λProlog:
An Extended Logic Programming Language

Amy Felty,[1] *Elsa Gunter,*[1] *John Hannan,*[1]
Dale Miller,[1] *Gopalan Nadathur,*[2] *Andre Scedrov*[3]

[1]Computer and Information Science, University of Pennsylvania
[2]Computer Science, Duke University
[3]Mathematics, University of Pennsylvania

The logic programming language λProlog is an extension of conventional Prolog in several different directions. These extensions provide higher-order functions, λ-terms, a polymorphic typing discipline, modules, and a mechanism for providing secure abstract datatypes. Our original goal in developing λProlog was to understand the essential logical and proof theoretic nature of these extensions. This work has led us to describe a class of formulas called *higher-order hereditary Harrop formulas* which play a role in λProlog that is similar to the role of positive Horn clauses in Prolog. This extended class of formulas permits stronger forms of logical reasoning than can be found in Prolog. For example, it allows universal quantification and implications into the bodies of program clauses as well as some forms of higher-order quantification. Higher-order hereditary Harrop formulas, therefore, significantly extend positive Horn clauses, and, consequently, λProlog significantly enriches Prolog.

Part of our efforts have also been directed at understanding how this enrichment can be used to write programs. In this direction we have been investigating how the mechanisms of λProlog can be used to implement theorem provers, program transformers, and natural language understanding systems.

We have also implemented most features of higher-order hereditary Harrop formulas in a prototype interpreter, called λProlog V2.6. This interpreter comprises roughly 4100 lines of C-Prolog code and has been distributed to about 30 sites in North America and Europe since August 1987. The performance of this interpreter leaves much to be desired, owing, in part, to the underlying implementation language and, to a much larger extent, to the fact that efficiency has not been a major concern in this experimental version. Despite this drawback, the system has been used by us and others for serious experimentation and prototype implementations. Two ongoing Ph.D. theses make use of it as their primary implementation language.

Below is an outline of the major research aspects of this effort to date.

Logic programming foundations The theory of higher-order Horn clauses is outlined in [6] and given in detail in [9]. The higher-order unification process of [2] plays a central role in this analysis. The first-order theory of a language which includes implications in the bodies of program clauses is contained in [4]. Higher-order hereditary Harrop formulas were introduced in [8] to describe program clauses which are both higher-order and contain

implications and universal quantification in clause bodies.

Modules for logic programming The extended use of implications and universal quantifiers permits notions of modules and abstract datatypes to be supported. See [4] and [8].

Theorem prover implementation λProlog has proved to be a very natural implementation environment for writing tactic style theorem provers [1].

Program transformation Extending the work of [3], we have experimented with implementing several program transformation algorithms. The relationship between the actual implementation code in λProlog and the semantics of the programs being transformed seems very tight. We hope this will permit us to establish formal correctness proofs for these transformers [7].

Computational linguistics Natural language understanding systems must bring together syntactic and semantic processing. Very often syntactic processing can be identified with first-order operations while semantic processing is often higher-order. λProlog seems to offer a good environment for bringing these two kinds of processing together [5].

References

[1] A. Felty and D. Miller, Specifying Theorem Provers in a Higher-Order Logic Programming Language. Ninth International Conference on Automated Deduction, 23 – 26 May 1988, Argonne Ill.

[2] G. P. Huet, A Unification Algorithm for Typed λ-Calculus. Theoretical Computer Science 1, 1975, 27 – 57.

[3] G. P. Huet and B. Lang, Proving and Applying Program Transformations Expressed with Second-Order Patterns. Acta Informatica 11, (1978), 31 – 55.

[4] D. Miller, A Logical Analysis of Modules for Logic Programming. To appear in the Journal of Logic Programming.

[5] D. Miller and G. Nadathur, Some Uses of Higher-Order Logic in Computational Linguistics. Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, 1986, 247 – 255.

[6] D. Miller and G. Nadathur, Higher-Order Logic Programming. Proceedings of the Third International Logic Programming Conference, London, June 1986, 448 – 462.

[7] D. Miller and G. Nadathur, A Logic Programming Approach to Manipulating Formulas and Programs. IEEE Symposium on Logic Programming, San Franciso, September 1987.

[8] D. Miller, G. Nadathur, and A. Scedrov, Hereditary Harrop Formulas and Uniform Proofs Systems. Second Annual Symposium on Logic in Computer Science, Cornell University, June 1987, 98 — 105.

[9] G. Nadathur, A Higher-Order Logic as the Basis for Logic Programming. Ph.D. Dissertation, University of Pennsylvania, May 1987.