A Semantics for Proof Evidence

Zakaria Chihani, Dale Miller, and Fabien Renaud
INRIA and LIX, Ecole Polytechnique, Palaiseau, France

October 1, 2013

Theorem provers are generally complex systems that search for proofs using some combination of automatic and interactive tools. Given their nature, it is important for them to be formally correct. Given their complexity, they are extremely hard to certify as such. One can arrange, however, for provers to be "certifying" in the sense that they can be built to output evidence of the proofs they achieved and then employ an independent proof checker to check them. Some proof systems, such as Coq, use this certifying approach by employing a trusted kernel to check proofs built outside the kernel [2].

In the ProofCert project, we are exploring to what extent we might be able to accommodate a wide range of proof evidence from various provers and then to have independent checkers certify such proof evidence. We have identified three different stages to achieving such a proof checking scheme [4].

1. The implementers of theorem provers must describe their proof evidence in some textual form. Presumably, such documents are roughly the result of "pretty printing" the evidence that they have collected. For example, the implementer of a resolution prover might output a numbered list of clauses as well as a list of triples (indicating which two clauses resolve to form a third clause). Such documents will be called *proof certificates*. One expects that there will be a great many kinds of proof certificate formats that are ultimately created and used.

2. A general framework for defining the semantics of proof evidence must be designed. In such a framework, the format and structure of proof certificates would be given a clear and precise semantics. Such a semantic framework will need to be sufficiently low-level so as to capture the essence of a proof and general enough to accommodate a wide range of proof systems. In the resolution prover example, the relationship of "resolvent" must be defined in terms of more basic inference rules of proof.

3. Trusted *proof checkers* must be implemented to execute the semantic descriptions of proof certificates. We need to be able to trust that a successful execution of the proof checker on a given certificate implies that the certificate does, in fact, elaborate into a recognized formal proof.

We shall show in this talk how focused sequent calculus proof systems, which are now available for linear, intuitionistic, and classical logics [1, 5, 6], can be used to address the second of these stages: i.e., they provide a flexible framework for defining the "semantics of proof evidence".

A simple analogy in the area of programming languages is worth pointing out for this three stage organization. (1) There are many kinds of programming languages and researchers are routinely designing new ones. (2) In order to define such programming languages precisely (for, say, mathematical treatment or for implementations), the semantics of such programming languages need to be given: a popular form of such semantic specifications is Structural Operational Semantics (SOS) [8]. (3) Finally, compliant interpreters and compilers for a given programming language can be built based on such semantic descriptions.

This analogy can be pushed an additional step. Since an SOS specification is given as a set of simple inference rules, a general purpose interpreters for SOS specifications can be given using logic programming languages [3, 7]. Similarly, since our evidence of proof is based on the generation of sequent calculus proofs, logic programming (particularly the more expressive and abstract form available in λProlog [7]) can be used to provide a reference interpreter for checking proof certificates.

References

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

[2] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004.

[3] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. Centaur: the system. In *Third Annual Symposium on Software Development Environments (SDE3)*, pages 14–24, Boston, 1988.

[4] Zakaria Chihani, Dale Miller, and Fabien Renaud. Foundational proof certificates in first-order logic. In Maria Paola Bonacina, editor, *CADE 24: Conference on Automated Deduction 2013*, LNAI 7898, pages 162–177, 2013.

[5] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46), 2009.

[6] Chuck Liang and Dale Miller. A focused approach to combining logics. *Annals of Pure and Applied Logic*, 162(9):679–697, 2011.

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

[8] Gordon Plotkin. A structural approach to operational semantics. DAIMI FN-19, Aarhus University, Aarhus, Denmark, September 1981.