Sorted HiLog: Sorts in Higher-Order Logic Data Languages

Weidong Chen*
Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275-0122, U.S.A.
wchen@seas.smu.edu

Michael Kifer†
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, U.S.A.
kifer@cs.sunysb.edu

September 30, 1994

Abstract

HiLog enhances the modeling capabilities of deductive databases and logic programming with higher-order and meta-data constructs, complex objects, and schema browsing. Its distinctive feature, a higher-order syntax with a first-order semantics, allows for efficient implementation with speeds comparable to Prolog. In fact, HiLog implementation in XSB [30, 26] together with tabulated query evaluation offers impressive performance with negligible penalty for higher-order syntax, thereby bringing the modeling capabilities of HiLog to practical realization.

The lack of sorts in HiLog, however, is somewhat of a problem in database applications, which led to a number of HiLog dialects such as DataHiLog [25]. This paper develops a comprehensive theory of sorts for HiLog. It supports HiLog's flexible higher-order syntax via a *polymorphic* and *recursive* sort structure, and it offers an easy and convenient mechanism to control the rules of well-formedness. By varying the sort structure we obtain a full spectrum of languages, ranging from classical predicate logic to the original (non-sorted) HiLog. In between, there is a number of interesting higher-order extensions of Datalog with various degrees of control over the syntax, including second-order predicate calculus with Henkin-style semantics, as described in [10]. We also discuss the benefits of using Sorted HiLog for modeling complex objects and for meta programming. Finally, Sorted HiLog can be easily incorporated into XSB, which makes its practical realization feasible.

Technical Report 94/8
June 1994

Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794

Short version published as [?].

*Work supported in part by the NSF grant IRI-9212074.

†Work supported in part by the NSF grant CCR-9102159 and a grant from New York Science and Technology Foundation RDG90173.

1 Introduction

HiLog [6] is a higher-order language for deductive databases and logic programming. It not only expands the limits of first-order logic programming and obviates the need for several non-logical features of Prolog, but also provides important features for databases, including schema browsing and nested and higher-order relations similar to those in COL [1] and LDL [2]. We refer the reader to [6] for the details of these applications. HiLog has been used by many researchers for various ends, such as for specifying types in logic programming [11, 34], for database query languages (*e.g.*, in the Glue-Nail! project [23]), and for object-oriented databases [19]. HiLog has been implemented as part of the XSB system with tabulated query evaluation [30],[1] and it runs at a very impressive speed compared to other deductive databases, such as LDL or Coral [26]. The on-going implementation [27] of C-logic and F-logic [16, 7] in XSB and its integration with HiLog will offer the ability to reason with objects and schema.

The main reason for the popularity of HiLog is its flexible syntax, the simplicity of its semantics, and the fact that its logical entailment is upward-compatible with classical logic. However, at the same time, it was felt that the syntax of HiLog is much too flexible, sometimes making it necessary to impose unwelcome restrictions on the range of logical variables in the program clauses.

Another problem is that HiLog has no higher-order counterparts for various tractable sub-logics of classical logic, such as Datalog, that are all-important in deductive databases. The Herbrand universe (which is the same as the Herbrand base) in HiLog is always infinite due to term application. One unpleasant off-shot of this is that the usual semi-naive bottom-up computation may not terminate, and even proper formulation of complexity results (analogous to those for Datalog) becomes an issue. As a result every query has to be analyzed for "finiteness" before it can be evaluated, even for programs with no applications of function symbols. To overcome this drawback, some researchers attempted to extract useful specialized sub-logics out of HiLog. Examples of this are Relational HiLog [24] and DataHiLog proposed [25]. However, strictly speaking, DataHiLog is not a sublanguage of HiLog in the sense in which Datalog is a sublanguage of classical Horn logic.

The third problem concerns the proof theory. Although HiLog has a sound and complete proof theory, the *direct* resolution-based proof theory of [6] has limitations, which are caused by the fact that Skolemization is not possible in some cases (see [6] for details).

In this paper, we show that all these problems can be rectified with a single mechanism, a *sorted* logic. A superposition of the idea of sorts and HiLog results in what we call *Sorted HiLog*. The idea of using sorts to control syntax is, of course, not new and one may even feel skeptical that applying this idea to HiLog may yield something original. However, as it turns out, a number of problems need to be solved. As they are known in classical logic, sorts are too limited when it comes to supporting the syntax of HiLog. Even the more elaborate theories [9, 14, 29] do not meet the requirements, as they were designed to address different problems. The requisite theory of sorts for HiLog should provide for more control over the syntax and, at the same time, be able to support those features of the syntax that make HiLog an attractive language.

The sort structure proposed in this paper is designed to accommodate both of these (seemingly conflicting) goals. The proposed sort structure is *polymorphic* and *recursive*. The logic itself is independent of the particular choice of a sort structure, and sorts can be viewed as a parameter to the logic. By varying the sort structure, we obtain a "continuum" of logic languages, ranging from ordinary HiLog to classical predicate calculus, with various decidable and higher-order extensions of Datalog in between. DataHiLog [25], mentioned earlier, is one of the special cases of Sorted HiLog and so is the second-order predicate calculus with Henkin-style semantics, described in [10, Section 4.4].

Before going into technical details of this paper, it may be useful to give a brief overview of the notions

[1] XSB and HiLog can be obtained via the anonymous FTP to *cs.sunysb.edu* in *pub/XSB/*.