Some Recent Developments in Deductive Databases

Shalom Tsur
Microelectronics and Computer Technology Corporation
Austin, Texas, 78759

1. Introduction

Deductive (or logical) databases have become over the last years a lively field of research. The initial research results have been disseminated and utilized in building the first generation of deductive database prototypes. These systems, in turn, are being used now for experimentation and application development. While the term *Deductive Database* has no precise definition, it is typically associated with a system having the following properties:

- The capability to express queries by means of logical rules.
- Support for recursive queries and efficient algorithms for their evaluation against the stored data.
- Advanced features e.g., stratified negation.
- Support for an underlying data domain that includes complex objects and sets.
- Optimization methods that guarantee the translation of the declarative specifications into efficient access plans and their termination when executed.

In addition, these systems provide support for the usual database amenities such as schema specification, data sharing, transaction management, recovery and updates as a set-at-a-time.

The purpose of this short article is not to provide a comprehensive review of the field but rather to point out, by means of a few selected references, the new directions towards which the field is moving. Recognizing that different readers may have different interests we have included references from theory, system and application development.

This review first provides a brief perspective and then moves to the discussion of a few recent papers that in this authors' opinion are indicative of the new directions taken.

2. Perspective

The initial efforts, that led to the development of deductive databases as a new technology, stemmed from two sources:

- The problem of recursive query specification and evaluation and the recognition that this problem was beyond the power of relational technology.
- The desire to combine the relational database and logic programming technologies and the recognition of the difficulties that this combination entailed. In particular, the "impedance mismatch" problem.

These problems led to the development of a fixpoint semantics of logic programs and the subsequent development of efficient compilation techniques from the logic-based descriptions to efficient data access plans. Recursive queries were a subject of particular interest and the early literature is marked by a large number of publications describing different schemes for the evaluation of such queries. We will not attempt to review all of these here. The reader is referred to [Ull89] for a comprehensive review of these topics. Many of the papers on these topics were published in the proceedings of the SIGMOD, PODS, VLDB and Logic Programming Conferences over the last years. An additional collection of research papers covering different aspects of deductive databases appears in [NaTs91].

3. Current Work

Current theoretical work concentrates mostly on developing more powerful semantics that can be associated with the logic program descriptions. Early work led to the development of the T_P operator and its fixpoint when applied to a program P to capture the semantics of a logic program. Intuitively, this means that the rules of the program are applied to the data, the derived results in the rule-heads are included and reapplied until no new results are derived. The use of negation in this context is permissible and consistent with this monotonic derivation process if the program P is *stratified*, meaning that the negated predicate in a rule body must first be derived in its positive (i.e., non-negated) form. Thus, for example the program†

$$even(X) \leftarrow -odd(X).$$
$$odd(1).$$
$$odd(Y) \leftarrow odd(X), Y = X + 2.$$

is stratified since the *odd* predicate is derived in its positive form prior to the derivation of *even*. On the other hand,

$$even(0).$$
$$even(Y) \leftarrow -even(X), Y = X + 1.$$

is non-stratified since *even* is defined in terms of its own negation. It is thus impossible, under this interpretation, to use negation in the definition of a recursive rule. Yet there are situations in which the use of negation as in a non-stratified way is useful. For example, consider a game in which the permissible moves between positions are given by the $move(X,Y)$ relation and the definition of the winning positions by the program

$$win(X) \leftarrow move(X,Y), -win(Y).$$

i.e., X is a winning position if for all positions Y that can be arrived from X, Y is not a winning position. Van Gelder [VG89] defines an alternative semantics to the T_P operator entitled "alternating fixpoint semantics" that is sufficiently powerful to capture the meaning of rules of this type. In his paper he also cites a number of related results, all designed to overcome the stratification limitation.

Another extension along the same lines is proposed by Ganguly, Greco and Zaniolo [GGZ91] in case of the derivation of maximum and minimum predicates. Consider for example the derivation of the minimum cost path in a labeled graph given by the relation arc:

$$min\_path(X,Y,C) \leftarrow min(C,X,Y,path(X,Y,C)).$$
$$path(X,Y,C) \leftarrow arc(X,Y,C).$$
$$path(X,Y,C) \leftarrow path(X,Z,C1), arc(Z,Y,C2), C = C1 + C2.$$

Under the stratified interpretation of this program first *all* paths and their costs between each pair of nodes X,Y are derived and then the minimum cost is selected. Clearly this is an inefficient method of derivation and a non-stratified solution could be thought of in the form

$$min\_path(X,Y,C) \leftarrow path(X,Y,C), -(path(X,Y,C1), C1 < C).$$

i.e., the minimal cost is C if there is no path between X and Y with a cost $C1$ such that $C1 < C$. The paper proposes a greedy fixpoint evaluation that pushes the extrema in the recursion and shows that the method corresponds to the known algorithmic solution to this problem by Dijkstra. This paper opens the intriguing possibility that known algorithmic results may be used in the evaluation declaratively specified problems.

Another direction in which the field evolves is the fusion of ideas from the object-oriented databases and languages with deductive databases. This activity is known under the term "deductive object-oriented database (DOOD) systems." Deductive database systems provide a powerful declarative tool for problem specification. Yet they are limited in terms of the data-structures and types that can

† In this program it is assumed that the value of X is given with the query e.g., $?even(100)$.

be used with these specifications. Object-oriented systems, on the other hand, support powerful mechanisms for the specification of complex data structures, type hierarchies and classification systems with inheritance but are weak in their declarative capabilities. DOOD systems attempt thus to combine the best of both worlds. The paper by Kifer and Wu [KW89] is representative of current research in this area. They present a logic which is designed to accommodate object-oriented notions within the declarative framework. ar From the systems' perspective, we noted already that some second-generation prototypes are presently in development. These systems incorporate the lessons learned from the initial prototype designs. Perhaps the most important lesson learned is that for a variety of practical reasons, it is necessary to supplement the declarative specifications supported by a deductive database with procedural extensions. These can be used to specify those parts of the problem in which the order of execution is a part of the specification, as e.g., in updates. Other reasons for using procedural extensions include the use of existing programs within a declarative specification. For example, a library of statistical routines that for practical reasons will be used as is. Typically these procedural extensions appear to the logic program as evaluable predicates, usable in rule-bodies, usable with a number of different argument binding patterns. The paper by Phipps, Derr and Ross [PDR91] is a description of a second-generation deductive database system. Their paper describes the GLUE procedural extension to the NAIL! system and in addition, describes some other constructs such as sets, meta-programming constructs and modules. Another project having similar objectives is LDL++ [LDL++], under development at MCC, which is an attempt to absorb the lessons learned from the LDL system.

Lastly, initial studies of applications that appear to benefit significantly from the deductive database technology are under way. These applications include bill-of-materials, exploratory data analysis, scientific database problems and others. For a brief review of some of these see Tsur [Ts91].

4. Conclusion

In this short note we have attempted to point out some of the salient developments in the field of deductive databases. From the outset the field was marked by a close collaboration between theoreticians and practitioners. The result is a cycle by which theoretical results are assimilated in prototypes, these prototypes are used for application development which in turn, give rise to new problems worthy of theoretical treatment. At this time we do not see an end to this fruitful relationship and expect it to increase as the technology is reaching the threshold of commercial development. Some problems of the field have not been mentioned. In particular, all aspects related to performance measurement and evaluation of deductive databases. Activity in this area is almost non-existent but is important. In particular, the creation of benchmarks of representative rule-sets and queries for the evaluation and efficiency comparison of the recursive query compilation schemes in use. It is hoped that this note will provide an impetus to the creation of this activity.

5. References

[NaTs91] *Annals of Mathematics and Artificial Intelligence,* Volume 3 (1991), No. 2-4. Special Issue on Deductive Databases. J.C. Balzer Scientific Publishing Company, Basel--Switzerland.

[GGZ91] Ganguly, S., Greco, S., and C. Zaniolo. *Minimum and Maximum Predicates in Logic Programming.* Proc. Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. May 29--31 1991, Denver, CO., pp. 154--163.

[KW89] Kifer, M. and J. Wu *A logic for Object-Oriented Programming (Maier's O-Logic Revisited).* Proc. Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29--32, 1989, Philadelphia, PA., pp. 379--393.

[NT89] Naqvi, S., and S. Tsur. *A Logical Language for Data and Knowledge Bases. W. H. Freeman, 1989.*

[PDR91] Phipps, G, Derr M.A., and K.A. Ross *Glue-Nail: A Deductive Database System.* Proc. 1991 ACM SIGMOD International Conference on Management of Data, Denver CO., May 29--31, pp. 308--317.

[Ts91] S. Tsur. *Deductive Databases in Action.* Proc. Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. May 29--31 1991, Denver, CO., pp. 142--153.

[Ull89] J.D. Ullman, *Principles of Database and Knowledge-Base Systems, Volume 2: The New Technologies.* Computer Science Press, 1989.

[VG89] A. Van Gelder, *The Alternating Fixpoint of Logic Programs with Negation,* Proc. Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29--32, 1989, Philadelphia, PA., pp. 1--10.

[LDL++] Arni, N., Ong, K., Tsur S., and C. Zaniolo. *The Design of the LDL++ System.* Invited Talk, Workshop on Deductive Databases, San Diego, Oct 31, 1991.