Program Verification

8. Program Verification
8.1 Introduction
8.2 A Hoare logic for IMP
8.3 Formalization and soundness of the Hoare logic
8.4 Program verification with Isabelle/HOL
8.5 Verifying procedural, heap-manipulating programs
8.6 Software verification tools
   Tools for interactive software verification
   Extended static checking
   Specification and refinement
   ATP: Automated theorem proving

Motivation

What is program verification?

- Show that a program has specified properties
- Style of specification depends on programming paradigm and language

We consider imperative programs with properties formulated over pre- and poststates

Why program verification?

- Verification of algorithms and their implementation
- Proof that specific errors cannot happen
8. Program Verification

8.1 Introduction

Learning objectives

• General concept of program verification
• Hoare logic
• Verification of simple sequential imperative programs
• Relationship of Hoare logic and semantics
• Formalization of Hoare logic and soundness proof
• Other approaches and tools for program verification

Example: Program

Splitting step in Quicksort:

```c
int split( int[] arr, int lwb, int upb){
    int left, pivot, right;
    int result, tmp;
    left = lwb;
    pivot = arr[upb];
    right = upb-1;
    while( left <= right ) {
        while( arr[left]<pivot ){
            left = left+1;
        }
        while( left <= right && pivot <= arr[right] ){
            pivot = arr[right];
            right = right-1;
        }
        if( left <= right ) {
            tmp = arr[left];
            arr[left] = arr[right];
            arr[right] = tmp;
            left = left+1;
        } else { ; }
        tmp = arr[left];
        arr[left] = arr[upb];
        arr[upb] = tmp;
        result = left;
        return result;
    }
}
```

Example: Specification

Precondition:

\[ \emptyset \leq lwb \land lwb < upb \land upb < arr\.length \]

Postcondition:

Splitting array into elements below and above pivot:

\[ lwb \leq result \land result \leq upb \land (\forall i. lwb \leq i \land i < result \rightarrow arr[i] \leq arr[result] ) \land (\forall i. result < i \land i \leq upb \rightarrow arr[i] \geq arr[result] ) \]

Not treated:

Array contains the same elements in poststate as in prestate

Books on program verification

• Krzysztof R. Apt, Frank S. de Boer, E.-R. Olderog: Verification of Sequential and Concurrent Programs (3. Auflage)
• Roland C. Backhouse: Program Construction and Verification
• Edsger Dijkstra: The Discipline of Programming
• David Gries: The Science of Computer Programming
**Section 8.2**

**A Hoare logic for IMP**

### Syntax of Hoare logic

A Hoare triple:

Formulas in Hoare logic are triples of the form \( \{ P \} \ C \ { Q } \) where
- \( C \) is a command/statement of the programming language
- \( P \) and \( Q \) are first-order formula, so-called assertions, such that
  - program variables can appear in \( P \) and \( Q \) (no quantification over program variables)
  - boolean expressions can be translated to equivalent formulas

**Example:**

Let \( C_2 \) be some command:

\[
\begin{align*}
\{ & x = 7 \land y \leq 3 \land p(x) \land z = Z \\
& \text{IF } x == 7 \land y <= 5 \text{ THEN } z := z + 1 \text{ ELSE } C_2 \text{ END} \\
\{ & p(x) \land z = Z+1 \}
\end{align*}
\]

### Discussion: Syntax

1. Hoare logics vary w.r.t.:
   - the programming language
   - the relationship between boolean expressions and assertions
   - the treatment of variables:
     - Are program and logical variables be syntactically distinguished?
     - Is quantification over program variables allowed?
   - the datatypes and functions available for writing assertions

2. In addition to rules for reasoning about Hoare triples, Hoare logic needs a base logic to reason about assertions, e.g. FOL. That is, strictly speaking, FOL formulas are part of Hoare logic.

### Semantics of Hoare Logic

**Definition**

Let \( s \xrightarrow{C} t \) denote the judgment of the big-step semantics and let

\[
P(s) \equiv \text{def } P[s(v_1)/v_1, \ldots, s(v_n)/v_n]
\]

where \( v_1, \ldots, v_n \) are the program variables in \( P \).

The Hoare triple \( \{ P \} C \{ Q \} \) is valid iff

\[
P(s) \land (s \xrightarrow{C} t) \rightarrow Q(t)
\]

is valid.

**Discussion:**

Often, the semantics of an assertion \( A \) is considered to be the set of states satisfying \( A \) (assuming no free logical variable).
Rules of Hoare Logic

{ P } SKIP { P }

{ P [ E / x ] } x := E { P }

{ P } C1 { Q } { Q } C2 { R }

{ T ( b ) ∧ P } C1 { Q } { ¬ T ( b ) ∧ P } C2 { Q }

{ P } IF b THEN C1 ELSE C2 END { Q }

where

- P[E/x] denotes the substitution of x in P by E
- T(b) denotes the translation of b to an equivalent formula

Remark:
Note: The axioms and rules are schemas.

Applying Hoare logic: an example

Let C ≡

c := 0; -- a1
sq := 1; -- a2
WHILE sq <= x DO
  c := c+1; -- a3
  sq := sq + (2*c+1) -- a4
END

Prove: { 0 ≤ x } C { c * c ≤ x ∧ x < (c + 1) * (c + 1) }

Partial and total correctness

Partial correctness:
If the precondition P holds in a prestate s and the program terminates in a poststate t, then Q holds in t.

Total correctness:
If the precondition P holds in a prestate s, then the program terminates and Q holds in the poststate.

Remark:
- We only considered partial correctness.
- For total correctness, a measure is needed to prove loop termination.
Section 8.3

Formalization and soundness of the Hoare logic

Formalization in Isabelle/HOL

Issues to solve:

1. How are assertions represented and how is syntactical substitution handled, e.g., the assignment axiom?
2. How to formalize axioms, rules, and derivations?
3. How to express (and prove) soundness?

Related Isabelle/HOL theory:
» HoareIMP.thy

Approaches

- Syntax and semantics of IMP: see Chapter 7
- Options to formalize Hoare triples:
  1. Deep embedding:
     » model assertions by a datatype, say assndeep (similar to IMP)
     » model Hoare triples as triples of type assndeep × com × assndeep
     » define validity for assndeep × com × assndeep
  2. Shallow embedding:
     » express assertions by Isabelle/HOL formulas such that they have type:
       type_synonym assn = state ⇒ bool
     » model Hoare triples as triples of type assn × com × assn
     » define validity for assn × com × assn

Type and validity:

- Type of Hoare triples is:
  assn × com × assn
  where assn = state ⇒ bool and state = var ⇒ int
- Validity is defined as a ternary predicate (with mixfix syntax):
  definition hoare_valid :: assn ⇒ com ⇒ assn ⇒ bool ("|= \{(1_)/ (\_)/ \{(1_)\}" 50)
  where
  |= {P}c{Q} ≡ ∀ s t. s -c → t −→ P s −→ Q t

© Arnd Poetzsch-Heffter et al. TU Kaiserslautern 463

© Arnd Poetzsch-Heffter et al. TU Kaiserslautern 464

© Arnd Poetzsch-Heffter et al. TU Kaiserslautern 465

© Arnd Poetzsch-Heffter et al. TU Kaiserslautern 466
8. Program Verification
8.3 Formalization and soundness of the Hoare logic

Deep vs. shallow embedding

Discussion

• Advantages of deep embedding:
  ▶ Faithfully reflects logic as syntactical calculus
  ▶ Assignment axiom can be realized by substitution
  ▶ Simplifies to prove meta-logical properties (e.g., soundness and completeness)

• Advantages of shallow embedding:
  ▶ Less work
  ▶ Base logic already available
  ▶ Full support of Isabelle/HOL for assertions

Soundness

Definition (Soundness/Korrekttheit)
A logical calculus/proof system is sound (German: korrekt) if all derivable formulas are valid.

Proof technique:
Use induction over the (height of) the derivation tree:

• Show that all instances of the axiom schemas are valid
• Assuming that the instances of the premisses in a rule application are valid, show that the instance of the conclusion is valid.

Formalizing Hoare axioms and rules

We demonstrate 2 approaches (see HoareIMP.thy):
1. Hoare axioms and rules as lemmas stating their soundness
2. An inductive definition of what it means to derive a Hoare triple

Remarks:

• The assignment axiom is realized by function update instead of substitution.
• Transformation of boolean expressions is done by the semantic function beval.
• In both approaches, rule application is managed by Isabelle/HOL.
• The second approach is the preferred technique; the first approach is shown for discussion.
Introduction

Using the Hoare logic in its classical form is tedious:
\[ \leadsto \]

Overview

- Hoare logic in \( \text{wp} \)-form (see \textit{HoareIMPwp.thy})
- Automated \( \text{wp} \)-technique in Isabelle/HOL for IMP (see \textit{HoareIMPwp.thy})
- Extension to IMP by arrays (see \textit{HoareIMParry.thy})
- Application in a case study (see \textit{HoareIMParray.thy})

Discussion

- If preconditions are considered as sets of states, the weakest precondition is unique (Why?).
- For more complex programming languages and Hoare logics, the assertion language might be not sufficiently expressive to specify the weakest precondition.
- WP-transformation provides a proof strategy.
- WP-transformation reduces program verification to reasoning on assertions:

\[ \{ P \} C \{ Q \} \leftrightarrow (P \rightarrow \text{wp}(C, Q)) \]

Weakest precondition transformation

Definition (Weakest precondition, WP-transformer)

An assertion \( A = \text{wp}(C, Q) \) expresses the \textit{weakest precondition} of statement \( C \) for postcondition \( Q \) if

\[ \forall P. \{ P \} C \{ Q \} \rightarrow (P \rightarrow A) \]

A \textit{WP-transformer} is an algorithm that takes \( C \) and \( Q \) as arguments and constructs \( \text{wp}(C, Q) \).

Example:
The assignment axiom provides us with a WP-transformer for assignments:

\[ \text{wp}(x := E, Q) = \text{def} \ Q[E/x] \]

WP-transformation for IMP

\[ \begin{align*}
\text{wp}(\text{SKIP}, Q) &= Q \\
\text{wp}(x := E, Q) &= Q[E/x] \\
\text{wp}(C_1; C_2, Q) &= \text{wp}(C_1, \text{wp}(C_2, Q)) \\
\text{wp}(\text{IF } b \ \text{THEN } C_1 \ \text{ELSE } C_2 \ \text{END}, Q) &= (T(b) \rightarrow \text{wp}(C_1, Q)) \land (\neg T(b) \rightarrow \text{wp}(C_2, Q))
\end{align*} \]

Weakest preconditions for while statements can – in general – not be computed.
WP reasoning for loops

To handle programs with while statements, we assume that they are annotated with appropriate invariants $P$:

\[
\text{WHILE } b \text{ INV } P \text{ DO } C \text{ END}
\]

Let $Q$ be the computed postcondition of the while statement, then:

- Use $P$ as precondition for the while statement
- Proof the following two implications:
  1. $P$ is an invariant: $\mathcal{T}(b) \land P \rightarrow \text{wp}(C, P)$
  2. $P$ is sufficiently strong: $\neg\mathcal{T}(b) \land P \rightarrow Q$

WP-reasoning with Isabelle/HOL

Approach:

- Start the proof with an application of the strengthening rule
- Apply wp-rules top-down following the program structure such that
  - the weakest precondition to be generated is represented by a schema variable
  - during further proof steps the schema variables are instantiated
- Finally, prove the remaining subgoals, i.e., the implications appearing as premisses in the strengthening rule (1) and the while rule (2)

Example:

- HoareIMPwp.thy, see lecture

Discussion

Remarks:

- In Isabelle/HOL, the described proof strategy can be implemented as a method (see HoareIMPwp.thy)
- Note: Here, we explicitly use schema variables in proof goals.
- Implementing the corresponding WP-transformer directly would be difficult because of the shallow embedding of the assertions.

Hoare logic in “WP-form”

Axioms and rules are WP-form if they allow to compute the precondition from the postcondition.

Skip and assignment axiom are already in WP-form; the new rules are:

\[
\begin{align*}
\{(Q) C_2 \{ R \} & \quad \{ P \} C_1 \{ Q \} \\
\{ P \} C_1; C_2 \{ R \} & \\
\{ (\mathcal{T}(b) \rightarrow P_1) \land (\neg\mathcal{T}(b) \rightarrow P_2) \} & \text{IF } b \text{ THEN } C_1 \text{ ELSE } C_2 \text{ END } \{ Q \} \\
\{ R \} C \{ P \} & \mathcal{T}(b) \land P \rightarrow R \quad \neg\mathcal{T}(b) \land P \rightarrow Q \\
\{ P \} & \text{WHILE } b \text{ INV } P \text{ DO } C \text{ END } \{ Q \}
\end{align*}
\]

A strengthening rule similar to the consequence rule is needed to prove that the precondition of a program implies the computed precondition.
Case study:
To illustrate a more realistic example, we

- extend IMP by arrays
- extend the Hoare logic appropriately
- verify the splitting step according to the program and specification
given in the introduction

See » HoareIMParray.thy