COL: A LOGIC-BASED LANGUAGE FOR COMPLEX OBJECTS

Serge Abiteboul, Stephane Grumbach
I.N.R.I.A.
78153 Le Chesnay, FRANCE
abitebou@inria.inria.fr
grumbach@bdblues.inria

Abstract: A logic-based language for manipulating complex objects constructed using set and tuple constructors is introduced. A key feature of the language is the use of base and derived data functions. Under some stratification restrictions, the semantics of programs is given by a canonical minimal and causal model that can be computed using a finite sequence of fixpoints. Applications of the language to procedural data, semantic database models, heterogeneous databases integration, and datalog query evaluation are presented.

INTRODUCTION

Two approaches have been followed for defining manipulation languages for complex objects: (1) an algebraic approach [AB, ABI, FT, SS and many others], and (2) a calculus approach [J, AB, RKS]. Recently, there has been some interest in pursuing a so-called logic programming approach [BK, AG, Be+, K]. This is the approach followed here.

The Complex Object Language (COL) based on recursive rules is presented. This language is an extension of datalog (Horn clauses without function symbols) which permits the manipulation of complex objects obtained using tuple and (heterogeneous) set constructors. The originality of the approach is that besides the base and derived relations, base and derived "data functions" are considered. As we shall see, data functions are either defined extensionally (base data functions) or intentionally (derived data functions).

Data functions are natural tools to manipulate complex objects. Data functions present other advantages as well:

(i) Since queries can be viewed as data functions, the inelegant dichotomy between data and queries of the relational model disappear. In particular, queries can be stored in the database as data functions. The model therefore permits the manipulation of procedural data [S].

1 This research was supported in part by the Projet de Recherches Coordonnees BD3.
(ii) In COL, data can be viewed both in a functional and in a relational manner. As a consequence, the language can be used in a heterogeneous database context (e.g., relational view on a functional database; integration of a relational database with a functional one).

(iii) COL can also be used as a kernel language for semantic database models like SDM [HM], IFO [AH1] or Daplex [Sh].

(iv) Some evaluation techniques for datalog queries like Magic Sets or others [B+,GM] make extensive use of particular functions. These functions can be formalized using our model.

The language COL is based on a clausal logic. The database consists of facts concerning both the predicates, and the data functions. New facts can be deduced using rules. The use of monovalued data functions in COL yields consistency problems which are studied in [AH3]. To simplify the presentation, we restrict here our formal presentation of the language to multivalued data functions. (However, some of the examples that are presented also use monovalued data functions.)

The present paper is devoted essentially to an informal presentation of the language. A detailed description of the theoretical foundations can be found in [AG]. The treatment of monovalued data functions is the object of a separate paper [AH3].

A COL program consists of facts and rules. Its semantics is given in terms of minimal models as it is standard for instance in datalog, or in datalog with negation. Unfortunately, because of sets and data functions, some programs may have more than one minimal models. A stratification in the spirit of the stratification introduced by [ABW,GN and others] is used. It is shown in [AG] that for stratified programs, a canonical causal [BH] and minimal model of a given program can be computed using a sequence of fixpoints of operators. We briefly sketch here the stratification, and the fixpoint semantics.

Like in [ABW,GN...], we allow negations in the body of rules. The stratification is necessary to handle such negations. We shall see that negation can be simulated using data functions.

As mentioned above, two other approaches have been independently followed to obtain a rule-based language for complex objects [Be+,K]. In [Be+], they do not insist on a strict typing of objects. In [K], only one level of nesting is tolerated. However, both approaches could easily be adapted to the data structures considered in this paper. Furthermore, in [AB], it is argued that all these approaches yield essentially the same power (i.e., the power of the safe calculus of [AB]). The points (i-iv) above clearly indicate advantages of our approach.