Reflections on the Foundations of Meta-Programming: Is a Type Theory Needed?

François Bry
ECRC, Arabellastraße 17, W 8000 München 81, Germany
bry@ecrc.de

Meta-programming is an important programming technique, which is widely applied in logic programming - cf. e.g. [1,2,3,4,5,6]. However, in spite of several studies - cf. e.g. [28,29,30] - the semantics of meta-programs and their formalization in logic remain open issues. Formalizations in classical logic, such as that subjacent to the Gödel language [31], are in the spirit of the classical formalization of first-order programs - cf. e.g. [8,9]. They interpret meta-predicates and predicate variables as higher-order symbols. How attractive they might be, the formalizations in classical logic are not conform to a programmer's intuition. Moreover, although such approaches rather convincingly formalize constructs such as Prolog "call" and "clause", they generally fail to account for constructs permitting a dynamic creation of predicate symbols and atoms, such as Prolog "univ" (=..), "functor", and "arg". The formalizations of meta-programming based on many-sorted logics, e.g. [10], are more intuitive. Nevertheless, they also fail to explain the dynamic creation of symbols.

We argue that the inadequacy of classical logic to formalize meta-programming is related to the theory of types of this logic, i.e. to the principles that were introduced first by Russel in [11].[1] We argue that the standard models of meta-programs viewed as higher-order theories in classical logic are inadequate to describe the intended meaning of these programs. We propose two alternative approaches to formalizing meta-programs. The first approach, which we call "Russel-Henkin semantics" keeps Russelian proof and types theories, but reconsider the model theory. The second approach, which we call "Frege-Henkin semantics" is more drastic: it reconsider the proof theory by discarding any notion of "order" or type theory.

The first approach consists in keeping a conventional proof theory and in reconsidering the model theory. More precisely, the conventional, Russelian theory of types is kept, but the models of higher-order theories are redefined. Instead of standard models, we consider non-standard models of the kind introduced by Henkin in [16]. In such models, terms of higher orders do not range over all sets of individuals of lower orders, but only over certain sets, namely those sets that can be described as (or constructed from) formulas. We think that such a restriction of the interpretation of higher-order terms reflects a programmer's intuition. We argue that the intended models of meta-programs are Henkin models, in the same way as the intended models of first-order programs are Herbrand models [9]. The Russel-Henkin semantics conveys, we think, the intuition of "amalgamation" [7] as well as of naming and ground representations [31]. We are not aware of any previous proposal to formalize the semantics of meta-programs in terms of non-standard, Henkin models. Defining the semantics of meta-programs in terms of Henkin models is, up to unsubstantial differences, similar

[1] Although programming language types are formalized in terms of logical theories of types - cf. [14] - the two notions are distinct. The "types" we refer to here are the "orders" of the hierarchical stratification of the universe of discourse into individuals, sets and relations of individuals, sets of sets, etc.

to formalizing meta-programs in terms of many-sorted formalisms. Such approaches have been considered in [10,31].

The second approach, which we call "Frege-Henkin semantics" considers a proof theory without any type theory, like Frege's *Begriffsschrift* [20,21,33] - a precursor of classical logic - and non-standard models à la Henkin. Terms and individual are thus no more hierarchically stratified into successive "orders" or "types". This approach in fact re-formalizes HiLog [16]: types can be considered absent from HiLog because every symbols of this language belong to every types. Formalizing meta-programs as theories in a logic without theory of types means that no syntactical distinction is made e.g. between terms interpreted as individuals and terms interpreted as relations. The Frege-Henkin semantics gives rise to treating formulas as terms. Therefore, programming languages based on this semantics need a construct similar to Lisp "quote", an explicit quantification, or equivalently, a "ground representation" as proposed in [31].

A semantics of meta-programs without "orders" or "types" does not prevent to reintroduce types for other purposes. If types are not necessary for formalizing meta-programs and meta-programming techniques, they might well be desirable, for example for an easy application of these techniques. As a matter of fact, extensions of the type-free system HiLog with "programming language types" have been proposed [17,18].

Both, the Russel-Henkin and the Frege-Henkin semantics, are convenient to formalizing common meta-interpreters [3] and language constructs such as Prolog "call" and "clause". However, only the Frege-Henkin semantics fully accounts for the use of dynamic predicate symbol and formula constructors such as Prolog "univ" (=..). The Russel-Henkin and Frege-Henkin semantics have one important property in common: According to both of them, the incompleteness results of higher order logic do not apply to meta-programs. In particular Gödel's incompleteness theorem for second-order logic and the undecidability of higher order unification [32] are not relevant to meta-programming - cf. [13]. This is conform to a widespread intuition.

The correctness of the Frege-Henkin semantics needs to be discussed. The reason of the introduction by Russel of his theory of types was indeed the inconsistency of Frege's *Begriffsschrift* resulting from its confusion of orders. Inconsistencies in Frege's logic are formulas affirming their own falsity, the "liar formulas". Such formulas are possible in this logic, because it assumes no stratification of the universe of discourse. A logic with liar sentences is inconsistent, in the sense that such sentences cannot be consistently interpreted: assuming the truth of a liar formula permits one to derive its falsity, and the other way around. Russel's stratification of the universe of discourse into types frees from liar formulas: such formulas are no more expressible. Liar sentences - or clauses that can be so qualified - are possible in HiLog. Due to the special acceptation of negation in logic programming, similar sentences are also possible in "unrestricted" logic programs (i.e. programs with no syntactical restrictions on the occurrences of negation in bodies of rules). The recent proposals of semantics for "unrestricted" logic programs - e.g. the Stable Model [22] and Well-Founded [23] semantics - can be seen as an alternative approach to that of Russel, for "disabling" liar formulas. Instead of discarding uninterpretable formulas, as the theory of types does, these proposals ignore them. Interestingly enough, this is possible because of the "directionality" of logic program clauses. No similar approaches seem to apply to the richer system *Begriffsschrift.* It is worth noting that primary attempts to formalize the semantics of logic programs with negation - the stratification semantics [24], further studied in [25,26] and refined in several publications - were in the spirit of Russel's theory of types: stratification semantics impose syntactical restrictions for discarding undesirable formulas.

We conclude that no type theory is, in our opinion, needed for formalizing meta-programs and term constructors.

References

[1] R. A. Kowalski. Logic for Problem Solving. North Holland, Amsterdam, The Netherland, 1979.

[2] D. H. D. Warren. Higher-Order Extensions to Prolog: Are They Needed? Machine Intelligence, 10:441-454, 1982.

[3] L. Sterling and E. Shapiro. The Art of Prolog. The MIT Press, Cambridge, Massachussetts, 1986.

[4] L. Sterling and R. Beer. Incremental Flavour-Mixing of Meta-Interpreters for Expert System Construction. In Proc. 3rd Logic Programming Symp. Salt Lake City, Utah, 1986.

[5] L. Sterling and R. Beer. Meta Interpreters for Expert System Construction. Journal of Logic Programming. 1988.

[6] T. Miyachi, S. Kunifuji, H. Kitakami, K. Furukawa, A. Takeuchi, A. and H. Yokota. A Knowledge Assimilation Method for Logic Databases. New Generation Computing, 2:385-404, 1984.

[7] K. A. Bowen and R. A. Kowalski. Amalgamating Language and Metalanguage in Logic Programming. In Logic Programming, K. L. Clark and S.-A. Taernlund eds.. Academic Press, New York, 153-172, 1983.

[8] M. van Emden and R. A. Kowalski. The Semantics of Predicate Logic as Programming Language. Journal of the ACM, 23, 4:733-742, 1976.

[9] J. W. Lloyd. Foundations of Logic Programming. Springer-Verlag, Berlin, Germany, 1984, 2nd ed. 1987.

[10] V. S. Subrahmanian. A Simple Formulation of the Theory of Metalogic Programming. In Proc. Workshop on Meta-Programming in Logic Programming, Technical Report, Department of Computer Science, The University of Bristol, England, 1988.

[11] B. Russel. Mathematical Logic as Based on the Theory of Types. American Journal of Mathematics, 30:222-262, 1908. Reprinted in [12].

[12] J. van Heijenoort, ed. From Frege to Gödel - A Source Book in Mathematical Logic, 1879-1931. Harvard University Press, Cambridge, Massachusetts, 1967.

[13] L. Henkin. Completeness in the Theory of Types. Journal of Symbolic Logic, 15:81-91, 1950.

[14] J. C. Mitchell. Type Systems for Programming Languages. In [15]: 364-458, 1990.

[15] J. van Leeuwen. Handbook of Theoretical Computer Science, Volume B. Formal Models and Semantics. Elsevier Science Publishers B.V., Amsterdam, 1990.

[16] C. Chen, M. Kifer, and D. S. Warren. HiLog: A First-Order Semantics for Higher-Order Logic Programming. In Proc. of the North American Conf. on Logic Programming, Cleveland, Ohio, 1989.

[17] T. Frühwirth. A Polymorphic Type Checking System for Prolog in HiLog. In Proc. 6th Israel Conference on Artificial Intelligence and Computer Vision, 1989.

[18] E. Yardeni, T. Frühwirth, and E. Shapiro. Polymorphically Typed Logic Programs. In [19], 1992.

[19] F. Pfenning, ed. Types in Logic Programming. The MIT Press, Cambridge, Massachussetts, 1992.

[20] G. Frege. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, Germany, 1879. Reprinted in [21]. English translation in [12].

[21] F. von Kutschera. Gottlob Frege - Eine Einführung in sein Werk. Walter de Gruyter, Berlin, Germany, 1989.

[22] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In Proc. 5th Int. Conference on Logic Programming, 1070-1080, Seattle, Washington, 1988.

[23] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. Journal of the ACM, 38(3):620-650, 1991.

[24] A. K. Chandra and D. Harel. Horn Clause Queries and Generalizations. Journal of Logic Programming, 1(1):1-15, 1985.

[25] K. R. Apt, H. A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In [26]:89-148. 1988.

[26] A. Van Gelder. Negation as Failure Using Tight Derivations for General Logic Programs. In [26]:149-176. 1988.

[27] J. Minker, ed. Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, Los Altos, California, 1988.

[28] J. W. Lloyd, ed. Proceedings of the Workshop on Meta-Programming in Logic Programming. June 1988, Bristol, England. Technical Report, Department of Computer Science, University of Bristol, England, 1988.

[29] M. Bruynooghe, ed. Proceedings of the 2nd Workshop on Meta-Programming in Logic. April 1990, Leuven, Belgium. Technical Report, Dept. of Computer Science, K.U. Leuven, Belgium, 1990.

[30] A. Pettorossi, ed. Proceedings of the 3rd International Workshop on Meta-Programming in Logic. June 1992, Uppsala, Sweden. Technical Report, Computing Science Department, Uppsala University, Sweden, 1992.

[31] JP. Hill and J. W. Lloyd. The Gödel Report. Technical Report, Department of Computer Science, Universtiy of Bristol, England, 1991.

[32] W. D. Goldfarb. The Undecidability of the Second-Order Unification Problem. Theoretical Computer Science, 13:225-230, 1981.

[33] G. Frege. Begriffsschrift und andere Aufsätze. Olms-Verlag, Hildesheim, Germany, 1964. Reprints of the Begriffsschrift and of other publications of G. Frege, edited by I. Angelilli.