Logical Foundations of Object-Oriented and Frame-Based Languages

Michael Kifer * Georg Lausen † James Wu ‡

Abstract

We propose a novel formalism, called *Frame Logic* (abbr., F-logic), that accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages. These features include object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation, and others. In a sense, F-logic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming. F-logic has a model-theoretic semantics and a sound and complete resolution-based proof theory. A small number of fundamental concepts that come from object-oriented programming have direct representation in F-logic; other, secondary aspects of this paradigm are easily modeled as well. The paper also discusses semantic issues pertaining to programming with a deductive object-oriented language based on a subset of F-logic.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Languages—*query languages*; I.2.3 [**Artificial Intelligence**]: Deduction and theorem proving—*deduction, logic programming, nonmonotonic reasoning*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical logic—*logic programming, mechanical theorem proving*

General Terms: Languages, Theory

Additional Key Words and Phrases: Object-oriented programming, frame-based languages, deductive databases, logic programming, semantics, proof theory, typing, nonmonotonic inheritance

Journal of the Association for Computing Machinery, May 1995

*Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, U.S.A.
Email: kifer@cs.sunysb.edu. Work supported in part by NSF grants DCR-8603676, IRI-8903507, and CCR-9102159.
†Institut für Informatik, Universität Freiburg, Rheinstrasse 10-12, 79104 Freiburg, Germany.
Email: lausen@informatik.uni-freiburg.de
‡Renaissance Software, 175 S. San Antonio Rd., Los Altos, CA 94022, U.S.A. Email: wu@rs.com.
Work supported in part by NSF grant IRI-8903507.

Contents

1 Introduction

In the past decade, considerable interest arose in the so-called *object-oriented* approach, both within the database community and among researchers in programming languages. Although "the object-oriented approach" is only a loosely defined term, a number of concepts, such as complex objects, object identity, methods, encapsulation, typing, and inheritance, have been identified as the most salient features of that approach [13, 96, 113, 107, 10].

One of the important driving forces behind the interest in object-oriented languages in databases is the promise they show in overcoming the, so called, *impedance mismatch* [74, 113] between programming languages for writing applications and languages for data retrieval. Concurrently, a different, *deductive* approach gained enormous popularity. Since logic can be used as a computational formalism *and* as a data specification language, proponents of the deductive programming paradigm have been arguing that this approach overcomes the aforesaid mismatch problem just as well. However, in their present form, both approaches have shortcomings. One of the main problems with the object-oriented approach is the lack of logical semantics that, traditionally, has been playing an important role in database programming languages. On the other hand, deductive databases rely on a flat data model and do not support data abstraction. It therefore can be expected that combining the two paradigms will pay off in a big way.

A great number of attempts to combine the two approaches has been reported in the literature (*e.g.*, [1, 2, 3, 14, 17, 18, 35, 58, 60, 68, 66, 73, 93, 11]) but, in our opinion, none was entirely successful. These approaches would either seriously restrict object structure and queries; or they would sacrifice declarativity by adding extra-logical features; or they would omit important aspects of object-oriented systems, such as typing and inheritance.

In this paper we propose a formalism, called *Frame Logic* (abbr., F-logic), that achieves *all* of the goals listed above and, in addition, it is suitable for defining, querying, and manipulating database schema. F-logic is a *full-fledged* logic; it has a model-theoretic semantics and a sound and complete proof theory. In a sense, F-logic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming.

Apart from object-oriented databases, another important application of F-logic is in the area of frame-based languages in AI [42, 80], since these languages are also built around the concepts of complex objects, inheritance, and deduction. It is from this connection that the name "Frame Logic" was derived. However, most of our terminology comes from the object-oriented parlance, not from AI. Thus, we will be talking about objects and attributes instead of frames, slots, and the like.

For reasoning about inheritance and for knowledge base exploration, a logic-based language would be greatly aided by higher-order capabilities. However, higher-order logics must be approached with caution in order to preserve desired computational properties. In the past, a number of researchers suggested that many useful higher-order concepts of knowledge representation languages can be encoded in predicate calculus [48, 77]. From the programmer's point of view, however, encoding is not satisfactory, as it gives no *direct* semantics to the important higher-order constructs and it does not retain the spirit of object-oriented programming. In contrast, F-logic represents higher-order and object-oriented concepts directly, both syntactically and semantically.

This work builds upon our previous papers, [58, 55, 60], which in turn borrowed several important ideas from Maier's O-logic [73] (that, in its turn, was inspired by Aït-Kaci's work on ψ-terms [7, 6]).

In [58, 60], we described a logic that adequately covered the structural aspect of complex objects but was short of capturing methods, types, and inheritance. The earlier version of F-logic reported in [55]