Structural Proof Theory and Logic Programming An extended abstract

Dale Miller
Inria & LIX/Ecole Polytechnique

The semantics of logic programming languages, particularly those based on first-order Horn clauses, have traditionally been given a denotational semantics using model theory and an operational semantics using SLD-resolution. Connecting these two different forms of semantic description has been satisfying, since it reassures us that the language is not *ad hoc*, and productive, since it supports new language designs based on, for example, stable models and constraints.

The theory of proof, particularly the approach developed by Gentzen in the 1930s and by Girard in the 1980s, is a different and appealing framework for developing computational logic. Employing proof theory as a framework for logic programming has at least two significant benefits.

First, proof theory—mainly, the theory of natural deduction proofs—has been used to describe the foundations of functional programming (via the Curry-Howard Correspondence). Thus from the proof theory point-of-view, a clear difference between these paradigms appears: functional programming can be seen as *proof-normalization* and logic programming as *proof-search*. The role of cut-elimination is different in these two programming paradigms: cut-elimination can be used in functional programming to describe computation steps, while it can be used to reason about computation in logic programming.

Second, proof theory provides a framework for extending the role of logical connectives and quantifiers in logic programs, thus allowing for much more expressive logic programs than those defined using first-order classical logic.

This talk will focus on the second of these benefits.

Proof theory supports extensions to logic programming

The first such extensions to the Horn clause framework for which proof-search was systematically studied involved allowing both hypothetical and universally quantified goal formulas. Such extensions were explored from the proof-search perspective nearly simultaneously by Gabbay & Reyle [6], Paulson [21], McCarty [13], Miller [14], and Hallnäs & Schroeder-Heister [9]. Once these extensions were understood within the structural proof theory of intuitionistic logic (using the technical notion of *uniform proofs* [20]), it was natural to make additional extensions using higher-order quantification [20, 21] and linear logic [1, 7, 16].

These extensions to logic programming have helped to increase the influence of logic programming ideas, techniques, and tools. For example, embracing linear logic within logic programming provided new means for specifying the operational semantics of imperative and object-orient programming languages [15]. Also, higher-order quantification in logic programming has had a significant impact in those areas where researchers need to manipulate syntax containing binding structures, e.g., the syntax of programs and quantificational formulas. There are many implementations of bindings in term structures, ranging from using De Bruijn numerals to nominal logic techniques. Proof theory, however, provides a different means for computing with binding structures in which bindings are allowed to move. For example, a term-level λ-binding can move to be a formula-level quantifier, and the latter can move again

© Dale Miller
This work is licensed under the
Creative Commons Attribution License.

to be an eigenvariable (a proof-level binding device introduced by Gentzen). This *mobility of binders* approach, which requires some extension to unification in logic programming [17], is now a popular device for implementing logical frameworks and meta-level reasoning systems, such as λProlog [19], Twelf [22], and the Abella theorem prover [2].

Logic programming has driven innovations in proof theory

By taking *proof search* as a serious computational principle, proof theoreticians have been lead to develop new proof-theoretic notions [18]. In particular, *focusing* and *polarization* [1, 8] are a significant extension to the earlier idea of uniform proofs [12]. With these innovations, we can return to Kowalski's equation $Algorithm = Logic + Control$ [11] and give a completely proof-theoretical explanation of the difference between bottom-up and top-down search within Horn clause programs using polarity and focusing [5].

Links between the logic programming paradigm and proof theory continue to be developed. For example, there is the recent work on cyclic proofs and coinductive logic programming [3, 4] as well as a new foundation for viewing model checking as proof search within an extension of (linear) logic using least fixed points [10].

References

[1] Jean-Marc Andreoli (1992): *Logic Programming with Focusing Proofs in Linear Logic*. J. of Logic and Computation 2(3), pp. 297–347, doi:10.1093/logcom/2.3.297.

[2] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu & Yuting Wang (2014): *Abella: A System for Reasoning about Relational Specifications*. Journal of Formalized Reasoning 7(2), pp. 1–89, doi:10.6092/issn.1972-5787/4650.

[3] Henning Basold, Ekaterina Komendantskaya & Yue Li (2019): *Coinduction in Uniform: Foundations for Corecursive Proof Search with Horn Clauses*. In Luís Caires, editor: 28th European Symposium on Programming, ESOP 2019, LNCS 11423, Springer, pp. 783–813, doi:10.1007/978-3-030-17184-1_28.

[4] James Brotherston, Nikos Gorogiannis & Rasmus Lerchedahl Petersen (2012): *A Generic Cyclic Theorem Prover*. In: APLAS, LNCS 7705, Springer, pp. 350–367, doi:10.1007/978-3-642-35182-2_25.

[5] Kaustuv Chaudhuri, Frank Pfenning & Greg Price (2008): *A Logical Characterization of Forward and Backward Chaining in the Inverse Method*. J. of Automated Reasoning 40(2-3), pp. 133–177, doi:10.1007/s10817-007-9091-0.

[6] D. M. Gabbay & U. Reyle (1984): *N-Prolog: An Extension of Prolog with Hypothetical Implications. I.* Journal of Logic Programming 1, pp. 319–355.

[7] Jean-Yves Girard (1987): *Linear Logic*. Theoretical Computer Science 50(1), pp. 1–102, doi:10.1016/0304-3975(87)90045-4.

[8] Jean-Yves Girard (1991): *A new constructive logic: classical logic*. Math. Structures in Comp. Science 1, pp. 255–296, doi:10.1017/S0960129500001328.

[9] Lars Hallnäs & Peter Schroeder-Heister (1990): *A Proof-Theoretic Approach to Logic Programming. I. Clauses as rules*. J. of Logic and Computation 1(2), pp. 261–283.

[10] 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.

[11] R. Kowalski (1979): *Algorithm = Logic + Control*. Communications of the Association for Computing Machinery 22, pp. 424–436.

[12] 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.

[13] L. T. McCarty (1988): *Clausal Intuitionistic Logic I. Fixed Point Semantics*. Journal of Logic Programming 5, pp. 1–31.

[14] Dale Miller (1989): *A logical analysis of modules in logic programming*. Journal of Logic Programming 6(1-2), pp. 79–108.

[15] Dale Miller (1996): *Forum: A Multiple-Conclusion Specification Logic*. Theoretical Computer Science 165(1), pp. 201–232, doi:10.1016/0304-3975(96)00045-X.

[16] Dale Miller (2004): *Overview of linear logic programming*. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet & Phil Scott, editors: Linear Logic in Computer Science, London Mathematical Society Lecture Note 316, Cambridge University Press, pp. 119–150.

[17] Dale Miller (2019): *Mechanized Metatheory Revisited*. Journal of Automated Reasoning 63(3), pp. 625–665, doi:10.1007/s10817-018-9483-3.

[18] Dale Miller (2019): *Reciprocal influences between logic programming and proof theory*. Philosophy & Technology, doi:10.1007/s13347-019-00370-x.

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

[20] Dale Miller, Gopalan Nadathur, Frank Pfenning & Andre Scedrov (1991): *Uniform Proofs as a Foundation for Logic Programming*. Annals of Pure and Applied Logic 51(1–2), pp. 125–157.

[21] Lawrence C. Paulson (1986): *Natural Deduction as Higher-Order Resolution*. Journal of Logic Programming 3, pp. 237–258.

[22] Frank Pfenning & Carsten Schürmann (1999): *System Description: Twelf — A Meta-Logical Framework for Deductive Systems*. In H. Ganzinger, editor: 16th Conf. on Automated Deduction (CADE), LNAI 1632, Springer, Trento, pp. 202–206, doi:10.1007/3-540-48660-7_14.