# Theoretical Issues in Type Inference — A Final Report on the Status of Strong Typing for SETL

Fritz Henglein
Courant Institute of Mathematical Sciences
New York University
715 Broadway, 7th floor
New York, N.Y. 10003
Internet: henglein@nyu.edu

March 31st, 1988

**Abstract**

We briefly outline the status of theoretical results on type inference in language settings that reflect relevant language features and their typical usage as found in the programming language SETL. Most of these results, a few by the author, most by a very active research community, are negative in nature since they indicate the — theoretical — infeasibility of certain types of type inference. This and some other considerations from a specifically programming language oriented point of view suggest a departure from a completely "orthogonal", general logical formalization of type inference in favor of more realistic – yet completely formal – and (hopefully) feasible problem formulation.

This report should be considered a follow-up to an earlier overview article [Hen88c].

## 1 Introduction

In [Hen88c] we had identified the following language features as "characteristic" of programming in SETL:

- uniform (implicit) parametric polymorphism with automatic type inference;
- operator overloading;
- implicit (undeclared) recursive types and union types.

We will briefly touch upon each of these features as they concern the problem of type inference. Some recent papers by the author of this report, exclusively on the problem of parametric polymorphic type inference, are [Hen88d], [Hen88b], and [Hen88a]

## 2 Parametric Polymorphism

There have been several "negative" results on the infeasibility of polymorphic type inference, in part shattering beliefs held for some ten years. In [Hen88c] we had incorrectly announced a polynomial-time algorithm for type inference in the Milner-Mycroft Calculus, a "uniform" extension of the typing discipline found in ML [Har86], based on a fast algorithm for uniform semi-unification [Hen88d]. This algorithm cannot easily be extended to nonuniform semi-unification, which is polynomial-time equivalent to Milner-Mycroft style type inference. Even worse, the proof of decidability of the Milner-Mycroft Calculus, as reported in [KTU88], has been retracted, and recently Kanellakis and Mitchell [KM89] proved that ML typing is in fact PSPACE-hard, thus dashing all reasonable hope of a polynomial-time type inference algorithm for ML and contradicting claims of polynomial-time computability that have pervaded the literature for many years.

These results contradict past and current experience with ML and its type checking system that have performed quite efficiently in practice. Kanellakis, Mitchell, and others have raised the question of why ML typing appears practically feasible in the face of negative theoretical evidence. A critical component in showing the PSPACE-hardness of ML typing in [KM89] is the construction of a class of "unrealistic" programs whose (inferred) typing information is exponential-sized (even in a "compact" graph representation) with respect to the underlying program. It can be argued that type information for a program fragment that is much bigger than the fragment itself fails to address an important intent of typing in programming, namely providing an *abstraction* — in the sense of conceptual simplification — of the semantics of the program. If we impose polynomial bounds on the sizes of acceptable type expressions relative to their underlying programs then it can be shown that ML typing and its extension to the Milner-Mycroft Calculus is — theoretically — feasible [Hen89] (although a careful definition of "size" is necessary). It appears that realistic programs satisfy even the strictest of these size restrictions on programs. A possible – technically very strict, but probably reasonable – restriction could be that the type expression of any program expression can be at most one type-written page long.

We believe that pursuing the full type inference problem — without size restrictions — is worthwhile for two reasons: it is a fundamental problem that sheds light on the basic combinatorial and computational problems of type inference and, more practically, type inference algorithms developed for the full problem can be easily adapted to restricted versions (which, as we believe, explains at least in part why type inference algorithms for hard problems perform satisfactorily in practice). In this vein we have been able to make the computational connections between simple type inference and ordinary unification [Hen88b] as well as between Milner-Mycroft style type inference and semi-unification [HP88] precise, and we have provided partially correct algorithms for semi-unification [Hen88d] whose practicality we tested with a prototype implementation in SETL. We have also investigated the algebraic structure of solutions of semi-unification problems [Hen88a], which can be interpreted (via the above-mentioned connections) as an investigation of the structure of typings in the Milner-Mycroft Calculus. However, the ultimate goal of our studies, namely proving decidability of semi-unification and type inference in the Milner-Mycroft Calculus has been elusive so far even though we have developed a candidate algorithm that we believe to be uniformly terminating [Hen88d].

## 3 Dynamic Overloading

In [Hen88c] we outlined a type discipline revolving around socalled *oligotypes* meant to capture in a strongly typed setting the sort of operator overloading found in SETL. We have not pursued oligotypes any further since [Hen88c], but a similar discipline [Kae88] has been found to be theoretically robust; in particular, it admits a principal typing property that guarantees uniqueness of inferred type information.

General overloading in type inference systems has been shunned largely because many of its variants are known to be NP-hard (e.g., [ASU86, ex. 6.23]). In view of the recent hardness results — yet practicality — of polymorphic type inference it may be worthwhile reevaluating this rejection of overloading on performance grounds.

## 4 Recursive Types and Union Types

A "clean" formulation of polymorphic type inference in the presence of implicit recursive types and free union types (in the spirit of Algol-68) seems very difficult and computationally prohibitive. Mishra and Reddy's approach [MR85] is rather *ad hoc*. The typing rules are not spelled out explicitly, their type inference algorithm is complicated and cannot be considered "complete" with respect to their (intended) type inference system since it relies on *ad hoc* restrictions — motivated by the development of the algorithm itself. Since the union type constructor for Algol-68 style free unions satisfies the algebraic properties of associativity, commutativity, and idempotence, the ACI-unification problem can be feasibly encoded as a type inference problem with free union types. Since ACI-unification is known to be NP-hard [KN86], this implies that type inference with free union types is also NP-hard even in a language without polymorphism. For this reason we have only included *tagged* union types along with recursive types into the type model of SETL (see, e.g, [Hen87]), although, once again, this decision may warrant reconsideration in view of the apparent contradiction of theoretical and practical complexity of other type inference problems.

## 5 Conclusion and Outlook

We have presented an outline of our recent work and insights on the problem of type inference for a strongly type variant of SETL. We have eliminated the programming language specifics of parametric (polymorphic) type inference by distilling their combinatorial essence in the form of unification-like problems, and we've developed algorithms for solving these problems. More generally, we consider our theoretical investigations to be a counterpart to the practical work on type checking and type finding for SETL done within the SED project [Kel87]. Furthermore, we believe that these studies provide valuable guidance in the design of a full-blown, practical type model and inference system for a wide array of languages, not only SETL. However, recent negative results indicate that, so far, type theory has failed to explain why type inference "works" in languages such as ML and Miranda or LEAP [PL88]. For this reason we have outlined some initial considerations that may aid in formulating such an explanation.

## References

[ASU86] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1986. Addison-Wesley, 1986, Reprinted with corrections, March 1988.

[Har86] R. Harper. Introduction to Standard ML (preliminary draft). Technical report, University of Edinburgh, February 1986.

[Hen87] Fritz Henglein. A polymorphic type model for SETL. Technical Report (SETL Newsletter) 221, New York University, July 1987.

[Hen88a] Fritz Henglein. Algebraic properties of semi-unification. Technical Report (SETL Newsletter) 233, Courant Institute of Mathematical Sciences, New York University, November 1988.

[Hen88b] Fritz Henglein. Simple type inference and unification. Technical Report (SETL Newsletter) 232, Courant Institute of Mathematical Sciences, New York University, October 1988.

[Hen88c] Fritz Henglein. Strong typing in SETL — an overview of work on strong typing in the NYU/SETL project. Manuscript, April 1988.

[Hen88d] Fritz Henglein. Type inference and semi-unification. In *LFP '88: Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 184–197, New York, NY, USA, July 1988. ACM.

[Hen89] Fritz Henglein. *Polymorphic Type Inference and Semi-Unification.* PhD thesis, Rutgers University, April 1989. Available as NYU Technical Report 443, May 1989, from New York University, Courant Institute of Mathematical Sciences, Department of Computer Science, 251 Mercer St., New York, N.Y. 10012, USA.

[HP88] Fritz Henglein and Ken Perry. On the complexity of ML$^+$ type inference. Research note. Submitted to Conf. on Logic in Computer Science '89, 1988.

[Kae88] S. Kaes. Parametric overloading in polymorphic programming languages. In H. Ganzinger, editor, *Proc. 2nd European Symosium on Programming, Lecture Notes in Computer Science, Vol. 300*, pages 131–144, Nancy, France, March 1988. Springer-Verlag.

[Kel87] J. Keller, editor. *Athens Review Meeting.* ESPRIT Project 1227: SETL Expermination and Demonstrator, July 1986 - July 1987.

[KM89] P. Kanellakis and J. Mitchell. Polymorphic unification and ML typing (extended abstract). In *Proc. 16th Annual ACM Symp. on Principles of Programming Languages.* ACM, January 1989.

[KN86] D. Kapur and P. Narendran. NP-completeness of the set unification and matching problems. In *Proc. 8th Int'l Conf. on Automated Deduction*, pages 489–495. Springer-Verlag, 1986. Lecture Notes in Computer Science, Vol. 230.

[KTU88] A. Kfoury, J. Tiuryn, and P. Urzyczyn. A proper extension of ML with an effective type-assignment. In *Proc. 15th Annual ACM Symp. on Principles of Programming Languages*, pages 58–69. ACM, ACM Press, January 1988.

[MR85] P. Mishra and U. Reddy. Declaration-free type checking. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 7–21. ACM, January 1985.

[PL88] F. Pfenning and P. Lee. LEAP: A language with eval and polymorphism. Technical Report ERGO-88-065, Carnegie-Mellon University, September 1988.