Proof Assistants - Project Report

Mathis Bouverot-Dupuis

November 2023

This is the report for the programming project of the course "Proof Assistants" (MPRI 2.7.1). I give a high-level overview of my design decisions : see the accompanying pdf and Coq files for the project instructions and detailed proofs.

1 Exercise 1

Overall there were no major difficulties in this exercise, and my proofs follow the instructions very closely. Here are the main ways in which I differed from the instructions :

- After proving the weakening lemma (1.1.c), I prove several specialized versions that do not require the user to provide a proof of `incl _ _`. I do the same every time I prove a version weakening.
- Before proving any substantial property of natural deduction, I prove some common rules that are consequences of the base rules : see the lemmas `nd_revert`, `nd_feed` and `nd_apply`.
- After defining the three-point semantics for propositional logic, I prove a collection of trivial lemmas about the `leq` operator on truth values. This allows for a relatively simple proof of consistency.

2 Exercise 2

The main technical difficulty I faced in this exercise was that Heyting Algebras are only preorders, not partial orders : the order relation is not antisymmetric. As a practical consequence, many lemmas which we would expect to prove `x = y` can only prove results of the form `x <= y` $\wedge$ `y <= x`, which I write `x <=> y`.

Although `<=>` is not strict equality, it is nonetheless an equivalence relation : I thus use setoid rewriting to facilitate proof development. This leads to numerous "compatibility" lemmas to show that the main operations on heyting algebras (`hle`, `hmeet`, `himp`, . . .) are compatible with the `<=>` relation : see the instance declaration `hmeet_compat` for an example. In the end it was well worth it to be able to rewrite using `<=>`.

I also prove a fairly complete set of trivial lemmas about the heyting algebra operations `hle`, `hmeet` and `himp`, which facilitated the subsequent (harder) proofs.

After this preparatory work, the proof of soundness and completeness of the semantics was fairly straightforward.

3 Exercise 3

The following is a high-level description of my tactic `decide_bool` on a goal of the form L = R :

1. Reify the formulas L and R, in such a way that `eval_bool (reify L)` is definitionally equal to L (and similarly for R).
2. Use the `change` tactic to replace the goal with `eval_bool (reify L) = eval_bool (reify R)`.
3. Rewrite using the lemma `evalB_evalZ` to change the goal to something like `eval_int L' = eval_int R'`.
4. Invoke `lia` to solve the goal.

This is enough to solve simple goals such as `a && b = b && a` or `a || true = true`, but fails to solve `a && a = a`. Let us inspect why this is the case. For this equation, `lia` is invoked on the following goal :

```
Z.b2z a * Z.b2z a = Z.b2z a
```

There are two issues here :

- This is not a linear equation. All we can do is use `nia` instead of `lia`, and hope it can deal with the non-linearity.
- We need to rely on the fact that Z.b2z a is either 0 or 1. I thus added a step before invoking `lia`/`nia`, which adds a hypothesis of the form `0 <= Z.b2z b <= 1` for suitable boolean terms `b`.

It was also straightforward to extend the tactic to deal with universal quantifications and implications in the goal, and with hypotheses of the form `a = b` where `a` and `b` are boolean terms.