FPC-Coq: Using ELPI to elaborate external proof evidence into Coq proofs

Roberto Blanco[1], Matteo Manighetti[2], and Dale Miller[2]

[1] Inria Paris
[2] Inria Saclay

Abstract

Logic programming implementations of the *foundational proof certificate* (FPC) framework are capable of checking a wide range of proof evidence. Proof checkers based on logic programming can make use of both unification and backtracking search to allow varying degrees of proof reconstruction to take place during proof checking. Such proof checkers are also able to elaborate proofs lacking full details into proofs containing much more detail. We outline here our use of the Coq-Elpi plugin, which embeds an implementation of λProlog into Coq, to check proof certificates supplied by external (to Coq) provers and to elaborate them into the fully detailed proof terms that are needed for checking by the Coq kernel.

The trusted base of Coq is its kernel, which is a type-checking program that certifies that a dependently typed λ-term has a given type. If type checking succeeds, we accept that the formula corresponding to that type is, in fact, a theorem of intuitionistic logic. The rest of the Coq system, especially its tactic language, is designed to help a human user build proofs-cum-λ-terms that can be checked by the kernel. Of course, there are many theorem provers for intuitionistic logic for which a completed proof is not the kind of detailed λ-term required by the Coq kernel but is, instead, a trace of some key aspects of a proof: some proof details might not be captured in such a trace. For example:

1. Substitution instances of quantifiers might not be recorded in a proof since such instances can, in principle, be reconstructed using unification.

2. Detailed typing information might not need to be stored within a proof certificate since types can often be reconstructed during proof checking.

3. Some simplifications steps might be applied within a proof without recording which rewrites were used. A simple non-deterministic proof-search engine might be expected to reconstruct an equivalent simplification.

The Foundational Proof Certificate (FPC) framework [4] has been designed for formally defining a rich collection of proof certificates that do not necessarily contain full details. Proof checkers for FPC certificates can be written using simple logic programs: the standard implementation mechanisms of unification and backtracking search can then be used to reconstruct missing proof details [10]. Such proof checking kernels can also be used to elaborate such certificates into fully detailed proofs [2, 3].

We built the FPC-Coq system to demonstrate the feasibility of using logic programming to check proof certificates produced by external provers and to elaborate such certificates into proofs with sufficient details that they can be given directly to Coq's kernel to check. Note that one does not need to trust the external prover nor the FPC-Coq tool-chain: one only needs to trust the Coq kernel.

Because of the need to represent and compute with quantificational formulas and dependently-typed λ-terms, we chose λProlog [11] as our implementation language since it has a direct and elegant treatment of binding in data structures. The ELPI implementation [5] of λProlog has been embedded recently into the Coq-Elpi plugin [13], and we have used this plugin to implement a system that checks proof certificates in the FPC framework and elaborates them into proof terms that can be submitted to the Coq kernel.

The FPC-Coq system can be seen as part of the larger effort to use λProlog within a type theory setting [6], in general, and within Coq [12], in particular. In doing so, the first step is to define how to consume Coq terms as proof evidence for a formula, i.e., to program a type checker in the FPC framework. Initially, the presentation of connectives as inductive types in Coq is translated to the usual connectives of intuitionistic first-order logic, the only ones known to the FPC checker.

The current version of FPC-Coq only works on proving theorems in first-order intuitionistic logic. Given that the λProlog proof checker, based on *LJF* (a focused version of intuitionistic sequent calculus), can also serve as a proof checker for *LKF* (its classical counterpart), FPC-Coq can be used to prove

double-negation translations of a formula for which there is a proof certificate in classical logic [4]. Extending the FPC framework to involve inductive and coinductive reasoning has also been considered [7, 8], and a future version of FPC-Coq could well include proof certificates that are output from inductive theorem provers as well as model checkers.

Given its origins in the theory of focused sequent calculus, our first FPC proof checker only implements a minimalistic sequent calculus in a purely declarative fashion. We have also developed a second approach to building a proof checker based on dependent types in the style of [9]. This second approach allows for a much more concise implementation, where there is no translation needed from Coq formulas to intuitionistic formulas since the kernel directly operates with dependent types. The usual definitions of proof certificates remain directly usable also in this context.

In particular, when the user loads the λProlog code for an FPC definition named, say `fpc`, into the system, that FPC definition is made available as a Coq-Elpi tactic that takes a proof certificate, say `cert`, as an argument. Inside Coq's proof mode, that tactic can be invoked by `elpi fpc cert`.

The code for both of these implementations is available at `https://github.com/proofcert/fpc-elpi`.

Although FPC-Coq currently targets the construction of proofs for the Coq kernel, it should be straightforward to use the same setup described here to build Dedukti proofs as well [1]. Our development could also provide the facilities for cleanly and easily building Coq tactics based on proof certificates and written in λProlog.

References

[1] Ali Assaf, Guillaume Burel, Raphal Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Expressing theories in the λΠ-calculus modulo theory and in the Dedukti system. In *TYPES: Types for Proofs and Programs*, Novi Sad, Serbia, 2016.

[2] Roberto Blanco. *Applications for Foundational Proof Certificates in theorem proving*. PhD thesis, Ecole Polytechnique, November 2017.

[3] Roberto Blanco, Zakaria Chihani, and Dale Miller. Translating between implicit and explicit versions of proof. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 255–273. Springer, 2017.

[4] Zakaria Chihani, Dale Miller, and Fabien Renaud. A semantic framework for proof evidence. *J. of Automated Reasoning*, 59(3):287–330, 2017.

[5] Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. ELPI: fast, embeddable, λProlog interpreter. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *LNCS*, pages 460–468. Springer, 2015.

[6] Ferruccio Guidi, Claudio Sacerdoti Coen, and Enrico Tassi. Implementing type theory in higher order constraint logic programming. *Mathematical Structures in Computer Science*, 29(8):1125–1150, 2019.

[7] Quentin Heath and Dale Miller. A framework for proof certificates in finite state exploration. In Cezary Kaliszyk and Andrei Paskevich, editors, *Proceedings of the Fourth Workshop on Proof eXchange for Theorem Proving*, number 186 in Electronic Proceedings in Theoretical Computer Science, pages 11–26. Open Publishing Association, August 2015.

[8] Quentin Heath and Dale Miller. A proof theory for model checking. *J. of Automated Reasoning*, 63(4):857–885, 2019.

[9] Stéphane Lengrand, Roy Dyckhoff, and James McKinna. A focused sequent calculus framework for proof search in pure type systems. *Logical Methods in Computer Science*, 7(1), 2011.

[10] Dale Miller. Proof checking and logic programming. *Formal Aspects of Computing*, 29(3):383–399, 2017.

[11] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.

[12] Enrico Tassi. Deriving Proved Equality Tests in Coq-Elpi: Stronger Induction Principles for Containers in Coq. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[13] Enrico Tassi. Coq plugin embedding ELPI. `https://github.com/LPCIC/coq-elpi`, 2020.