Program verification in Lean II

Completing the verification proof and planning the final project

CMPUT 664 · Formal Verification · October 26, 2026

Where we stopped

We defined the language, execution relation, and VC generator in Lean.

```
def Verify {S : Type} (P : S → Prop) (c : ACmd S)
    (Q : S → Prop) : Prop :=
  (∀ s, P s → pre c Q s) ∧ VC c Q
```

The remaining theorem formalizes the connection to the Hoare triple:

```
theorem verify_sound {S : Type} (P : S → Prop) (c : ACmd S)
    (Q : S → Prop) (h : Verify P c Q) :
    Hoare P (erase c) Q
```

Soundness proof plan

First prove the triple for the calculated entry assertion:

```
1 vc_sound (c : ACmd S) (Q : S → Prop) (hvc : VC c Q) :
2   Hoare (pre c Q) (erase c) Q
```

Use induction on `c`, composing the Hoare rules for each command form.

Then use the entry implication from `Verify`:

$$P(s) \quad \Longrightarrow \quad \mathrm{pre}(c, Q)(s).$$

The induction hypothesis

```
1  induction c generalizing Q with
```

Subcommand	Postcondition required by the recursive calculation
First command in `seq c₁ c₂`	`pre c₂ Q`
Second command	`Q`
Loop body	`I`

The induction hypotheses must work for **any requested postcondition**.

This lets the proof follow the recursive calls in `pre` and `VC`.

Assignment and sequence cases

Assignment: `pre (.assign f) Q` is `fun s => Q (f s)`, exactly the precondition of the assignment rule.

Sequence: let $R = \mathrm{pre}(c_2, Q)$.

$$\{\mathrm{pre}(c_1, R)\}\ \ \mathrm{erase}(c_1)\ \{R\}$$

$$\{R\}\ \ \mathrm{erase}(c_2)\ \{Q\}$$

The induction hypotheses establish these two triples. The Hoare sequence rule composes them.

The loop case

Generated fact	Role in the proof
Body VCs	Use the body induction hypothesis with target I
Preservation	Establish the body's calculated entry assertion from I and a true guard
Exit	Obtain Q from I and a false guard

The resulting body triple restores I after an iteration.

Apply the Hoare while rule, then the exit implication, to obtain the loop's postcondition Q.

The two induction arguments

Theorem	Induction object
`vc_sound`	Annotated program syntax
Semantic while rule	A terminating execution derivation

The body is smaller syntax. The remaining iterations execute the same loop, with smaller execution evidence.

The generator proof uses the already established semantic while rule.

Finishing verify_sound

```
1 theorem verify_sound {S : Type} (P : S → Prop) (c : ACmd S)
2     (Q : S → Prop) (h : Verify P c Q) :
3     Hoare P (erase c) Q := by
4   have hCore : Hoare (pre c Q) (erase c) Q := vc_sound c Q h.2
5   intro s t hs hrun
6   have hEntry : pre c Q s := h.1 s hs
7   exact hCore s t hEntry hrun
```

`h.2` supplies the side conditions. `h.1` establishes the calculated entry assertion from `P`.

The counting contract

```
1 theorem count_correct :
2     Hoare (fun _ => True) count (fun x => x = 10) :=
3   verify_sound _ _ _ count_vcs
```

`count_vcs` proves the conditions for counting. The generic theorem turns that proof into the Hoare triple.

Every terminating execution of counting ends at ten.

Final project planning

Scope, timeline, and proposal meetings

The project target

Try a verification idea in a small, thoughtful, and relatively creative project that you can complete in about a month.

- Choose a focused question connected to the course.
- Make a meaningful choice of application, adaptation, comparison, or extension.
- Produce technical work and evidence that let you explain what you learned.

Existing tools and examples can be starting points. Reproducing a tutorial or an existing result unchanged is not enough.

A scope you can finish

Define a **minimum result** that would already answer a useful question.

- Keep the program, model, or technique small enough to understand and evaluate.
- Identify what you need to implement, prove, or experiment with.
- Explain what evidence would support your conclusion.
- Identify the main risk and a smaller fallback if the ambitious part fails.

Leave time in November to evaluate the work and prepare the presentation. Treat extra features as optional extensions.

Project timeline

Date	Milestone
Wed, October 28	Group project meetings in my office; **no class session**
Fri, October 30	**One-page proposal due at 11:59 p.m. MT**
November	Develop the project, evaluate the results, and prepare the presentation
November 9–13	Reading Week; no class
November 18 and 25	No class meeting; project work
Wed, December 2	Project presentations I
Mon, December 7	Project presentations II

Final-report details and deadline will be announced separately.

Wednesday: group project meetings

October 28: no class session. Meet with me in my office.

- Each project group must book **one 30-minute meeting**.
- Meetings are available between **12:30 and 4:30 p.m. MT**.
- Discuss what you want to do and why the plan is feasible.

Use the feedback to refine your proposal before Friday's deadline.

Book your group's 30-minute meeting

Be ready to discuss

What do you want to do?

State the question, the verification idea, and what your group will contribute.

Why is it feasible in about a month?

Explain the starting point, required tools or background, minimum result, and main risk.

How will you know what you achieved?

Describe the property to prove, comparison to make, or evidence to collect.

The one-page proposal

One proposal per group, due Friday, October 30, at 11:59 p.m. MT.

Include:

- Team members and the focused problem or question.
- The verification idea, relevant prior work, and your intended contribution.
- The technical approach and how you will evaluate the result.
- A feasible work plan with a minimum result, main risk, and fallback.

Incorporate the feedback from Wednesday's meeting.

Teams normally have **two or three students**; individual projects require prior approval.