OLOG: A Deductive Object Database Language  
(Extended Abstract)

Mengchi Liu  
Department of Computer Science, University of Regina  
Regina, Saskatchewan, Canada S4S 0A2  
mliu@cs.uregina.ca  
http://www.cs.uregina.ca/~mliu

Abstract. Deductive object-oriented databases are intended to combine the best of the deductive and object-oriented approaches. However, some important object-oriented features are not properly supported in the existing proposals. This paper proposes a novel deductive language that supports important structurally object-oriented features such as object identity, complex objects, typing, classes, class hierarchies, multiple property inheritance with overriding, conflict-handling, and blocking, and schema definitions in a uniform framework. The language effectively integrates useful features in deductive and object-oriented database languages. The main novel feature is the logical semantics that cleanly accounts for those structurally object-oriented features that are missing in object-oriented database languages. Therefore it establishes a theoretical foundation for a practical deductive object-oriented database system for advanced database applications.

1 Introduction

In the past few years, a number of deductive object-oriented database languages have been proposed, such as O-logic [28], revised O-logic [17], F-logic [16], IQL [3], LOGRES [9], LLO [27], Noodle [30], Complex [14], CORAL++ [32], DLT [3], Golog [12], Rock & Roll [6], ROL [22], DO2 [19], and ROL2 [20, 25].

The objective of deductive object-oriented databases is to combine the best of the deductive and object-oriented approaches, such as recursion, declarative querying, and a firm logical foundation from the deductive approach, and object identity, complex objects, typing, classes, class hierarchy, multiple property inheritance with overriding, conflict-handling and blocking, and schema definition from the object-oriented approach. However, the existing proposals fall into two kinds: languages with a logical semantics but only with limited object-oriented features such as revised O-logic, F-logic, and ROL, and languages with more object-oriented features but without a logical semantics such as IQL, Rock & Roll, CORAL++.

Non-monotonic multiple property inheritance is a fundamental feature of object-oriented database languages, such as O₂ [11] and Orion [18]. The user can explicitly redefine (override) the inherited attribute. For some applications,

© Springer-Verlag Berlin Heidelberg 1999
it is also important to be able to block the inheritance of attributes from superclasses (i.e., selective inheritance \cite{18}). Besides, possible ambiguities may arise when an attribute name is defined in two or more superclasses, and the conflict should be handled properly. Unfortunately, multiple property inheritance with such overriding, conflict-handling and blocking capacity does not have a well-defined semantics even though it has been used in some object-oriented database languages. The main difficulty is that the inherited instances of a superclass may not be well-typed with respect to its type definition because of overriding, conflict and blocking. Most deductive object-oriented database languages, including revised O-logic, F-logic, only supports monotonic multiple property inheritance, which is not powerful enough. ROL has a semantics that accounts for multiple property inheritance with overriding but not conflict-handling and blocking in a limited context. Until now, a well-defined semantics for multiple property inheritance with overriding, conflict handling, and blocking in object-oriented databases is still missing from the literature.

Object identity is another fundamental feature of object-oriented databases. It is useful for supporting sharing and cyclicity \cite{2,3,11}. In the object-oriented data model $O_2$, an object is a pair of object identifier (oid) and value. The value can be not only a tuple, but also a set, an atomic value, and even an image. In other words, an object identifier in $O_2$ can identify any kind of value. However, this important use of object identity is not directly supported in most deductive object-oriented database languages, including O-logic, revised O-logic, F-logic, Golog, ROL, and ROL2, in which an object identifier can only directly identify a tuple. IQL is based on $O_2$ and is so far the only deductive object-oriented language that supports this use of object identity.

Another problem with existing languages is how to deal with object generation. In Datalog, we only have constants and facts that are used to represent real objects and their relationships. In O-logic, F-logic, and ILOG \cite{15}, skolem functions are used for generated object identifiers. Consider the following typical interesting pair \textit{ip} rule in revised O-logic (also in F-logic):

\[
\begin{align*}
\text{f}(E, M) : \text{ip}[\text{emp} \rightarrow E, \text{mgr} \rightarrow M] : & :- \\
E : \text{empl}[\text{name} \rightarrow N, \text{works} \rightarrow D] \\
D : \text{dept}[\text{mgr} \rightarrow M : \text{empl}[\text{name} \rightarrow N]]
\end{align*}
\]

This rule generates object identifiers of the class \textit{ip} of the form \textit{f}(E, M) with values $E$ and $M$ for corresponding attributes \textit{emp} and \textit{mgr}. Clearly, information about $E$ and $M$ is duplicated in the object identifier and the attribute values. As discussed in \cite{17,34}, termination is still not guaranteed. The user has to properly define rules involving skolem functions in order to guarantee the termination of execution. For this reason, pure relation-based deductive approach is considered better in this regard \cite{34}. Indeed, skolem functions are nothing more than relationships. Treating them as object identifiers is unnecessary and introduces redundancy. If relations are directly supported, we can avoid redundancy and guarantee termination. For example, we can represent the above rule equivalently as follows: