Deductive[1] and Object-Oriented Databases

Serge Abiteboul
I.N.R.I.A.,
78153 Le Chesnay, France.
Serge.Abiteboul@inria.fr

Abstract

These notes were written for a tutorial in JICSLP (Washington, 1992) and are not intended to be a comprehensive survey of the field.

We are concerned with *dooD*, i.e. deductive object-oriented databases (systems). We briefly review *ooD systems* (object-oriented database management systems) and *deductive database systems* (the deductive ones). A main purpose is to try to clarify these concepts. As we shall see, although there is already a conference entitled dooD and a large body of literature on the topic, the meaning of dooD is not yet clear. The issues are not precise and we are still looking forward to the first real implementation of a dooD system. We will try to present the motivations, the main achievements so far, and the open questions.

1 Introduction

Data intensive applications are nowadays currently written using C and SQL. This is a major source of difficulties since these two languages are based on different paradigms. Furthermore, the user interface is often implemented using specific tools and in particular 4th generation languages and yet different environments. Therefore, application programmers have to deal with tables and tuples at the SQL level, pointers and records at the C level, objects, windows or menus at the interface level. The prime goal of *deductive object-oriented database systems* is to provide a uniform rule-based environment to facilitate the design and maintenance of data intensive applications. The challenge is to improve application programmer's productivity while still guaranteeing reasonable performances. It is not clear that such a goal can be achieved. Also, as we shall see, from a scientific viewpoint, the topic is yet rather ill defined. However, we believe that the research community should participate in the study of these systems: (i) there are indications that such systems will become available and (ii) they raise a number of interesting technical issues.

[1] A subtlety in the term dooD is that the first "D" can be understood as *deductive* or as *declarative*. In this note, we often think of it as declarative, or in other words, we use "deductive" in a very liberal sense.

An important question is "a dooD system for which database applications". One often, for instance, cites engineering databases (notably software engineering), CAD databases or information bases (integrating text, image, graphic, etc.). This important issue seems unclear to the author of the present note and will be ignored here.

2 Databases, Deduction, Object-Orientation

2.1 Databases

A DBMS provides easy access to a *persistent, shared, large volume of data*. Each of the terms is important to understand those systems. For instance, because of the volume of data, secondary storage management including index management, data clustering, query optimization, is critical in terms of performance although it is not directly visible to the user. There have already been a number of DBMS generations, in particular, the "relational revolution" [44, 32] which stressed *data independence* by providing (i) a cleaner separation between logical and physical data structures and (ii) higher level query languages (logic-based and set-oriented).

We are concerned here with future generations. In this respect, it is important to see that the switch to a new generation is only feasible when the modelling benefits are crucial and when the technical difficulties are decently taken care of. For instance, although the success of relational systems is certainly due to the simplicity and elegance of the relational model, it was permitted by technological breakthroughs in access structures and optimization techniques. To consider the next generations, it is important to understand the major limitations of relational systems. The first one stems from the nature of the data structure, the relation, which is too simplistic for many database applications. The second one is that the core of the relational model is essentially a query language, i.e. a language for accessing data. To write real database applications, one also needs the full power of a programming language. Thus one has to use an embedding of the query language in a programming language (e.g., C+SQL) which leads to the so-called *impedance mismatch*.

To overcome these problems, two main directions have been followed. One is based on the integration of logic-programming features and the second on the object-oriented paradigm.

2.2 Deductive databases

Deductive databases were first studied in the 70's and were promoted by researchers such as Jack Minker, Hervé Gallaire or Jean-Marie Nicolas, notably through a series of workshops in Toulouse. It gained a new life in the 80's in the research community in particular by the development of optimization heuristics (e.g., Magic [12] or QSQ [45]). At about the same time, several important projects were launched in industry in Europe (ECRC), the States (MCC) or in Japan (ICOT), or in academic environments (Stanford, Inria, Milano, etc.). These resulted in important technological advances, but less obvious industrial impact and no serious industrial product.

A deductive system incorporates aspects of logic programming and thereby provides advantages over pure relational systems by bridging the gap towards knowledge bases. The basic data structure in deductive databases is the relation like in the relational model. The main difference lies in the language for manipulating relational facts. Instead of using first-order calculus as in the relational model, programs allow to specify the deduction of new facts from stored data and rules. One therefore distinguishes between extensional predicates (which are stored) and intentional predicates (which are specified using rules). One usually insists on the increase of expressive power by mentioning that transitive closure *not* expressible in relational calculus can be expressed in deductive languages. Although this is an important observation (transitive-closure-like operations are essential for graph manipulation), we view here logical rules in a more general context. Rules provide the means to specify knowledge about the application under various forms. These of course includes implicit knowledge (rules specifying intensional predicates) but other aspects as well:

1. integrity constraint maintenance. Most recent activity on integrity constraints has centered around the deductive paradigm. This is clearly a benefit from the sound foundations of this paradigm. But it is also due to a convergence of techniques between deduction handling and integrity verification. This is an important aspect since the lack of tools for maintaining integrity constraint is viewed by many developers as one of the main limitations of available database systems.

2. Explanations. Providing explanations to users is considered important in expert systems. This aspect has not been properly addressed in database systems and with applications becoming more and more complex, this is becoming an issue that only the deductive paradigm seems capable of addressing.

3. Active rules. The capability to properly model a rapidly changing reality is (we believe) *the* major challenge facing databases. Unfortunately, there is little to build on. Object-oriented databases provide a solution based on methods and (unfortunately on) a

procedural approach. A major issue for deductive databases is to come up with a declarative answer to this problem of modelling database evolution.

The deductive paradigm is clearly strongly influenced by the logic programming field. However, from a database point of view, standard logic-programming languages such as Prolog are viewed as too procedural, too complicated for database users and too rich to be efficiently optimized. The deductive approach has centered around the study of *datalog*, i.e., Horn clauses without function symbols [27, 13]. Indeed, most of the efforts in deductive databases have been devoted to optimizing datalog. Some of the optimization techniques are based on top-down semantics as standard in logic programming (e.g., QSQ). Some are bottom-up which is more common in databases (e.g., Magic). In most cases, they favor computing with sets of tuples (vs. single tuples in Prolog) to take advantage of a relational algebraic engine and relational algebra optimization.

Although interesting and crucial for the implementation of deductive systems, these optimization techniques are not within the scope of the present tutorial. Descriptions of such optimization techniques and appropriate references can be found in [44, 23].

In deductive databases, other important research directions investigated recently are:

1. providing declarative semantics to negation. After a number of proposals and heated debate, two semantics seem to emerge: stratification (see [44]) and well-founded models [28]. We ignore this issue here and refer to, for instance, [44, 18];

2. considering deductive languages with more complex data structures and in particular set-based data structures;

3. introducing updating capabilities into rules.

These two last items are at the core of the dooD proposal and are considered below.

2.3 Object-orientation

The integration of object-orientation in database systems is now mastered from an implementation point of view and several ooD systems are already commercially exploited (e.g., O_2-Technology or Versant). The main purpose of this new technology was to enlarge the range of applications that could benefit from database technology (e.g., engineering or CAD databases) and at the same time improve programmer's productivity. A measure of its success is that more and more relational systems claim object-oriented features while at the same time relational vendors are disclaiming the interest of an ooD approach.

The data structure provided by the relational model is too simplistic for many applications. Although there is no accepted standard data model for ooD systems, these systems are in general based on richer structures previously studied in the so-called *semantic data models* [30] and the object-oriented approach brings improvements in data modelling. First, the notion of *object* elegantly fits data from the disk to the screen. For instance, in the O_2 database system, a persistent object can be displayed on the screen by a simple method application. This uniformity of thinking facilitates the design of applications. Furthermore, the data models used in ooD presents the following advantages:

1. object identity allows to describe data sharing and is essential for modelling data behavior and data evolution through time (an object keeps its identity when updated).

2. classification of objects in classes naturally leads to inheritance and therefore allows for sharing and reuse of code.

3. many models integrate “complex objects”. This is a rather vague concept which is usually meant to capture: (i) particular data types for large pieces of data such as images, and (ii) structured values built using constructors such as sets or lists. Complex objects are essential in many applications, in particular in engineering applications.

The design of applications is facilitated by semantically richer models. Also, application modelling methodologies are often object-based (e.g., the entity-relationship model is very popular). Therefore the gap between the database model and the data model of the methodology is shortened when the database model is object-based.

The other major advantage of object-orientation is that, via dynamic linking and incremental compilation, it enormously participates in simplifying the life of application programmers. These together with the benefits from using inheritance are classical advantages of object-oriented languages and although essential, will not be discussed further.

On the negative side, the object-oriented models have focused on navigational languages to the prejudice of declarative aspects. Furthermore, ooD systems lack the formal basis of relational systems. Finally, although most ooD systems offer query languages, those do not provide deductive capabilities and in most cases do not even take full advantage in a real sense of object-orientation.

The hope is that a dooD approach would yield the same advantages and correct the negative aspects of pure ooD systems.

3 DooD

We will consider various aspects of the integration of deduction and object-orientation in a database context.

3.1 Structured Objects

We consider first "structured objects" which are commonly found in ooD models. An issue is to incorporate them in deductive systems.

The data structure of the relational model was early criticized as being too simplistic for many applications [41]. Richer models were developed (e.g., [2]), a difficulty being to generalize relational calculus (to have a declarative access language) and relational algebra (to benefit from set-oriented optimization techniques).

In particular, database languages with primitives for handling finite trees, whose internal nodes denote tuple and (finite) set constructors have been intensively studied. We call such finite trees *structured objects*. Whereas, the tuple constructor is simple to add to the traditional relational data model, the set constructor requires a number of interesting new ideas since it leads to quantification over sets, a non-first-order feature. Similar set constructs are also used in general-purpose programming languages such as SETL [43].

Of interest to databases is the development of calculi, algebra and rule-based languages for such structured objects which allows to naturally incorporate them in dooD systems. The introduction of the set constructor in a rule-based language [16, 4, 34] leads to problems resembling those encountered when allowing negative literals in the body of rules. Indeed, consider the following rules (in COL):

$$\begin{array}{lcl} y \in child(x) & \leftarrow & parent(x,y) \\ child\_less\_person(x) & \leftarrow & person(x), child(x) = \emptyset. \end{array}$$

which defines a function *child* over the set of persons and a predicate *child_less_person*. This program is apparently negation-free. Clearly, its evaluation requires the use of negation. Not surprisingly, techniques such as stratification and well-founded models have also been considered in this context.

¿From a logical point of view, quantification over sets leads to non-first-order features. From a complexity point of view, the introduction of the set constructor moves from PTIME (for datalog) to EXPTIME (structured object datalog) and thus is infeasible for large databases if not somewhat controlled.

3.2 Object identity

Another issue is to incorporate object identity in deductive systems.

Sets and tuples are the core types that must be supported in object-oriented database systems [10]. The theory of structured object can therefore serve as a starting point for an object-oriented database theory. (But only a starting point, since it does not include key concepts such as "abstract types with methods" and "inheritance"). Also conspicuously missing is the notion of object identity. This was considered in an early model, the logical data model of [36]. More recently it has gained a lot of attention, e.g., [5, 19, 33]

There are a number of important issues not yet quite-understood. These include for instance typing issues and object creation. The problem in object creation is the "naming" of the newly created object. In IQL [5], LOGRES [19], LOCO [37], or LLO [38], object identities come from a unique infinite sort, and one new identifier is taken from that sort when necessary. In other formalisms (e.g., O-logic of [39] or F-logic of [33]), new objects are denoted using Skolem functions so the programmer directly controls their names.

A number of interesting theoretical issues have resulted from the introduction of object identity (e.g., with respect to duplicate elimination [5]). It should be noted that the use of created objects (from an infinite sort or based on Skolem) is a clear departure from traditional databases since it results in a potentially infinite models whereas relational database theory can be viewed (to a large extent) as a subset of finite model theory.

3.3 Query languages for ooDB

As mentioned earlier, the core of dooD (according to the author of the present note) is the introduction of declarative languages in object-oriented systems. Several declarative *query* languages for object-oriented database management systems have already been proposed in the last few years (e.g. [11, 14, 21]). A survey of that topic can be found in [26].

Most proposals consisted in relational language extensions, often extensions of relational algebra or of SQL. However, it seems now clear that object-orientation requires to consider entirely new languages and new techniques for compiling and optimizing these languages. Furthermore, notions which were quite simple in a relational context become quite involved in an object-based environment. One such notion is the concept of *view* (see [3]) which is central to declarative languages for databases. Updates also have to be reconsidered. This is the topic of the next section.

3.4 Active Rules

In this section, we consider the advantages of using rules. This is a clear advantage of the deductive approach (and not specific to dooD). However, objects introduce a new dimension to the management of rules. Also, we insist here on the use of rules not only as a means of specifying implicit information but as a tool for specifying behavior.

Relational systems provide no solution with respect to specification of behavior and focus primarily on factual updates (adding, removing or modifying a fact). One needs in many applications to model a rapidly changing world. Types (signatures) change: one may want to add or remove an attribute from a relation; one may perform more radical restructurings of the data; one may also modify the class hierarchy. The code of applications changes: new methods are introduced, methods are refined. Finally, the objects live, i.e., they are created, they are destroyed, they migrate from class to class (e.g., a student is "promoted" to Ph.D. student status).

In ooD, evolution is captured by explicit updates emerged in method code. This is not satisfactory when the application becomes too complicated (see Section 3.5). In particular, when one starts adding triggers, the general control structure of the system and the semantics of a specific application become unclear. This may result in unpredictable behavior and applications impossible to debug.

In deductive systems, rules are usually considered in deductive databases as a means of specifying implicit facts and therefore queries. When considering applications in a more general context, rules have to be upgraded to capture other aspect as well: e.g., updates or integrity constraints. However, integrity constraints are a good example of the mismatch between the theory and practice of databases. In theory, a large variety of constraints have been studied. Little of this has ever been implemented in real systems. A first reason is the cost of integrity enforcement. Another reason is that one rarely considers logical sentences in real applications. An integrity in the real world cannot be separated from the actions to be taken when a violation occurs.

Therefore, a real need is the use of rules as specification of behavior. Such rules are sometimes called triggers. Examples of triggers are:

- when balance becomes negative do ...
- when inserting in relation R do ...

Active rules also provide better organization of the application code as discussed below.

Again, this behavioral aspect of rules is not specific to dooD. However, it is crucial indeed to answer the needs of the new target applications.

To conclude this section, note that there is little theory on database evolution. One can note works in a variety of distinct fields: e.g., fixpoint logics (e.g., [6, 7]), production rule systems (e.g., [25]) or attempts to introduce updates in logic programming [42]. We believe that known formal tools such as temporal logic or more recent ones such as the Evolving Algebra of Gurevich could be used for specifying data evolution.

3.5 Knowledge independence

Even if we agree on the importance of data evolution, one may wonder why insist on using rules in place of procedural code (as used in ooD systems). The answer is *knowledge independence* which we discuss next. Knowledge independence is an advantage of the deductive approach and is not specific to dooD.

Perhaps *the* major achievement of database technology has been the separation of data and programs, the so-called *data independence*. As a result, data can be shared between several users and/or application programs and its consistency can be more easily enforced by the system. A crucial consequence is that application programs only see the logical structure of the data (logical schema). Its physical structure (the physical schema) can be modified for instance to improve performances, without affecting application programs.

However, in many applications, the "information" has other forms that simple database facts. One can also consider statements such as: *the balance of an account is positive*, or *when a balance becomes negative, start charging interest*. Unfortunately, such information is usually dispersed in the code and as a consequence almost unmanageable. An important goal of dooD technology is also to separate such information from the actual code of the application. This would facilitate maintenance: (i) such laws are now clearly isolated and (ii) they are written in a declarative language, and so, are more readable.

3.6 Logic programming with object-orientation

This direction was initiated in the database context by Maier with O-logic and was pursued in a series of new "logics" such as C-logic and F-logic (see [33]). In these languages, the semantics of rule control is in the spirit of Prolog.

These proposals have to be related to important efforts to provide formal foundations to object-oriented programming such as for instance the works of Cardelli [20] or Ait-Kaci [8]. For instance, the Ψ-terms of Login [8] are utilized in $TEDM$ [40] an object-oriented database system with deductive reasoning on objects.

3.7 Formal foundations of object-oriented databases

We have found few motivations so far that were specific to dooD. Most features that we discussed were directly imported from the ooD approach or from the deductive one. We conclude this section with a most important motivation for dooD: providing a formal foundation to ooD. By moving towards more and more complex models and notably by adopting object-orientation with the complexity of object identity, inheritance, overwriting, etc., the applications tend to become unmanageable: hard to design, hard to understand, hard to debug, hard to maintain, etc. The user requires more and more help in terms of environment. A declarative language with a formal and sound semantics clearly participates in the effort of simplification.

4 Developing a dooD system

We have seen the advantages of using a dooD. We now briefly consider the technological possibilities for developing such systems:

1. Relational system extensions
   Since relational systems proved to be very successful, it seems natural to try to extend them with the new features. Indeed, this is already proposed by relational vendors to a certain extent. For instance, Postgres has some deductive capabilities based on triggers. However, although the relational foundations were sound and clear, standard SQL already suffers from being a mixture of algebraic, logic and functional features. Ad-hoc extensions on such basis would be quite inelegant. Furthermore, the core engines of relational systems cannot be easily modified to incorporate object-orientation and/or deduction.

2. OoD system extensions
   The advantages of object-orientation were achieved at the cost of introducing procedurality in database languages. The evolution towards more declarative languages would probably be perceived in the ooD community as a loss of direct control on data and require a change in mentality. Furthermore, It is unclear how to incorporate in ooD deductive capabilities and appropriate optimization techniques to handle deduction. The lack of formal foundations for ooD does not facilitate the task.

3. Deductive database system extensions
   The problem is that no such system exists as a product. One can of course build up on top of a research prototype but this would require important efforts of rewriting. From a logical viewpoint, this

seems a most feasible approach. However, from a technical one, a major issue is to introduce object-orientation with good performances. One cannot for instance carry the weight of late binding to the deductive engine. Even if deduction is fast, this would be prohibitive when a large number of tuples are concerned. It seems therefore necessary to introduce oo-optimization techniques as used in C++ *inside* the deductive engine and this remains an unsolved issue.

4. Kbms and Expert Systems
   Knowledge-base systems also offer solutions to the modelling of complex data applications. However, many attempts to loosely couple these technologies and database technology have failed in the past. The major handicaps are (i) poor performances because of the limited optimization of rule processing, and (ii) too heavy cost of the transfer of data back and forth between the deductive engine and the persistent store. For these reasons, such systems are rarely used in production. They did not provide convincing alternatives to deductive database. They will not provide an answer to dooD.

5. Starting from scratch
   One does not have (anymore) to start (entirely) from scratch when designing a database system. In particular, one can build on an available object store. But even with such a tool, the design of a dooD system remains a considerable task and a risky venture.

If some solutions such as building on an expert system shell or a relational engine were put aside (by the author of these notes), several feasible solutions can be considered. It is likely that they will all be pursued and that time will determine the winner(s).

5 Conclusion

In the case of relational DBMS's, there has been the rare convergence (in the sixties and seventies) of intense experimental activities and theoretical studies. The situation is less clear now since with notable exceptions, the experimental activities over the last five years have been concentrated around ooD systems whereas the theoretical ones preferred the deductive paradigm. This resulted in a theory with almost no implementation and systems with fragile theoretical basis. DooD are an attempt to reconcile the two approaches.

DooD systems are complex because of the many features they offer and because of the complexity of integrating these features. As mentioned above, a new generation of database systems comes from the convergence of more advanced modelling and of the technology to support them. We believe that the gain in productivity and in particular in flexibility motivates the move to dooD technology in a variety of application domains when such systems will become available. Advanced models incorporating object-oriented features and deduction are beginning to be understood and the formal basis is slowly emerging. The implementation of object-oriented features is now beginning to be mastered, and the technology needed to optimize data intensive deduction is now available. All the ingredients for the future dooD systems are almost in place. The *only* remaining issue is the design of such a system and it can now be tackled. Whether it will be built on top of an existing ooD system, deductive system or constructed from scratch remains to be seen and is probably not the most interesting issue.

We have mentioned many reasons for considering dooD systems: In short, this is to get the best of worlds by technology integration. On the negative side, one may be tempted to believe that the triangular integration of database, logic-programming and object-oriented technologies would lead to a monster. This may indeed be the case. We believe optimistically that to the contrary, (i) object-orientation provides the basis for incorporating deductive capabilities in *real* database systems, and (ii) the logical framework is the natural candidate for providing a sound theoretical basis to ooD's.

Acknowledgments

We thank Laurent Vieille for discussions on the topic; and Sophie Cluet, Claude Delobel and Tova Milo for criticizing a first draft of this paper.

References

[1] S. Abiteboul. Towards a deductive object-oriented language. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 453–472. Elsevier Science Publishers B.V., 1990.

[2] S. Abiteboul and C. Beeri. On the manipulation of complex objects. Technical report, INRIA and Hebrew Univ., 1988.

[3] S. Abiteboul and A. Bonner. Objects and views. In *Proc. ACM SIGMOD Symp. on the Management of Data*, 1991.

[4] S. Abiteboul and S. Grumbach. A Rule-Based Language with Functions and Sets. ACM Trans. on Database systems, 16(1), pages 1–30, 1991.

[5] S. Abiteboul and P.C. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 159–173, 1989. to appear in *J. ACM.*

[6] S. Abiteboul and V. Vianu. A transaction language complete for database update and specification. In *Proc. ACM Symp. on Principles of Database Systems*, pages 260–268, 1987.

[7] S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer and System Sciences*, 41:181–229, 1990.

[8] H. Ait-Kaci. Login: a logic programming language with built-in inheritance. *J. Logic Programming*, 3:185–215, 1986.

[9] A.M. Alashqur, S.Y.W. Su, and H. Lam. A rule-based language for deductive object-oriented databases. In *International Conf. on Data Engineering*, pages 58–67, 1990.

[10] F. Bancilhon. Object-oriented database systems. In *Proc. ACM Symp. on Principles of Database Systems*, pages 152–162, 1988.

[11] F. Bancilhon, S. Cluet, and C. Delobel. Query languages for object-oriented database systems: the O_2 proposal. In *proc. Second Intern. Work. on Data Base Programming Languages*, 1989.

[12] F. Bancilhon, D. Maier, Y. Sagiv, and J.D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–15, 1986.

[13] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing. In *Proc. ACM SIGMOD Symp. on the Management of Data*, pages 16–52, 1986.

[14] J. Banerjee, W. Kim, and K. Kim. Queries in object-oriented databases. Technical Report DB 188-87, MCC, Austin, Texas, June 1987.

[15] C. Beeri and T. Milo. Functional and predicative programming in oodb's. In *Proc. ACM Symp. on Principles of Database Systems*, pages 176–190, 1992.

[16] C. Beeri, S. Naqvi, R. Ramakrishnan, O. Shmueli, and S. Tsur. Sets and negation in a logic database language (LDL1). In *Proc. ACM Symp. on Principles of Database Systems*, pages 21–37, 1987.

[17] E. Bertino D. Montesi. Toward a logical-object oriented programming language for databases. In *edbt*, pages 168–183, 1992.

[18] N. Bidoit. *Bases de Donnees Deductives.* A. Colin, Paris, 1992.

[19] F. Cacace, S. Ceri, S. Crespi-Reguizzi, L. Tanca, and R. Zicari. The Logres project: integrating object-oriented data modelling with a rule-based programming paradigm. Technical Report 89-039, Politecnico di Milano, 1989.

[20] L. Cardelli. Amber. In B. Robinet G. Cousineau, P.L. Curien, editor, *Combinators and Functional programming languages*, page 1986. Springer-Verlag, New-York.

[21] M. Carey, D. DeWitt, and S. Vandenberg. A data model and query language for exodus. In *Proc. SIGMOD, Chicago, Illinois*, 1988.

[22] Y. Caseau. an object-oriented deductive language. *Annals of Mathematics and Artificial Intelligence*, 3:211–258, 1991.

[23] S. Ceri, G. Gottlob, and L. Tanca. *Logic programming and databases.* Springer-Verlag, Berlin Heidelberg, 1990.

[24] S. Danforth, E. Simon, F. Cacace, S. Ceri, and L. Tanca. the design of the RL language. Technical Report 2443, Esprit, 1990.

[25] C. de Maindreville and E. Simon. Modelling non-deterministic queries and updates in deductive databases. In *Proc. of Intl. Conf. on Very Large Data Bases*, 1988.

[26] C. Delobel. Evaluating query languages for oject-oriented database systems. Technical report, University Paris, 1992.

[27] H. Gallaire, J. Minker, and J.-M. Nicolas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–185, 1984.

[28] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–11, 1989.

[29] A. Heuer and P. Sander. Semantics and evaluation of rules over complex objects. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 473–492. Elsevier Science Publishers B.V., 1990.

[30] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19:201–260, 1987.

[31] M. Jarke and M. Jeusfeld. Rule representation and management in ConceptBase. *Sigmod Record*, 18:3:46–51, 1989.

[32] P. C. Kanellakis. Elements of relational database theory. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 1074–1156. North Holland, 1991.

[33] M. Kifer and G. Lausen. F-logic: A higher-order language for reasoning about objects. In *sigmod*, 1989.

[34] G.M. Kuper. Logic programming with sets. In *Proc. ACM Symp. on Principles of Database Systems*, pages 11–20, 1987.

[35] G. Kuper and M.Y. Vardi. A new approach to database logic. In *Proc. ACM Symp. on Principles of Database Systems*, pages 86–96, 1984.

[36] G. Kuper and M.Y. Vardi. On the complexity of queries in the logical database model. In *2nd Inter. Conf. on Database Theory*, pages 267–280, 1988.

[37] E. Laenens, B. Verdonk, D. Vermeir, and D. Sacca. the LOCO language: towards an integration of logic and object-oriented programming. Technical Report 90-09, U. of Antwerpen, 1990.

[38] Y. Lou and Z. M. Ozsoyoglu. LLO: an object-oriented deductive language with methods and method inheritance. In *sigmod*, pages 198–207, 1991.

[39] D. Maier. A logic for objects. In *Workshop on foundations of deductive databases and logic programming*, pages 6–26, 1986.

[40] D. Maier, J. Zhu, and H. Ohkawa. Features in the TEDM object model. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 511–530. Elsevier Science Publishers B.V., 1990.

[41] A. Makinouchi. A consideration of normal form of not-necessarily-normalized relations in the relational data model. In *Proc. of Intl. Conf. on Very Large Data Bases*, pages 447–453, 1977.

[42] S. Manchanda and D.S. Warren. A logic-based language for database updates. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363–394. Morgan-Kaufmann, 1988.

[43] J.T. Schwartz, R.B.K. Dewar, E. Dubinsky, and E. Schonberg. *Programming with Sets: An Introduction to SETL*. Springer-Verlag, New-York, 1986.

[44] J.D. Ullman. *Principles of Database and Knowledge Base Systems: Volume I and II*. Computer Science Press, 1988.

[45] L. Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science*, 69(1):1–53, 1987.

[46] L. Wong. Inference rules in object-oriented programming systems. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 493–509. Elsevier Science Publishers B.V., 1990.

[47] C. Zaniolo. Object identity and inheritance in deductive databases – an evolutionary approach. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, pages 7–21. Elsevier Science Publishers B.V., 1990.