5. Hoare Logic

David Pereira José Proença Eduardo Tovar

FVOCA 2021/2022
Formal Verification of Critical Applications

CISTER – ISEP
Porto, Portugal

https://cister-labs.github.io/fvoca2122

Ingredients to Reason About Partial Correctness

Recap the Syntax of our Simple While Language

Let us recall the simple While language that we have seen when getting introduced to Formal Semantics.

$$x \in \text{Identifiers}$$

$$n \in \text{Numerals}$$

$$B ::= \text{true} \mid \text{false} \mid B \wedge B \mid B \vee B \mid \neg B \mid E < E \mid E = E \quad \text{(boolean-expr)}$$

$$E ::= n \mid x \mid E + E \mid E * E \mid E - E \quad \text{(int-expr)}$$

$$C ::= \text{skip} \mid C; C \mid x := E \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C \quad \text{(command)}$$

Towards Program Correctness

- In this class we will get to know how can this language be enriched with logical assertions so that we are able to apply Hoare Logic (an axiomatic semantics) to reason about the correctness of programs.
- We consider only partial correctness specifications: programs are required to behave properly if they terminate, but are not required to terminate.
- for total correctness we need to consider assertions that allow to prove termination, that is, the programs behave properly and they terminate.

Extending the While language with Logical Specifications

Syntactic Extension

$$
\begin{array}{lcll}
x & \in & \text{Identifiers} & \\
n & \in & \text{Numerals} & \\
B & ::= & \text{true} \mid \text{false} \mid B \,\&\&\, B \mid B \,||\, B \mid !B \mid E < E \mid E = E & \text{(boolean-expr)} \\
E & ::= & n \mid x \mid E + E \mid E * E \mid E - E & \text{(int-expr)} \\
C & ::= & \text{skip} \mid C; C \mid x := E \mid \text{if } B \text{ then } C \text{ else } C \mid \text{while } B \text{ do } C & \text{(command)} \\
A & ::= & \text{true} \mid \text{false} \mid E < E \mid E = E \mid A \wedge A \mid A \vee A \mid \neg A \mid A \rightarrow A & \\
 & & \mid \forall x.A \mid \exists x.A & \text{(assertions)} \\
S & ::= & \{A\}\ C\ \{A\} & \text{(specification)}
\end{array}
$$

Hoare Triples

Core concept

The atomic concept that we will be using in order to reason about the partial correctness of programs is that known as *Hoare Triple*. It is a specification of the form

$$\{P\}\, C\, \{Q\}$$

that reads as follows: for all states satisfying the *precondition* P, if the program C executes and terminates in a state satisfying the *postcondition* Q, then the triple is valid.

These preconditions and postconditions, as well as other types of logical specifications that we will see ahead, are also known as **contracts**.

Expanding Formal Semantics to Assertions

So far, the formal semantics presented do not include the evaluation of logical assertions. We present below the rules for their interpretation, where s is a program state (the notion of state already presented)

$$
\begin{aligned}
[\![\text{true}]\!](s) &= \top \\
[\![\text{false}]\!](s) &= \bot \\
[\![\neg A]\!](s) &= \neg [\![A]\!](s) \\
[\![A_1 \wedge A_2]\!](s) &= [\![A_1]\!](s) \wedge [\![A_2]\!](s) \\
[\![A_1 \vee A_2]\!](s) &= [\![A_1]\!](s) \vee [\![A_2]\!](s) \\
[\![A_1 \rightarrow A_2]\!](s) &= [\![A_1]\!](s) \rightarrow [\![A_2]\!](s) \\
[\![\forall x.A]\!](s) &= \forall v : int.[\![A]\!](s[x \mapsto v]) \\
[\![\exists x.A]\!](s) &= \forall v : int.[\![A]\!](s[x \mapsto v])
\end{aligned}
$$

Hoare Logic

Validity of Hoare Triples

How to check the validity of Hoare Triples?

- the usual method for assuring the validity of Hoare Triples is to use a **sound** program-proof system.
- by sound one means that it should not allow to conclude specifications that are not valid.

Program-Proof System

A program-proof system is a set of inference rules that can be seen as fundamental laws about programs. In the next slides, we will present such rules. We will introduce proof-systems for Hoare logic and explain the differences between them, mostly focusing on their applicability in terms of verification activities.

The case of the skip command

The skip command does not change the state of the program. Hence, the precondition and postconditions must be the same.

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

The Program-Proof System Rules

The case of the assignment command

The assignment rule states that a postcondition Q can be granted for an assignment command if the condition that results from substituting E for x in Q holds as precondition.

$$\frac{}{\{Q[x \mapsto E]\}\ x := E\ \{Q\}}$$

Instances of the above axiom

- $\{Y = 2\}\ x := 2\ \{Y = X\}$
- $\{x + 1 = n + 1\}\ x := x + 1\ \{x = n + 1\}$
- $\{E = E\}\ x := E\ \{x = E\}$

The case of command sequence

When in the presence of a sequence of commands, we must prove that if the first command C_1 terminates in a postcondition R then, the second command C_2, satisfying R at start, if terminates, must do so in the prescribed postcondition Q.

$$\frac{\{P\}C_1\{R\} \qquad \{R\}C_2\{Q\}}{\{P\}C_1\,;C_2\{Q\}}$$

An example of the sequence rule

A short example

By the assignment axiom we have:

(i) $\{X = x \wedge Y = y\}\ R := X\ \{R = x \wedge Y = y\}$

(ii) $\{R = x \wedge Y = y\}\ X := Y\ \{R = x \wedge X = y\}$

(iii) $\{R = x \wedge X = y\}\ Y := R\ \{Y = x \wedge X = y\}$

By (i) and (ii) and the sequence rule we obtain

$$(iv)\ \{X = x \wedge Y = y\}\ R := X; X := Y\ \{R = x \wedge Y = y\}$$

Now, by (iv) and (iii) we finish the deduction

$$\{X = x \wedge Y = y\}\ R := X; X := Y; Y := R\ \{Y = x \wedge X = y\}$$

The case of conditionals

In the case of conditionals, one must prove that independently of the value of the Boolean B that serves as a guarda for the conditional.

$$\frac{\{B \wedge P\}\, C_1\, \{Q\} \qquad \{\neg B \wedge P\}\, C_2\, \{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ C_1\ \text{else}\ C_2\ \{Q\}}$$

The case fo while loops

In the case of loops, we need to prove an invariant I (i.e., a condition that must hold at the entry and exit of the loop) throughout all the iterations of the loop (we don't know beforehand how many will be).

$$\frac{\{B \wedge I\}\ C_1\ \{I\}}{\{I\}\ \text{while } B \text{ do } C_1 \{I \wedge \neg B\}}$$

The case of logical consequence

This one is a special rule that deals only with logical assertions. It establishes a connection with first-order logic by means of side conditions that are assertions rather than specifications.

$$\frac{\{P_1\}\, C\, \{Q_1\}}{\{P\}\, C\, \{Q\}}$$

if $P \to P_1$ and $Q_1 \to Q$ hold (these are called side-conditions of the rule). Note also that this rule implies that Hoare logic is not meant to be used by itself; it must always be accompanied by some device for establishing the validity of side conditions, such as a decision procedure based on satisfiability, or an inference system for first-order logic.

A simple example using the logical consequence rule

The small derivation below shows the applicability of the logical consequence rule, followed by the assignment rule. The logical consequence rule generates two side effects which can be trivially proved, and enables the application of the assignment rule.

$$\dfrac{\dfrac{}{\{x+1>10\}\ x := x+1\ \{x>10\}}\ \text{(assignment rule)}}{\{x>10\}\ x := x+1\ \{x>10\}}\ \text{if } x>10 \rightarrow x+1>10 \text{ and } x>10 \rightarrow x>10$$

Hoare Logic in practice

How to deal with Hoare Logic proofs

Two ways

- directly encoding the inference system in the logic of the proof tool and reasoning simultaneously with rules of both Hoare logic and first-order logic as required: reasoning starts with the former but switches to the latter logic when side conditions are introduced by the consequence rule.
- The alternative approach consists in two steps:
  - A proof tree is constructed for the desired specification, assuming that the side conditions generated by the consequence rule are valid
  - An external proof tool is used (such as a theorem prover or a proof assistant) to actually establish the validity of the side conditions

We will continue in the next theoretical class with the alternative method mentioned.