

# Proving Correctness of Imperative Programs in Higher Order Logic

Alfio Martini  
alfio.martini@gmail.com

5<sup>th</sup> 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 Triples (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

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

# Hoare Proof Rules

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

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

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

$$\frac{\vdash \{P \wedge B\} c_1 \{Q\} \quad \vdash \{P \wedge \neg B\} c_2 \{Q\}}{\vdash \{P\} \text{if } b \text{ then } c_0 \text{ else } c_1; \text{fi} \{Q\}} \text{IfE}$$

$$\frac{\vdash \{P \wedge B\} c \{P\}}{\vdash \{P\} \text{while } b \text{ do } c \text{ od } \{P \wedge \neg B\}} \text{PWh}$$

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

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



# Power Algorithm - Proof Draft

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

$\text{init} \triangleq i := 0; p := 1$

$\text{Pre} \triangleq a = A \wedge b = B \wedge B \geq 0$

$\text{bw} \triangleq i < b$

$\text{body} \triangleq p := p * a; i := i + 1;$

$\text{Pos} \triangleq p = A^B$

$\text{INV} \triangleq p = a^i \wedge i \leq b \wedge a = A \wedge b = B$

$$\frac{\frac{\vdots [1]}{\vdash \{\text{Pre}\} \text{init} \{\text{INV}\}} \quad \frac{\frac{\vdots [2]}{\vdash \{\text{INV} \wedge \text{bw}\} \text{body} \{\text{INV}\}} \quad \frac{\vdash \{\text{INV}\} w \{\text{INV} \wedge \neg \text{bw}\} \quad \vdots [3]}{\vdash \{\text{INV}\} w \{\text{Pos}\}}}{\vdash \{\text{Pre}\} \text{init}; w \{\text{Pos}\}}$$

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

□

# Hoare Logic in Theorem Provers

## Hoare Logic Automation

- Formalization of PL Syntax
- Formalization of PL Semantics
- Formalization of Proof Rules (theorems of the 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.

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

## 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.  $\bigwedge a\ b\ p\ i. a = A \wedge b = B \implies 1 = a \wedge 0 \leq b \wedge a = A \wedge b = B$
2.  $\bigwedge a\ b\ p\ i.$   
 $(p = a \wedge i \leq b \wedge a = A \wedge b = B) \wedge i < b \implies$   
 $p * a = a \wedge (i + 1) \leq b \wedge a = A \wedge b = B$
3.  $\bigwedge a\ b\ p\ i.$   
 $(p = a \wedge i \leq b \wedge a = A \wedge b = B) \wedge \neg i < b \implies p = A \wedge B$

# 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  $\Rightarrow$  'a list  $\Rightarrow$  bool"  
where "is_perm l1 l2  $\equiv$  length l1 = length l2  
 $\wedge$  ( $\forall x$ . count x l1 = count x l2)"
```

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

# Function Definitions

```
fun ins::"'a::linorder  $\Rightarrow$  'a list  $\Rightarrow$  'a list" where
  "ins x [] = [x]" |
  "ins x (y # ys) =
    (if x  $\leq$  y then (x # y # ys) else y#ins x ys)"
```

```
fun iSort:: "('a::linorder) list  $\Rightarrow$  'a list" where
  "iSort [] = []" |
  "iSort (x # xs) = ins x (iSort xs)"
```

```
fun le:: "('a::linorder)  $\Rightarrow$  'a list  $\Rightarrow$  bool" where
  "le x [] = True" |
  "le x (y # ys) = (x  $\leq$  y  $\wedge$  le x ys)"
```

```
fun isorted:: "('a::linorder) list  $\Rightarrow$  bool" where
  "isorted [] = True" |
  "isorted (x # xs) = (le x xs  $\wedge$  isorted xs)"
```

```
fun count:: "'a  $\Rightarrow$  'a list  $\Rightarrow$  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.  $\bigwedge (xs :: 'a \text{ list}) \ ys :: 'a \text{ list}.$ 
       $xs = X \implies \text{isorted } [] \wedge \text{is\_perm } X ([] @ xs)$ 
  2.  $\bigwedge (xs :: 'a \text{ list}) \ ys :: 'a \text{ list}.$ 
       $(\text{isorted } ys \wedge \text{is\_perm } X (ys @ xs)) \wedge xs \neq [] \implies$ 
       $\text{isorted } (\text{ins } (\text{hd } xs) \ ys) \wedge \text{is\_perm } X (\text{ins } (\text{hd } xs) \ ys @ \text{tl } xs)$ 
  3.  $\bigwedge (xs :: 'a \text{ list}) \ ys :: 'a \text{ list}.$ 
       $(\text{isorted } ys \wedge \text{is\_perm } X (ys @ xs)) \wedge \neg xs \neq [] \implies$ 
       $\text{isorted } ys \wedge \text{is\_perm } X \ ys$ 
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  $\wedge$  is_perm X (ys @ xs))  $\wedge$  xs  $\neq$  []"
  show "isorted (ins (hd xs) ys)
         $\wedge$  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:" $\forall$  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 pgl:"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
qed
```

# Outline

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

# Concluding Remarks

## Conclusion

- Isabelle 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.