Proof Assistants (L81): Assignment I

Thomas Bauereiss Meven Lennon-Bertrand

Michaelmas 2024

This is the first of two marked assignments for the course "Proof Assistants" (L81), covering the Isabelle part of the course. This assignment comprises a number of small formalisation and verification tasks related to the imperative programming language IMP, as well as a short write-up explaining your work.

The due date for this assignment is Wednesday, 20 November 2024, 4pm. You must work on this assignment as an individual; collaboration is not allowed. Copying material found elsewhere counts as plagiarism. Please use the Moodle page for the course to submit an archive containing your theories and write-up by the deadline.

Each of the two assignments is worth 100 marks, distributed as follows:

- 50 marks for completing basic formalisation and verification tasks assessing grasp of the material taught in the lecture. For this assignment, please use the Isabelle theory file template provided on the course website. When preparing your submission, please include all theory files that you have edited; some tasks ask you to modify definitions and proofs in existing theories, so please be sure to include those. You are allowed to use Sledgehammer to find proofs, although you might want to streamline any complex proofs it finds into something more readable. The main assessment criteria for these tasks are correctness and completeness of the specifications and proofs. You might want to test your specifications to check that they work as intended.

- 20 marks for completing designated more challenging tasks, which might require more creativity when designing specifications and proof strategies, and might require some Isabelle techniques that go beyond what was taught in the lecture. Completing these advanced tasks can help you achieve a distinction grade, but you might want to focus first on completing the main tasks before attempting the advanced tasks.

- 30 marks for a clear write-up, where 10 of these marks will be reserved for write-ups of exceptional quality, e.g. demonstrating particular insight. In general, your write-up should explain the design decisions

you made during the formalisation and the strategies you used for the proofs. It might also discuss proof attempts that failed in interesting ways and lessons you learned from them, if that happens. The maximum length of the write-up is 2,500 words, although it could be much shorter, especially if you add comments to your theory files. It is possible to use Isabelle to generate a document,[1] but you can use any tool you prefer to prepare your write-up and add a PDF to your submission.

[1]The template files for the assignment are set up to generate the handout document with this introduction and the tasks. For more information on document preparation from Isabelle, see Chapter 4 of "The Isabelle/Isar Reference Manual" and Chapter 3 of "The Isabelle System Manual".

Assignment I: The IMP language in Isabelle

Task 1 (5 marks) Define a function *vars* :: *aexp* ⇒ *vname set* that collects all variables appearing in an expression, and prove that the value of an expression is unaffected by changes to variables not appearing in it:

lemma *aval_other_var_update*:
$x \notin vars\ e \Longrightarrow aval\ e\ (s(x := y)) = aval\ e\ s$

Then show the following commutativity lemma for assignments:

lemma *assign_commute*:
assumes $x \neq y$ **and** $x \notin vars\ e2$ **and** $y \notin vars\ e1$
shows $(x ::= e1;;\ y ::= e2) \sim (y ::= e2;;\ x ::= e1)$

Hint: Remember lemma *assign_simp*. You might want to add a corresponding lemma for *Seq*.

Task 2 (15 marks) Define a function *csimp* :: *com* ⇒ *com* that simplifies all arithmetic and Boolean expressions in commands using *asimp* and *bsimp*.

Show that *csimp* preserves the behaviour of commands:

lemma *csimp_sim*: $csimp\ c \sim c$

Hint: Remember lemma *sim_while_cong* for proving the WHILE case. You might want to add a similar lemma that also allows changing the loop condition to an equivalent one.

Task 3 (10 marks) Now define a function *csimp_full* :: *com* ⇒ *com* that not only simplifies expressions, but also does some dead code elimination in cases where Boolean expressions can be simplified to constants, like in the example below, and show that this function still preserves the behaviour of commands.

lemma $csimp\_full\ (IF\ (Less\ (N\ 0)\ (N\ 1))\ THEN\ x ::= N\ 0\ ELSE\ x ::= N\ 1) = (x ::= N\ 0)$

lemma *csimp_full_sim*: $csimp\_full\ c \sim c$

Task 4 (5 marks) Modify theory *AExp* by adding a unary negation constructor *Neg* to *aexp*, so that subtraction can be expressed using *Plus* and *Neg*:

abbreviation *Minus e1 e2* ≡ *Plus e1* (*Neg e2*)

Update the definitions of *aval* and *asimp* to handle *Neg* and fix any broken proofs, then show that *Minus* behaves as expected:

lemma *Minus_correct*: *aval* (*Minus e1 e2*) *s* = *aval e1 s* − *aval e2 s*

Add an abbreviation *Equal* that checks for equality of *aexp* expressions using existing constructors, and show that it is correct:

lemma *Equal_correct*: *bval* (*Equal e1 e2*) *s* ⟷ *aval e1 s* = *aval e2 s*

Task 5 (15 marks) Consider the following implementation of Euclid's algorithm for calculating the greatest common divisor (using the *Minus* and *Equal* expressions defined above):

abbreviation
imp_gcd ≡
(*WHILE* (*Not* (*Equal* (*V* ''*a*'') (*V* ''*b*''))) *DO*
IF (*Less* (*V* ''*a*'') (*V* ''*b*''))
THEN ''*b*'' ::= *Minus* (*V* ''*b*'') (*V* (''*a*''))
ELSE ''*a*'' ::= *Minus* (*V* ''*a*'') (*V* (''*b*'')))

Show its (partial) correctness:

lemma *imp_gcd_partial_correctness*:
assumes (*imp_gcd*, *s*) ⇒ *t* **and** *s* ''*a*'' > *0* **and** *s* ''*b*'' > *0*
shows *gcd* (*s* ''*a*'') (*s* ''*b*'') = *t* ''*a*''

Task 6 (10 marks, advanced) Show that *imp_gcd* terminates:

lemma *imp_gcd_terminates*:
assumes *s* ''*a*'' > *0* **and** *s* ''*b*'' > *0*
shows ∃ *t*. (*imp_gcd*, *s*) ⇒ *t*

Hint: You might want to prove a lemma about the termination of while loops. Induction rules like *measure_induct_rule* might be useful, which allows induction using a measure function *f* :: *'a* ⇒ *nat*. The function *nat* for converting from *int* to *nat* might also be useful.

Task 7 (10 marks, advanced) Modify theory *Com* by extending the type *com* with a nondeterministic choice command *CHOOSE x vs* that sets variable x to a value chosen from a list of values that may depend on the state, i.e. *vs* has type *state* $\Rightarrow$ *val list*.

Fix the existing proofs in theories *Big_Step* and *Small_Step*, including the equivalence proof between big-step and small-step semantics, but excluding the proofs that the semantics are deterministic and the proof about final configurations (those have been removed from the versions of the theories coming with this assignment).

As an example, show that the following refinement holds:

```
abbreviation refines :: com ⇒ com ⇒ bool (infix ⊑ 50) where
  c1 ⊑ c2 ≡ (∀ s t. (c1, s) ⇒ t ⟶ (c2, s) ⇒ t)

lemma (x ::= (N 2)) ⊑ (CHOOSE x (λ_. [0, 1, 2]))
```

Now consider the following specification for computation of the greatest common divisor:

```
abbreviation
  imp_gcd_spec ≡
    CHOOSE ''a'' (λs. [gcd (s ''a'') (s ''b'')]);;
    ''b'' ::= V ''a''
```

Find a context $C :: com \Rightarrow com$ under which *imp_gcd* refines *imp_gcd_spec* and prove the lemma:

```
lemma C imp_gcd ⊑ C imp_gcd_spec
```