Automated Reasoning – Coursework Assignment 1

Software verification using Hoare logic in Isabelle

Petros Papapanagiotou
pe.p@ed.ac.uk

7 October 2013

THE UNIVERSITY of EDINBURGH
informatics

Breakdown

- Part 1 : Natural Deduction (40 marks)
  - 14 lemmas to prove
- Part 2 : Hoare Logic (60 marks)
  - Part 2a : Verify 6 algorithms (15 marks)
  - Part 2b : Verify the MinSum algorithm (45 marks)

Isabelle / HOL

- A modern proof assistant.
- Written in PolyML.
- Supports multiple interfaces:
  - ProofGeneral – Developed in UoE, supported on DICE.
  - jEdit
- Multiple tools:
  - Extensive libraries of theories and lemmas.
  - Automated proof procedures.
  - Various helpful tools (eg. counterexample checker)

Isabelle / HOL - Resources

- Getting started guide (use this to run Isabelle under DICE):

  http://www.inf.ed.ac.uk/teaching/courses/ar/isabelle/isabelle-startup.pdf

- Tutorial / Documentation:

  http://www.cl.cam.ac.uk/research/hvg/Isabelle/documentation.html

- Cheat Sheet:

  http://www.inf.ed.ac.uk/teaching/courses/ar/FormalCheatSheet.pdf

Isabelle / HOL - Syntax

- Comments:

```
text {* COMMENTS *}
```

- Symbols:

\<and>	/\	∧
\<or>	\/	∨
\<forall>	ALL	∀
\<exists>	EX	∃
\<longrightarrow>	-->	⟶
\<Longrightarrow>	==>	⟹

- To view a theorem:

```
thm FOO
```

Isabelle HOL – Tactics + rules

- Basic tactics:

rule	rule_tac	introduction (backward)
erule	erule_tac	elimination (forward + backward)
drule	drule_tac	destruction (forward)
frule	frule_tac	forward

- Basic natural deduction rules:

conjI	conjE	conjunct1	conjunct2
disjI1	disjI2	disjE	
impI	impE	mp	
iffI	iffD1	iffD1	iffE
notI		notE	
allI	allE	exI	exE
excluded-middle		ccontr	

Isabelle / HOL – Tactics usage

- Simple application:

```
apply (rule exI)
```

- Instantiation:

```
apply (rule_tac x=A in exI)
```

- Multiple instantiations:

```
apply (drule_tac P=P and Q=Q in disjI1)
```

Other basic commands and tactics

`apply (assumption)`	Prove by matching the goal to an assumption.
`prefer`	Prioritize a subgoal.
`defer`	Postpone a subgoal.
`done`	Finish a proof with no subgoals.
`oops / sorry`	Postpone a proof. (*that doesn't mean you proved it!*)

Assignment Part 1

- Practice in natural deduction proofs in Isabelle.

- Using **only** basic rules and tactics, prove 14 lemmas.

- Including one of DeMorgan's laws and Russel's "barber" paradox.

- Lemmas marked individually, total **40%.**

Isabelle / HOL – Advanced tactics

- You are **not** allowed to use these in Part 1!

`case_tac P`	Case split over possible values of P (not necessarily boolean).
`clarify`	Clarify the subgoal using simple rules.

`simp`
`simp add: FOO BAR`
`simp only: FOO BAR`
`simp del: FOO BAR`
	Simplify goal + assumptions using core rules.
	- Add theorems FOO and BAR.
	- Use only theorems FOO and BAR (not core rules).
	- Exclude FOO and BAR from the core rules.

`auto` `auto simp add: FOO BAR`	Try to prove all subgoals automatically. - Also use the simplifier adding rules FOO and BAR.
`blast / force`	Other automated procedures.
`oops / sorry`	Postpone a proof. (*that doesn't mean you proved it!*)

Isabelle / HOL – Hoare Logic

- We can use Isabelle's Hoare Logic library to reason about a simple WHILE programming language:

`VARS x y z`	Local variables.
`p ; q`	Sequence.
`SKIP`	Do nothing.
`X := 0`	Assignment.

`IF cond`
`THEN p`
`ELSE q`
`FI`
	Conditional.

`WHILE cond`
`INV { invariant }`
`DO p`
`OD`
	While loop.
	
	Invariant must be explicit!

Isabelle / HOL – Formal Specification

- Using this programming language, we can express Hoare triples in Isabelle.
- Example (from Hoare Logic lecture):

```
lemma Fact: "VARS (Y::nat) Z
 {True}
 Y := 1;
 Z := 0;
 WHILE Z ≠ X
 INV { Y = fact Z }
 DO
  Z := Z + 1;
  Y := Y * Z
 OD
 { Y = fact X }"
```

Isabelle / HOL – VCs

- Isabelle can automatically extract VCs with the Verification Condition Generation tactic:

```
apply vcg
```

- Result :

```
proof (prove): step 1

goal (3 subgoals):
 1. ⋀ Y Z. True ⟹ 1 = fact 0
 2. ⋀ Y Z. Y = fact Z ∧ Z ≠ X ⟹ Y * (Z + 1) = fact (Z + 1)
 3. ⋀ Y Z. Y = fact Z ∧ ¬ Z ≠ X ⟹ Y = fact X
```

** Remember these from the Hoare Logic lecture?*

Isabelle HOL - VCs

```
proof (prove): step 1

goal (3 subgoals):
 1. ⋀ Y Z. True ⟹ 1 = fact 0
 2. ⋀ Y Z. Y = fact Z ∧ Z ≠ X ⟹ Y * (Z + 1) = fact (Z + 1)
 3. ⋀ Y Z. Y = fact Z ∧ ¬ Z ≠ X ⟹ Y = fact X
```

- We can use Isabelle tactics, rules, and lemmas to prove VCs.
- In this example, `simp` "knows enough" about `fact` to solve all subgoals, but this will not always be the case.
- Alternative: `vcg_simp` (`vcg` + `simp`)
- Correctness of the `Fact` algorithm is now verified based on the definition and properties of `fact` in Isabelle!

Assignment Part 2a

- Verify 6 simple algorithms:

`Min`	`Multi1`	`DownFact`
`Copy`	`Multi2`	`Div`

- Use any rule/lemma from the available theories (you may **not** import more) and any of the tactics described here or in the Cheat Sheet (including `simp` and `auto`).
- Introduce the appropriate loop invariant and postcondition where necessary:
  - Replace the `Inv` variable (**not** the `INV` keyword) with your invariant.
  - Replace the Postcondition variable with your postcondition.
- Algorithms marked individually, total **15%.**

Assignment Part 2b

- Verify the minimum section sum algorithm `MinSum`.

$$S_{i,j} = A[i] + A[i+1] + \ldots + A[j]$$

eg: $A = [1,2,3,4]$ $S_{1,2} = 2 + 3 = 5$

- Two specifications:
  - **S1**: *The sum* `s` *is less than or equal the sum of any section of the array.*
  - **S2**: *There exists a section of the array that has sum* `s`.

Assignment Part 2b

- Verify the minimum section sum algorithm `MinSum`.

```
fun sectsum :: "int list ⇒ nat ⇒ nat ⇒ int" where
"sectsum l i j = listsum (take (j-i+1) (drop i l))“
```

eg: `sectsum [1,2,3,4] 1 2 =`

```
listsum (take (2-1+1) (drop 1 [1,2,3,4])) =
listsum (take 2 [2,3,4]) =
listsum [2,3] =
2 + 3 = 5
```

- Two specifications:
  - **S1**: `∀i j. 0≤i ∧ i≤j ∧ j<length A → s ≤ sectsum A i j`
  - **S2**: `∃i j. 0≤i ∧ i≤j ∧ j<length A ∧ s = sectsum A i j`

Assignment Part 2b

- **S1**: `∀i j. 0≤i ∧ i≤j ∧ j<length A → s ≤ sectsum A i j`
- Proof:

  Huth & Ryan, Section 4.3.3 (pp. 287-292)

  - Introduces a loop invariant with 2 parts. These are already defined as functions `Inv1` and `Inv2`. Use `simp` with `Inv1.simps` and `Inv2.simps`.
  - Requires proof of Lemma 4.20 which has 2 parts:

    `lemma4_20a` and `lemma4_20b`

- Prove both parts of Lemma 4.20 and use them to verify **S1** by proving `lemma MinSum`. (**25%**)

Assignment Part 2b

- **S2**: `∃i j. 0≤i ∧ i≤j ∧ j<length A ∧`
  `s = sectsum A i j`

- Introduce the appropriate invariant.
- Develop your own proof from scratch.

- Verify **S2** by proving `lemma MinSum2` (**20%**).

- Lecture 6 – H&R Secs 4.1-4.3
- Isabelle links
- Drop-in lab: AT 5.05 (West Lab), Thursdays 2pm – 3pm
- Discussion Forum & Mailing list
- Me: pe.p@ed.ac.uk

- Don’t change imports and definitions!
- Plan your proofs on paper *before* you try them on Isabelle!
- Prove as many extra lemmas as you need!
- Write comments (especially for part 2b)!
- If you cannot prove something, take it as far as you can, write comments, and use “`sorry`”!

- Your matriculation number in the file!
- **Start early!**
- No plagiarism!

- Don't chang … s!
- Plan your … n Isabelle!
- … t as far as you can,
- …, and use "`sorry`"!

Good Luck!

<u>Deadline:</u>

Monday, 28 Oct 2013, 14:00