Linking a λProlog proof checker to the Coq kernel An extended abstract

Roberto Blanco
INRIA Paris, France

Matteo Manighetti and
INRIA Saclay
LIX, École Polytechnique, France

Dale Miller
INRIA Saclay
LIX, École Polytechnique, France

Synopsis. The Coq kernel works with a specific form of proof structure, and that proof structure is fully detailed. In particular, the Coq kernel performs type-checking on a dependently typed λ-term. If type checking succeeds, the formula corresponding to its type is, in fact, a theorem of intuitionistic logic. Most external-to-Coq theorem provers do not generally build such detailed, dependently typed λ-terms for proofs. To the extent that theorem provers output proof objects (called *proof certificates* here), their structure can vary a great deal. Also, since some details can be reconstructed, they are seldom traced and inserted into proof certificates. In this extended abstract, we describe the design of the FPC-Coq system that can take externally generated proof certificates and elaborate them into the kind of proof structures required by the Coq kernel. This *elaboration* system is built using three technologies: (1) the foundational proof certificate (FPC) framework [3] that allows for the specification of a wide range of proof certificates, (2) the λProlog programming language [9] that can interpret a given FPC definition and then perform both proof checking and proof reconstruction [1, 2], and (3) the Coq-Elpi plugin [10], which embeds the Elpi implementation [4] of λProlog into Coq.

Consider an external (to Coq) prover capable of exporting its proofs as certificates in some particular format. Such a theorem prover (for intuitionistic logic) is unlikely to build the detailed λ-term required by the Coq kernel. More likely, its notion of a proof certificate 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 FPC framework can be used to formally define the proof evidence of many different formats [3] including those with such details missing. Combining such a definition, which is a simple logic program, with an FPC proof checking kernel yields a complete logic programming proof kernel. During proof checking, unification and backtracking search can infer missing proof details. The FPC framework also allows for the convenient capture and reorganization of such details and, as a result, it can be used to build proof structures appropriate for the Coq kernel.

Given its origins in the theory of focused sequent calculus, our first FPC proof checker implements a purely declarative sequent calculus proof checker, along the lines described in [3]. We have also developed a second approach to building a proof checker based on dependent types in the style of [7]. This second approach allows for a more concise implementation, where there is no translation needed

Submitted to:
LFMTP 2020

from Coq formulas to intuitionistic formulas since the kernel directly operates with dependent types. The usual FPC definitions of proof certificates remain directly usable also in this context.

The code for both of these implementations is available at `https://github.com/proofcert/fpc-elpi`. It is important to recognize that both λProlog and Coq-Elpi can be seen as parts of a tool chain, connecting an external prover to the Coq kernel and that these two pieces of technology do not need to be trusted by the Coq kernel.

The current version of FPC-Coq only works on proving theorems in first-order intuitionistic logic. We also assume that external proof certificates are in a format that is formally defined by some given FPC: examples of such formats which have already been developed are resolution refutations, Hilbert proofs, de Bruijn notation, and natural deduction [3]. Given that the λProlog proof checker internally implements *LJF* (a focused version of intuitionistic sequent calculus [8]), it can also serve as a proof checker for *LKF* (its classical counterpart). As a result, FPC-Coq can be used to prove double-negation translations of a formula for which there is a proof certificate in classical logic [3]. Extending the FPC framework to involve inductive and coinductive reasoning has also been considered [5, 6], and a future version of FPC-Coq could include proof certificates that are output from inductive theorem provers as well as model checkers.

References

[1] Roberto Blanco (2017): *Applications for Foundational Proof Certificates in theorem proving*. Ph.D. thesis, Université Paris-Saclay. Available at `https://tel.archives-ouvertes.fr/tel-01743857`.

[2] Roberto Blanco, Zakaria Chihani & Dale Miller (2017): *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*, *Lecture Notes in Computer Science* 10395, Springer, pp. 255–273, doi:10.1007/978-3-319-63046-5_16.

[3] Zakaria Chihani, Dale Miller & Fabien Renaud (2017): *A semantic framework for proof evidence*. *J. of Automated Reasoning* 59(3), pp. 287–330, doi:10.1007/s10817-016-9380-6. Available at `https://doi.org/10.1007/s10817-016-9380-6`.

[4] Cvetan Dunchev, Ferruccio Guidi, Claudio Sacerdoti Coen & Enrico Tassi (2015): *ELPI: Fast, Embeddable, λProlog Interpreter*. In Martin Davis, Ansgar Fehnker, Annabelle McIver & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015*, *LNCS* 9450, Springer, pp. 460–468, doi:10.1007/978-3-662-48899-7_32.

[5] Quentin Heath & Dale Miller (2015): *A framework for proof certificates in finite state exploration*. In Cezary Kaliszyk & Andrei Paskevich, editors: *Proceedings of the Fourth Workshop on Proof eXchange for Theorem Proving*, *Electronic Proceedings in Theoretical Computer Science* 186, pp. 11–26, doi:10.4204/EPTCS.186.4.

[6] Quentin Heath & Dale Miller (2019): *A proof theory for model checking*. *J. of Automated Reasoning* 63(4), pp. 857–885, doi:10.1007/s10817-018-9475-3.

[7] Stéphane Lengrand, Roy Dyckhoff & James McKinna (2011): *A Focused Sequent Calculus Framework for Proof Search in Pure Type Systems*. *Logical Methods in Computer Science* 7(1). Available at `http://www.lix.polytechnique.fr/~lengrand/Work/Reports/TTSC09.pdf`.

[8] Chuck Liang & Dale Miller (2009): *Focusing and Polarization in Linear, Intuitionistic, and Classical Logics*. *Theoretical Computer Science* 410(46), pp. 4747–4768, doi:10.1016/j.tcs.2009.07.041.

[9] Dale Miller & Gopalan Nadathur (2012): *Programming with Higher-Order Logic*. Cambridge University Press, doi:10.1017/CBO9781139021326.

[10] Enrico Tassi (2020): *Coq plugin embedding ELPI*. `https://github.com/LPCIC/coq-elpi`.