{ then } \models_p \{P\}\ S\ \{Q\}$$

Completeness (We can infer all valid triples)

$$\text{If } \models_p \{P\}\ S\ \{Q\} \text{ then } \vdash_p \{P\}\ S\ \{Q\}$$

Soundness of Hoare logic

Let us consider some statement $S \in \textbf{Stm}$, some assertions P and Q, and suppose $\vdash_p \{P\}\ S\ \{Q\}$. We shall prove $\models_p \{P\}\ S\ \{Q\}$

We conduct a proof by induction on the shape of the inference tree to get $\vdash_p \{P\}\ S\ \{Q\}$, that is to infer $\{P\}\ S\ \{Q\}$ with the inference system.

Let us consider some states σ, σ' and suppose $(S, \sigma) \to \sigma'$ and $P(\sigma)$.

[ass] Necessarily S is of the form $x := a$ and $P[a/x](\sigma) = \textbf{tt}$.
[ass$^{\text{nos}}$] gives $\sigma' = \sigma[x \mapsto \mathcal{A}[a]\sigma]$.
$P(\sigma') = \textbf{tt}$ (from correctness of substitution – see the tutorial exercise).

[skip] Straightforward.

[comp] Necessarily S is of the form $S_1; S_2$, and $P(\sigma) = \textbf{tt}$.
Assume $\models_p \{P\}\ S_1\ \{R\}$ and $\models_p \{R\}\ S_2\ \{Q\}$ (I.H.).
[comp$^{\text{nos}}$] gives $(\exists \sigma'' \in \textbf{State})\ (S_1, \sigma) \to \sigma''$ and $(S_2, \sigma'') \to \sigma'$.
From $(S_1, \sigma) \to \sigma''$, $\models_p \{P\}\ S_1\ \{R\}$, and $P(\sigma) = \textbf{tt}$, we get $R(\sigma'') = \textbf{tt}$.
From $(S_2, \sigma'') \to \sigma'$, $\models_p \{R\}\ S_2\ \{Q\}$, and $R(\sigma'') = \textbf{tt}$, we get $Q(\sigma') = \textbf{tt}$.

[if] Necessarily S is of the form if b then S_1 else S_2 fi.
Assume $\models_p \{b \wedge P\}\ S_1\ \{Q\}$ and $\models_p \{\neg b \wedge P\}\ S_2\ \{Q\}$ (I.H.).

- Case $\mathcal{B}[b]\sigma = \textbf{tt}$. Then, following the semantics of $\wedge$: $(P \wedge b)(\sigma) = \textbf{tt}$.
  [if$_{\text{nos}}$] gives $(S_1, \sigma) \to \sigma'$.
  $\models_p \{b \wedge P\}\ S_1\ \{Q\}$ gives $Q(\sigma') = \textbf{tt}$.
- Case $\mathcal{B}[b]\sigma = \textbf{ff}$. Similar.

Soundness of Hoare logic

Proof by induction on the shape of the inference tree to infer $\{P\}\ S\ \{Q\}$

[while] Necessarily, S is of the form while b do S' od.
Assume $\models \{b \wedge P\}\ S'\ \{P\}$ (I. H.)
(We want to prove $\models \{P\}$ while b do S' od $\{\neg b \wedge P\}$)

- Case $\mathcal{B}[b]\sigma = \mathbf{tt}$, then $(S', \sigma) \to \sigma'$ and (while b do S' od$, \sigma') \to \sigma''$
  $(b \wedge P)(\sigma) = \mathbf{tt}$ and $\models \{b \wedge P\}\ S'\ \{P\}$ gives $P(\sigma') = \mathbf{tt}$.
  IH on (while b do S' od$, \sigma') \to \sigma''$ gives $(\neg b \wedge P)(\sigma'') = \mathbf{tt}$.
- Case $\mathcal{B}[b]\sigma = \mathbf{ff}$, then $\sigma' = \sigma''$ and $(\neg b \wedge P)(\sigma'') = \mathbf{tt}$.

[cons] Suppose $\models \{P'\}\ S\ \{Q'\}$, $P \Rightarrow P'$ and $Q' \Rightarrow Q$.
(Recall that we assume $(S, \sigma) \to \sigma'$ and $P(\sigma) = \mathbf{tt}$).
From $P(\sigma) = \mathbf{tt}$ and $P \Rightarrow P'$, we get $P'(\sigma)$.
From $P'(\sigma) = \mathbf{tt}$ and $\models_p \{P'\}\ S\ \{Q'\}$, we get $Q'(\sigma')$.
From $Q'(\sigma') = \mathbf{tt}$ and $Q' \Rightarrow Q$, we get $Q(\sigma')$.

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

Hoare Triples

The Assertion Language

The inference system - Hoare Calculus

Properties of the Semantics: soundness and completeness

Building proofs automatically?

Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

Building proofs using the weakest-precondition calculus

For a statement S without while loop, if $\models_p \{P\}\ S\ \{Q\}$, then a proof can be built systematically using a calculus called the weakest-precondition calculus (wp).

Given the postcondition Q and the loop-free statement S, this calculus computes the weakest precondition $wp(S, Q)$ such that $\models_p \{wp(S, Q)\}\ S\ \{Q\}$. This precondition is called weakest precondition because $\models_p \{P\}\ S\ \{Q\}$ holds if and only if $P \Rightarrow wp(S, Q)$.

Definition 13 (Weakest precondition calculus for loop-free statements)

$$\begin{array}{rcl}
wp(\text{skip}, Q) & = & Q \\
wp(x := a, Q) & = & Q[a/x] \\
wp(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, Q) & = & (b \Rightarrow wp(S_1, Q)) \wedge (\neg b \Rightarrow wp(S_2, Q)) \\
wp(S_1; S_2, Q) & = & wp(S_1, wp(S_2, Q))
\end{array}$$

We shall prove that $\vdash_p \{wp(S, Q)\}\ S\ \{Q\}$, i.e., $wp(S, Q)$ is indeed a valid precondition. This proof is constructive, in the sense that we can deduce from it an algorithm that actually builds a proof tree. The proof that it is the weakest precondition is out of the scope of this lecture.

From wp to proof trees: skip and assignment

Let S be any loop-free statement. We show by induction on S that

$$(\forall Q)\ \vdash_p \{wp(S, Q)\}\ S\ \{Q\}$$

Skip

$$wp(\text{skip}, Q) = Q$$

The proof tree is a valid instance of the axiom of skip:

$$\frac{}{\{Q\}\ \text{skip}\ \{Q\}}$$

Assignment

$$wp(x := a, Q) = Q[a/x]$$

The proof tree is a valid instance of the axiom of assignment:

$$\frac{}{\{Q[a/x]\}\ x := a\ \{Q\}}$$

From wp to proof trees: sequential composition

$$wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$$

Let $R = wp(S_2, Q)$ and $P = wp(S_1, R)$. The proof tree is an instance of the inference rule for sequential composition:

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

The premises have valid proof trees from the induction hypothesis: $\vdash_p \{\underbrace{wp(S_1, R)}_{P}\}\ S_1\ \{R\}$ and $\vdash_p \{\underbrace{wp(S_2, Q)}_{R}\}\ S_2\ \{Q\}$.

From wp to proof trees: if-then-else

$$wp(\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}, Q) = (b \Rightarrow wp(S_1, Q)) \land (\neg b \Rightarrow wp(S_2, Q))$$

Let $P = (b \Rightarrow wp(S_1, Q)) \land (\neg b \Rightarrow wp(S_2, Q))$. The proof tree is an instance of the inference rule for if-then-else:

$$\frac{\{b \land P\}\ S_1\ \{Q\} \quad \{\neg b \land P\}\ S_1\ \{Q\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}\ \{Q\}}$$

The premises have valid proof trees because:

- $b \land P$ is equivalent to $wp(S_1, Q)$ and by induction hypothesis, $\vdash_p \{wp(S_1, Q)\}\ S_1\ \{Q\}$.
- $\neg b \land P$ is equivalent to $wp(S_2, Q)$ and by induction hypothesis, $\vdash_p \{wp(S_2, Q)\}\ S_2\ \{Q\}$.

Automatic proofs for loop-free programs

Theorem 1

If $\models_p \{P\}\ S\ \{Q\}$ and S does not contain while loops, then a proof tree of $\{P\}\ S\ \{Q\}$ can be built automatically.

Proof.

We can build a proof tree for $\{wp(S,Q)\}\ S\ \{Q\}$ automatically, as described in the above proof. Since Hoare logic is sound, this means that $\models_p \{wp(S,Q)\}\ S\ \{Q\}$.

Since $\models_p \{P\}\ S\ \{Q\}$, we have $P \Rightarrow wp(S,Q)$. Therefore a proof of $\{P\}\ S\ \{Q\}$ can be given using an instance of the rule of consequence:

$$\frac{\{wp(S,Q)\}\ S\ \{Q\}}{\{P\}\ S\ \{Q\}}$$

□

Building proofs for programs with loops

Weakest precondition is not computable in general for while loops. Approximations can be computed if a correct invariant is provided by the user together with each while loop of the program.

Finding an appropriate loop invariant I to prove a goal of the form $\{P\}$ while b do S od $\{Q\}$ may be difficult as it must simultaneously satisfy:

- I must be strong enough to imply the postcondition: $\neg b \wedge I \Rightarrow Q$
- I must hold initially: $P \Rightarrow I$
- I must hold when completing any iteration, under the assumption that it holds when starting the iteration: $\{b \wedge I\}\ S\ \{I\}$

Unfortunately, there is no general recipe to find an appropriate invariant. Proofs for programs with while loops cannot be built automatically in general.

Hints to find an appropriate invariant

Remind invariants from your courses on algorithmics...
If the program is correct, the invariant exists.

Hints:

- Observe how variables evolve after a few iterations.
- Find a predicate giving the value of those variables, for instance in function of the number n of iterations.
- Find an expression defining n in terms of the program variables and use it in the predicate defined in previous step.
- Try to derive a proof tree using the resulting predicate as invariant.
- If it fails, analyze what is wrong:
  - The invariant may have to be strengthen (e.g., adding a sign condition) if its conjunction with the halting condition does not imply the postcondition.
  - The invariant may have to be weakened if it is not implied by the precondition.
- If you cannot derive a proof, the precondition, postcondition, and/or program may be incorrect! Find a counterexample.

Example: finding the invariant

Goal:
$\{x = a \wedge a \geq 0 \wedge y = 1\}$ while $x > 0$ do $y := 2 * y; x := x - 1$ od $\{y = 2^a\}$

iteration	value of x	value of y
0	$x_0 = a$	$y_0 = 1$
1	$x_1 = x_0 - 1 = a - 1$	$y_1 = 2 * y_0 = 2 * 1$
2	$x_2 = x_1 - 1 = a - 2$	$y_2 = 2 * y_1 = 2 * 2 * 1$
3	$x_3 = x_2 - 1 = a - 3$	$y_3 = 2 * y_2 = 2 * 2 * 2 * 1$
...	...	...

- After n iterations, $x = a - n$ and $y = 2^n$.
- From $x = a - n$, we get $n = a - x$. Therefore, $y = 2^{a-x}$.
- Checking the precondition: $x = a$ and $y = 1$ implies $y = 2^{a-x}$ as $2^{a-x} = 2^{a-a} = 2^0 = 1 = y$.
- Checking the postcondition: $x \leq 0$ (on halting) and $y = 2^{a-x}$ do not imply the postcondition $y = 2^a$. Must add constraint $x \geq 0$.

We can now finish the proof using the invariant $I \equiv y = 2^{a-x} \wedge x \geq 0$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{y = 2^{a-x} \wedge x > 0\}\ y := 2 * y\ \{y = 2^{a-x+1} \wedge x > 0\}}}{\{y = 2^{a-x} \wedge x > 0\}\ y := 2 * y\ \{y = 2^{a-x+1} \wedge x \geq 0\}} \qquad \dfrac{}{\{y = 2^{a-x+1} \wedge x > 0\}\ x := x - 1\ \{I\}}}{\{I \wedge x > 0\}\ y := 2 * y; x := x - 1\ \{I\}}}{\{x = a \wedge x \geq 0 \wedge y = 1\}\ \text{while}\ x > 0\ \text{do}\ y := 2 * y; x := x - 1\ \text{od}\ \{y = 2^a\}}$$

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

- Hoare Triples
- The Assertion Language
- The inference system - Hoare Calculus
- Properties of the Semantics: soundness and completeness
- Building proofs automatically?
- Partial correctness and procedures

Axiomatic Semantics for Total Correctness

Summary

Hoare logic for partial correctness with procedures

Using procedures from **pProc**:

- Of the form proc $F(y_1, \ldots, y_m, ?z_1, \ldots, ?z_p)$ is var $x_1, \ldots, x_n$ in S_F end
- S_F may only use the local variables $x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_p$

To simplify the presentation, we also assume here that:

- S_F does not modify the input parameters $y_1, \ldots, y_m$
- In a call of the form $F(a_1, \ldots, a_m, ?w_1, \ldots, ?w_p)$, $a_1, \ldots, a_m$ do not contain occurrences of $w_1, \ldots, w_p$
  Example: $F(x + 1, ?x)$ is not allowed, but $F(x + 1, ?x'); x := x'$ is

Proving programs with procedures requires to attach a contract to each procedure F: its precondition P_F and its postcondition Q_F satisfying:

- $\text{vars}(P_F) \cap \{x_1, \ldots, x_n, z_1, \ldots, z_p\} = \emptyset$
- $\text{vars}(Q_F) \cap \{x_1, \ldots, x_n\} = \emptyset$

Then, the goal $\{P_F\}\ S_F\ \{Q_F\}$ must be proven for each procedure F.
This requires an inference rule for procedure call.

Inference rule for procedure call

Definition 14 (Rule of procedure call)

- Def: proc $F(y_1, \ldots, y_m, ?z_1, \ldots, ?z_p)$ is var $x_1, \ldots, x_n$ in S_F end
- Contract: $\{P_F\}\ S_F\ \{Q_F\}$
- Call: $F(a_1, \ldots, a_m, ?w_1, \ldots, ?w_p)$

Inference rule:

$$\frac{}{\{R \wedge P'_F\}\ F(a_1, \ldots, a_m, ?w_1, \ldots, ?w_p)\ \{R \wedge Q'_F\}}$$

where:

- $P'_F = P_F[a_1/y_1, \ldots, a_m/y_m]$
- $Q'_F = Q_F[a_1/y_1, \ldots, a_m/y_m, w_1/z_1, \ldots, w_p/z_p]$
- R is any predicate such that $\{w_1, \ldots, w_p\} \cap \mathrm{vars}(R) = \emptyset$, allowing constraints unaffected by the call to occur in both pre and postcondition.

Remark The proof of a statement S is valid only if a proof can be built for the contract of each procedure called in S. □

First example of proof with procedure call

Procedure:

$$\text{proc incr}(x, ?y) \text{ is } y := x + 1 \text{ end}$$

with contract:

$$\{\underbrace{\mathrm{true}}_{P_{\mathrm{incr}}}\}\ y := x + 1\ \{\underbrace{y = x + 1}_{Q_{\mathrm{incr}}}\}$$

We prove $\{x = 0 \wedge y = 0\}\ \mathrm{incr}(x, ?x'); \mathrm{incr}(y, ?y')\ \{x' = 1 \wedge y' = 1\}$.

$$\frac{T_1 \quad T_2}{\{x = 0 \wedge y = 0\}\ \mathrm{incr}(x, ?x'); \mathrm{incr}(y, ?y')\ \{x' = 1 \wedge y' = 1\}}$$

where T_1 is the following tree (consequence + procedure call):

$$\frac{\dfrac{}{\{\underbrace{x = 0 \wedge y = 0}_{R} \wedge \underbrace{\mathrm{true}}_{P'_{\mathrm{incr}}}\}\ \mathrm{incr}(x, ?x')\ \{\underbrace{x = 0 \wedge y = 0}_{R} \wedge \underbrace{x' = x + 1}_{Q_{\mathrm{incr}'}}\}}}{\{x = 0 \wedge y = 0\}\ \mathrm{incr}(x, ?x')\ \{x' = 1 \wedge y = 0\}}$$

and T_2 is the following similar tree:

$$\frac{\dfrac{}{\{\underbrace{x' = 1 \wedge y = 0}_{R} \wedge \underbrace{\mathrm{true}}_{P'_{\mathrm{incr}}}\}\ \mathrm{incr}(y, ?y')\ \{\underbrace{x' = 1 \wedge y = 0}_{R} \wedge \underbrace{y' = y + 1}_{Q_{\mathrm{incr}'}}\}}}{\{x' = 1 \wedge y = 0\}\ \mathrm{incr}(y, ?y')\ \{x' = 1 \wedge y' = 1\}}$$

Question: Why is this proof valid?

Second example of proof with procedure call

Procedure:

proc $\text{fact}(x, ?y)$ is if $x = 0$ then $y := 1$ else $\text{fact}(x-1, ?y); y := y * x$ fi end

with contract:

$$\{\underbrace{x \geq 0}_{P_{\text{fact}}}\} \text{ if } x = 0 \text{ then } y := 1 \text{ else fact}(x-1, ?y); y := y * x \text{ fi } \{\underbrace{y = x!}_{Q_{\text{fact}}}\}$$

We build a proof of the contract:

$$\dfrac{\dfrac{\dfrac{}{\{x! = 1\}\ y := 1\ \{y = x!\}}}{\{x = 0\}\ y := 1\ \{y = x!\}} \qquad \dfrac{T_0 \qquad \dfrac{\dfrac{}{\{x \neq 0 \wedge y = (x-1)!\}\ y := y * x\ \{x \neq 0 \wedge y = x!\}}}{\{x \neq 0 \wedge y = (x-1)!\}\ y := y * x\ \{y = x!\}}}{\{x > 0\}\ \text{fact}(x-1, ?y); y := y * x\ \{y = x!\}}}{\{x \geq 0\}\ \text{if } x = 0 \text{ then } y := 1 \text{ else fact}(x-1, ?y); y := y * x \text{ fi}\ \{y = x!\}}$$

where T_0 is the following valid instance of the axiom of procedure call:

$$\frac{}{\{\underbrace{x \neq 0}_{R} \wedge ((\underbrace{x \geq 0}_{P_{\text{fact}}})[x-1/x])\}\ \text{fact}(x-1, ?y)\ \{\underbrace{x \neq 0}_{R} \wedge ((\underbrace{y = x!}_{Q_{\text{fact}}})[x-1/x, y/y])\}}$$

which is equivalent to: $\dfrac{}{\{x > 0\}\ \text{fact}(x-1, ?y)\ \{x \neq 0 \wedge y = (x-1)!\}}$

Question: Why is this proof valid?

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

Axiomatic Semantics for Total Correctness

Summary

Motivation for total correctness

Partial vs total correctness:

- Partial correctness provides correctness, supposing the termination of the program.
- Total correctness additionally provides termination.

Example 16 (Limitation of partial correctness)

- $\vdash_p \{\text{true}\}$ while true do skip od $\{\text{false}\}$
- $\vdash_p \{\text{true}\}$ while true do skip od $\{\text{true}\}$
- $\vdash_p \{\text{false}\}$ while true do skip od $\{\text{true}\}$
- $\vdash_p \{x = 1 \land y = 2\}$ while true do skip od $\{x = 2 \land y = 1\}$

because for all P, Q (e.g., true, false, $x = 1 \land y = 2$, $x = 2 \land y = 1$),
$P \implies \text{true}$, $\text{false} \implies Q$, and then:

$$\cfrac{\cfrac{\cfrac{}{\{\text{true}\}\ \text{skip}\ \{\text{true}\}}\ [\texttt{skip}]}{\{\text{true}\}\ \text{while true do skip od}\ \{\text{false}^{(*)}\}}\ [\texttt{while}]}{\{P\}\ \text{while true do skip od}\ \{Q\}}\ [\texttt{conseq}]$$

$^{(*)}$ $\neg\text{true} \land \text{true} = \text{false}$

Motivation for total correctness (continued)

Partial vs total correctness:

- Partial correctness provides correctness, supposing the termination of the program.
- Total correctness additionally provides termination.

Example 17 (Limitation of partial correctness - continued)

Similarly, one can easily show that:

- For any program S:
  - $\vdash_p \{\text{true}\}\ S\ \{\text{true}\}$
  - $\vdash_p \{\text{false}\}\ S\ \{\text{true}\}$
  - $\vdash_p \{\text{false}\}\ S\ \{\text{false}\}$
- For any program S that always loops

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

(generalization of previous slide)

Total correctness assertions

Triples of the form:

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

if the precondition P is fulfilled
then S is guaranteed to terminate ($\Downarrow$)
and the state after executing S satisfies the postcondition Q

Inference of Hoare triples

$$\vdash_t \{P\}\ S\ \{\Downarrow\ Q\}$$

Validity of Hoare triples

$$\models_t \{P\}\ S\ \{\Downarrow\ Q\}$$

iff $\forall \sigma \in \textbf{State} : P(\sigma)$ implies $(\exists \sigma' \in \textbf{State}) \begin{cases} Q(\sigma') = \textbf{tt} \\ (S, \sigma) \to \sigma' \end{cases}$

The inference system: terminating statements

Those rules are the same as for partial correctness as they may not themselves involve non-termination.

Skip:

$$\{P\}\ \text{skip}\ \{\Downarrow\ P\}$$

Assignment:

$$\{P[a/x]\}\ x := a\ \{\Downarrow\ P\}$$

Sequential composition:

$$\frac{\{P\}\ S_1\ \{\Downarrow\ Q\} \qquad \{Q\}\ S_2\ \{\Downarrow\ R\}}{\{P\}\ S_1; S_2\ \{\Downarrow\ R\}}$$

Conditional statement:

$$\frac{\{b \wedge P\}\ S_1\ \{\Downarrow\ Q\} \quad \{\neg b \wedge P\}\ S_2\ \{\Downarrow\ Q\}}{\{P\}\ \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}\ \{\Downarrow\ Q\}}$$

Consequence: If $P \Rightarrow P'$ and $Q' \Rightarrow Q$, then:

$$\frac{\{P'\}\ S\ \{\Downarrow\ Q'\}}{\{P\}\ S\ \{\Downarrow\ Q\}}$$

The inference system: while

While loop is where non-termination may be introduced.

The invariant is replaced by a predicate $P(z)$, called variant, which depends on a positive integer logical variable z that decreases strictly at each iteration:

$$\frac{\{P(z+1)\}\ S\ \{\Downarrow P(z)\}}{\{(\exists z \geq 0)\ P(z)\}\ \text{while } b \text{ do } S \text{ od}\ \{\Downarrow P(0)\}}$$

where:

- $P(z+1) \Rightarrow b$, i.e., if $P(z+1)$ holds then the loop body is executed
- $P(0) \Rightarrow \neg b$, i.e., if $P(0)$ holds then the loop is terminated

Since there is a positive number that strictly decreases at each iteration, the loop will have that finite number of iterations (z is an upper bound on the number of iterations), i.e., the while loop will always terminate.

Remark Another source of non-termination is recursive procedures. The total correctness rule for procedures remains out of the scope of this course. □

Example: Total correctness

Example 18

Proving $\{x \geq 0\}$ while $x \neq 0$ do $x := x - 1$ od $\{\Downarrow \text{true}\}$,
i.e., if $x \geq 0$ then the program terminates.

$$\cfrac{\cfrac{\cfrac{}{\{x = z+1\}\ x := x-1\ \{x = z\}}\ [\texttt{ass}]}{\{(\exists z \geq 0)\ x = z\}\ \text{while } x \neq 0 \text{ do } x := x-1 \text{ od}\ \{\Downarrow x = 0\}}\ [\texttt{while}]}{\{x \geq 0\}\ \text{while } x \neq 0 \text{ do } x := x-1 \text{ od}\ \{\Downarrow \text{true}\}}\ [\texttt{conseq}]$$

Justifications of the rule of while:

- $P(z)$ is the predicate $x = z$
- $P(z+1) \implies x \neq 0$ (because $P(z+1)$ is $x = z+1$ with $z \geq 0$)
- $P(0) \implies x = 0$ (trivially as $P(0)$ is $x = 0$)

Justifications of the rule of consequence:

- $x \geq 0 \implies (\exists z \geq 0)\ x = z$
- $x = 0 \implies \text{true}$

Properties of Hoare logic for total correctness

Correctness (We can infer only valid triples)

For every total correctness formula $\{P\}\ S\ \{\Downarrow Q\}$, we have:

$$\text{If } \vdash_t \{P\}\ S\ \{\Downarrow Q\} \text{ then } \models_t \{P\}\ S\ \{\Downarrow Q\}$$

Completeness (We can infer all valid triples)

For every total correctness formula $\{P\}\ S\ \{\Downarrow Q\}$, we have:

$$\text{If } \models_t \{P\}\ S\ \{\Downarrow Q\} \text{ then } \vdash_t \{P\}\ S\ \{\Downarrow Q\}$$

Relation between partial and total correctness

For every assertion P, Q, and statement S, we have:

$$\text{If } \vdash_t \{P\}\ S\ \{\Downarrow Q\} \text{ then } \vdash_p \{P\}\ S\ \{Q\}$$

Remark The converse property does not hold. □

Outline - Axiomatic Semantics - Hoare Logic

Introduction

Axiomatic Semantics for Partial Correctness

Axiomatic Semantics for Total Correctness

Summary

Summary - Axiomatic Semantics - Hoare Logic

Axiomatic Semantics

- Partial vs total correctness of programs.
- Focus on the essential properties.
- Hoare triples are assertions on programs.
- Hoare calculus - inference system.
- Sound and complete.

Applications: design by contract and its implementations (e.g., JML for Java, D and in/out blocks), computer-aided proof of programs, B method.