Proving Correctness of Imperative Programs in Higher Order Logic

Alfio Martini
alfio.martini@gmail.com

5^{th} Workshop-School on Theoretical Computer Science
Passo Fundo - RS - Brasil

October 9-11, 2019

Contents

1 General Goals

2 Hoare Logic & Isabelle

3 Case Study

4 Structured Proofs

5 Concluding Remarks

Outline

1 General Goals

2 Hoare Logic & Isabelle

3 Case Study

4 Structured Proofs

5 Concluding Remarks

General Goals

General Goals

- Correctness Proofs of Sequential Imperative Programs in Isabelle/HOL
- Proof Exploration using Scripts
- Readable and Structured Proofs using Isar
- Discussion of Insertion Sort

Outline

1 General Goals

2 Hoare Logic & Isabelle

3 Case Study

4 Structured Proofs

5 Concluding Remarks

Hoare Tiples (Partial Correctness Assertions)

$$\{P\}\ c\ \{Q\}$$

IF THE PROGRAM c STARTS EXECUTING IN A STATE WHERE THE ASSERTION P IS TRUE, THEN IF c TERMINATES, IT DOES SO IN A STATE WHERE THE ASSERTION Q HOLDS.

Hoare Triple - Power Algorithm

$\{a = A \ \wedge \ b = B \wedge B \geq 0\}$
$i := 0; p := 1; \underline{\textbf{while}}\ i < b\ \underline{\textbf{do}}\ p := p * a; i := i + 1\ \underline{\textbf{od}}$
$\{p = A^B\}$

Hoare Proof Rules

$$\frac{}{\vdash \{P\}\ \underline{\textbf{skip}}\ \{P\}}\ \text{Skip} \qquad \frac{}{\vdash \{Q[x/a]\}\ x := a\ \{Q\}}\ \text{Ass}$$

$$\frac{\vdash \{P\}\ c_1\ \{Q\} \quad \vdash \{Q\}\ c_2\ \{R\}}{\vdash \{P\}\ C_1; C_2\ \{R\}}\ \text{Comp}$$

$$\frac{\vdash \{P \wedge B\}\ c_1\ \{Q\} \quad \vdash \{P \wedge \neg B\}\ c_2\ \{Q\}}{\vdash \{P\}\ \underline{\textbf{if}}\ b\ \underline{\textbf{then}}\ c_0\ \ \underline{\textbf{else}}\ c_1;\ \underline{\textbf{fi}}\ \{Q\}}\ \text{IfE} \qquad \frac{\vdash \{P \wedge B\} c \{P\}}{\vdash \{P\}\ \underline{\textbf{while}}\ b\ \underline{\textbf{do}}\ c\ \underline{\textbf{od}}\ \{P \wedge \neg B\}}\ \text{PWh}$$

$$\frac{\vdash P \rightarrow Q \quad \vdash \{Q\}\ c\ \{R\}}{\vdash \{P\}\ C\ \{R\}}\ \text{Stren} \qquad \frac{\vdash \{P\} C \{Q\} \quad \vdash Q \rightarrow R}{\vdash \{P\}\ C\ \{R\}}\ \text{Weakn}$$

Power Algorithm - Proof Draft

$w \triangleq \underline{\textbf{while}}\ i < b\ \underline{\textbf{do}}\ body\ \underline{\textbf{od}}$

$init \triangleq i := 0; p := 1$ $\qquad body \triangleq p := p * a; i := i + 1;$

$Pre \triangleq a = A \wedge b = B \wedge B \geq 0$ $\qquad Pos \triangleq p = A^B$

$bw \triangleq i < b$ $\qquad INV \triangleq p = a^i \wedge i \leq b \wedge a = A \wedge b = B$

$$\dfrac{\dfrac{\vdots\ \boxed{1}}{\vdash \{Pre\}\ init\ \{INV\}} \qquad \dfrac{\dfrac{\dfrac{\vdots\ \boxed{2}}{\vdash \{INV \wedge bw\}\ body\ \{INV\}}}{\vdash \{INV\}\ w\ \{INV \wedge \neg bw\}} \qquad \vdots\ \boxed{3}}{\vdash \{INV\}\ w\ \{Pos\}}}{\vdash \{Pre\}\ init; w\ \{Pos\}}$$

where $\boxed{3}$ *corresponds to the proof tree of the judgment* $\vdash INV \wedge \neg bw \rightarrow Pos$.

$\square$

Hoare Logic in Theorem Provers

Hoare Logic Automation

- Formalization of PL Syntax
- Formalization of PL Semantics
- Formalization of Proof Rules (theorems w.r.t. the PL semantics)
- Computation of Verification Conditions (VC's)
- Proof of VC's using interactive or automatic theorem provers.

Verification Condition Generator (VCG)

REDUCE PROVABILITY IN HOARE LOGIC TO PROVABILITY IN THE SPECIFICATION LANGUAGE.

$$
\begin{array}{lcl}
vc(\{P\}\ \underline{\textbf{skip}}\ \{Q\}) & = & \{P \rightarrow Q\} \\
vc(\{P\}\ x := a\ \{Q\}) & = & \{P \rightarrow Q[x/a]\} \\
vc(\{P\}\ c_0; x := a\ \{Q\}) & = & vc(\{P\}\ c_0\ \{Q[x/a]\}) \\
vc(\{P\}\ c_0; \{D\}\ c_1\ \{Q\}) & = & vc(\{P\}\ c_0\ \{D\}) \cup vc(\{D\}\ c_1\ \{Q\}) \\
 & (c_1 \text{ not an assignment}) & \\
vc(\{P\}\ \underline{\textbf{if}}\ b\ \underline{\textbf{then}}\ c_0\ \ \underline{\textbf{else}}\ c_1;\ \underline{\textbf{fi}}\ \{Q\}) & = & vc(\{P \wedge b\}\ c_0\ \{Q\}) \\
 & & \cup\ vc(\{P \wedge \neg b\}\ c_1\ \{Q\}) \\
vc(\{P\}\ \underline{\textbf{while}}\ b\ \underline{\textbf{do}}\ \{D\}c\ \{Q\}) & = & vc(\{D \wedge b\}\ c\ \{D\}) \\
 & & \cup\ \{P \rightarrow D\} \cup \{D \wedge \neg b \rightarrow Q\}
\end{array}
$$

Why Using Verification Condition Generator?

- No knowledge of Hoare logic is required by the person or machine that attempts to prove the generated verification conditions.
- Most systems for the verification of imperative programs are based on VCG's
- JML,KEY TOOL, DAPHNY, HAHA,FRAMA-C,SPARKADA,ETC.
- BOOGIE, WHYML (intermediate languages)

Power Algorithm - Verification Conditions

1. Invariant is true on the initial state
2. Invariant is preserved by the loop
3. Invariant establishes the postcondition

```
proof (prove)
goal (3 subgoals):
 1. ⋀a b p i. a = A ∧ b = B ⟹ 1 = a ^ 0 ∧ 0 ≤ b ∧ a = A ∧ b = B
 2. ⋀a b p i.
       (p = a ^ i ∧ i ≤ b ∧ a = A ∧ b = B) ∧ i < b ⟹
       p * a = a ^ (i + 1) ∧ i + 1 ≤ b ∧ a = A ∧ b = B
 3. ⋀a b p i.
       (p = a ^ i ∧ i ≤ b ∧ a = A ∧ b = B) ∧ ¬ i < b ⟹ p = A ^ B
```

Proof Script

```
lemma imp_pot:
"VARS (a::int) (b::nat) (p::int) (i::nat)
{a=A ∧ b=B}
i := 0; p := 1;
WHILE i<b
  INV { p = a^i ∧ i≤ b ∧ a=A ∧ b = B}
  DO p := p * a;i:=i+1 OD
{p = A^B}"
apply (vcg)
apply (auto)
done
```

Proof Scripts - Features

- Imperative language
- List of `apply` commands that manipulate the proof state.
- Arguments are usually **proof methods** or **proof rules**.
- One has to play the proof in Isabelle in order to understand the sequence of state changes in the proof state.
- Very useful for proof exploration.

Outline

1. General Goals
2. Hoare Logic & Isabelle
3. Case Study
4. Structured Proofs
5. Concluding Remarks

Insertion Sort Triple

```
definition is_perm::"'a list ⇒ 'a list ⇒ bool"
where "is_perm l1 l2 ≡ length l1 = length l2
                       ∧ (∀x. count x l1 = count x l2)"

lemma inss_hoare: "VARS xs ys :: ('a::linorder) list
  {xs=X}
  ys:=[];
 WHILE xs ≠ []
    INV {isorted ys ∧ is_perm X (ys @ xs)}
 DO ys := ins (hd xs) ys; xs := tl xs OD
{isorted ys ∧ is_perm X ys}"
```

Function Definitions

```
fun ins::"'a::linorder ⇒ 'a list ⇒ 'a list" where
  "ins x [] = [x]" |
  "ins x (y # ys) =
     (if x ≤ y then (x # y # ys) else y#ins x ys)"

fun iSort::"('a::linorder) list ⇒ 'a list" where
"iSort [] = []" |
"iSort (x # xs) = ins x (iSort xs)"

fun le::"('a::linorder) ⇒ 'a list ⇒ bool"  where
"le x [] = True" |
"le x (y # ys) = (x ≤y ∧ le x ys)"

fun isorted::"('a::linorder) list ⇒ bool" where
"isorted [] = True" |
"isorted (x # xs) = (le x xs ∧ isorted xs)"

fun count:: "'a ⇒ 'a list ⇒ int" where
"count x [] = 0" |
"count x (y # ys) =(if x=y then  1 + count x ys else count x ys)"
```

Verification Conditions

```
proof (prove)
goal (3 subgoals):
 1. ⋀(xs::'a list) ys::'a list.
       xs = X ⟹ isorted [] ∧ is_perm X ([] @ xs)
 2. ⋀(xs::'a list) ys::'a list.
       (isorted ys ∧ is_perm X (ys @ xs)) ∧ xs ≠ [] ⟹
       isorted (ins (hd xs) ys) ∧ is_perm X (ins (hd xs) ys @ tl xs)
 3. ⋀(xs::'a list) ys::'a list.
       (isorted ys ∧ is_perm X (ys @ xs)) ∧ ¬ xs ≠ [] ⟹
       isorted ys ∧ is_perm X ys
variables:
  X :: 'a list
```

Essential Lemmas

```
lemma le_ins: "le x (ins a xs) = (x ≤ a ∧ le x xs)"
lemma le_mon:"x≤y ⟹ le y xs ⟹ le x xs"
lemma ins_sorted: "isorted (ins a xs) = isorted xs"
lemma is_sorted:"isorted(iSort xs)"
lemma ins_count:
 "count x (ins k xs) = (if x = k then 1 + count x xs else count x xs)"
lemma count_sum:"count x (xs @ ys) = count x xs + count x ys"
lemma len_sort:"length(iSort xs) = length xs"
lemma count_iSort: "count x (iSort xs) = count x xs"
lemma ins_len:"length (ins k xs) = 1 + length xs"
```

Proof Script

```
apply (vcg)
apply (auto simp add:is_perm_def) — ‹ 1 ›
   apply (simp add: ins_sorted)  — ‹ 2 ›
   apply (simp add: ins_len) — ‹ 3 ›
   apply (smt count.simps(2) count_sum hd_Cons_tl ins_count) — ‹ 4 ›
done
```

Outline

1 General Goals

2 Hoare Logic & Isabelle

3 Case Study

4 Structured Proofs

5 Concluding Remarks

Isar Proof Language

- Structured Proofs
- Declarative Language
- Atomic and Compound Proofs
    - **fix** variables
    - **assume** assumptions
    - **show** proposition
    - **have** is used to establish intermediate facts
    - **show** is used to establish current or main goal

Insertion Sort - Isar Draft

```
proof (vcg)
   fix xs ys
   assume ass:"(isorted ys ∧ is_perm X (ys @ xs)) ∧ xs ≠ []"
   show "isorted (ins (hd xs) ys)
          ∧ is_perm X ((ins (hd xs) ys) @ tl xs)"
    proof (rule conjI)
       show "isorted (ins (hd xs) ys)" sorry
    next
       have pg1:"length X  =  length ((ins (hd xs) ys) @ tl xs)"
          sorry
       have pg2:"∀ k. count k X = count k (ins (hd xs) ys @ tl xs)"
          sorry
       from pg1 pg2 show "is_perm X (ins (hd xs) ys @ tl xs)" sorry
   qed
qed (auto simp add:is_perm_def)
```

Insertion Sort - Isar Proof

```
proof (rule conjI)
   from ass have "isorted ys" by simp
   from this show "isorted (ins (hd xs) ys)" by (simp add:ins_sorted)
next
  from ass have 1:"is_perm X (ys @ xs)" and 2:"xs ≠ []" by auto
  from 2 have hdtl:"xs = hd xs # tl xs" by simp
  from 1 have 3:"∀ x. count x X = count x (ys @ xs)" by (simp add:is_perm_def)
  have pg1:"length X  =  length ((ins (hd xs) ys) @ tl xs)"
     proof -
        from 1 have 4:"length X = length (xs @ ys)" by (simp add:is_perm_def)
        also have "...= length xs + length ys" by simp
        also have "... =  1 + length ys + length xs - 1" by simp
        also have "... = length (ins (hd xs) ys) + length (tl xs)"
                       by (simp add: "2" ins_len)
        also have "... = length ((ins (hd xs) ys) @ tl xs)" by simp
        finally show ?thesis  by simp
     qed
```

Outline

1. General Goals
2. Hoare Logic & Isabelle
3. Case Study
4. Structured Proofs
5. Concluding Remarks

Concluding Remarks

Conclusion

- Isabele provides a flexible set of tools for reasoning with about imperative programs in Hoare Logic.
- Proof exploration based on proof scripts and automation.
- Proof documentation based on structured proofs.
- Structured proofs help to control proof complexity and to convey clear reasoning.
- Final proof text can be presented in several levels. of detail, depending on the target audience and user experience.
- Proofs are often hard.
- Interactive Proving and Counter Model generation are essential.