Deductive Databases: Achievements and Future Directions

Jeffrey D. Ullman

Stanford University, Stanford, California

Carlo Zaniolo

MCC, Austin, Texas

Abstract

In the recent years, Deductive Databases have been the focus of intense research, which has brought dramatic advances in theory, systems and applications. A salient feature of deductive databases is their capability of supporting a declarative, rule-based style of expressing queries and applications on databases. As such, they find applications in disparate areas, such as knowledge mining from databases, and computer-aided design and manufacturing systems.

In this paper, we briefly review the key concepts behind deductive databases and their newly developed enabling technology. Then, we describe current research on extending the functionality and usability of deductive databases and on providing a synthesis of deductive databases with procedural and object-oriented approaches.

1 Motivations

There are a number of applications that have a database "flavor," and yet are not well-addressed by conventional database management systems. Examples of such applications are

1. Computer-aided design and manufacturing systems,
2. Scientific databases, often involving feature detection and extraction, such as studies involving chemical structures (e.g., the human genome), or analysis of satellite data.

In addition to the traditional requirements of databases (such as integrity, sharing and recovery), these new applications pose demands that are not answered by conventional DBMS, such as the following:

- *The need to deal with complex structures and recursively defined objects.* For example, a VLSI CAD system typically allows the definitions of "cells," which are designs having other cells as subparts. Operations on such a design often begin by expanding out the design, say, to create a checkplot. The expansion must be carried on to arbitrary depth, so a recursive logic program is appropriately used to define the operation of cell expansion.

- *The need for active database components.* For example, in a software engineering database, a change to one aspect of the design may trigger changes to other components.
- *The need to support browsing and complex ad-hoc queries.* For example, a medical researcher may wish to examine a database of medical histories to test a variety of hypotheses about possible causes of diseases.

Conventional Database Systems fail to address these needs, and in addition, they suffer from the limited power of their query languages. Since conventional query languages, as exemplified by SQL, are only capable of accessing and modifying data in limited ways, database applications are now written in a conventional language with intermixed query language calls. But since the nonprocedural, set-oriented computational model of SQL is so different from that of procedural languages, and because of incompatible data types, an "impedance mismatch" occurs that hinders application development and causes expensive run-time conversions. It has thus become generally accepted that for applications at the frontier we need a single, computationally complete language that answers the needs previously discussed and serves both as a *query language* and as a *general-purpose host language.*

Object-oriented systems, where the database is closely integrated with languages such as Smalltalk or C++, address many of the previous requirements, and support useful concepts, such as object-identity and a rich type structure with inheritance of properties from types to their subtypes. The main limitation of object-oriented systems is that, for application development, they are heavily dependent on procedural languages, even though some systems offer a limited declarative query capability. Now, relational databases, have demonstrated the desirability of using a declarative logic-based language, whereby substantial portions of the algorithm required to meet a user's request are left to the system. This ability is essential for ease of use, data independence and code reusability. Therefore, deductive databases take the declarative approach in addressing those requirements: they provide a *declarative, logic-based* language for expressing *queries, reasoning,* and complex *applications* on databases.

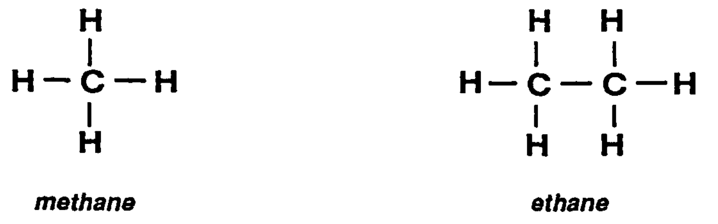
2 Declarative Programming

The declarative nature of deductive database languages manifest itself in two important ways:

1. The order in which goals are written in the rules does not determine their actual execution order, which is controlled by the system rather than the programmer.
2. The selection between forward-chaining and backward-chaining execution is automatic—it is done by the system, rather than the programmer.

Point 1 generalizes to rule-based languages the nonnavigational paradigm of relational query languages, where select/join expressions are executed in an order chosen by the query optimizer according to performance considerations (pertaining, e.g., to the selectivity of the various conditions and the availability of indexes.) As in relational systems, this promotes ease of use, since the programmer is relieved of performance-related concerns, such as navigating through the database and using access structures; furthermore, data independence is greatly enhanced since the resulting code can be reused even after physical changes have occurred in the database.

Point 2 is one of the most novel features of deductive databases, which sets them apart from current rule-based systems. Some of the latter, such as OPS5, only support forward-chaining; others, such as Prolog, only support backward-chaining. Some expert system shells support both, but the programmer must select the better strategy for the situation at hand, and code it as part of the actual program. In systems such as $\mathcal{LDL}$ [NaTs] and NAIL! [Meta], instead, the system will make the proper choice for the user, who can now focus on logical correctness rather than execution strategy. The significance of this point may be better illustrated by an example. A methane molecule consists of a carbon atom linked with four hydrogen atoms. An ethane molecule can be constructed by replacing any H of a methane by a carbon with three Hs. The respective structure of methane and ethane molecules are as follows:



More complex alkanes can be obtained inductively in the same way: that is, by replacing an H of a simpler alkane by a carbon with three Hs. We can now define alkanes using Horn clauses. A methane molecule will be represented by a complex term `carb(h, h, h)`, and an ethane molecule by `carb(h, h, carb(h, h, h))`. In general, alkane molecules can be defined inductively as follows:

```
all_mol(h, 0, Max).
all_mol(carb(M1, M2, M3), N, Max) ←
                                    all_mol(M1, N1, Max),
                                    all_mol(M2, N2, Max),
                                    all_mol(M3, N3, Max),
                                    N= N1+N2+N3+1, N <= Max.
```

In addition to defining alkanes of increasing complexity, these nonlinear recursive rules count the carbons in the molecules, and ensure finiteness in their size and number by ensuring that the tally of carbons never exceeds a Max. This example illustrates the need for recursion in representing complex objects, and the simplicity and versatility of declarative programming. Indeed, our alkane definition can be used in different ways. For example, to generate all molecules with no more than four carbons, one can write

```
? all_mol(Mol,Cs, 4).
```

To generate all molecules with exactly four carbons, one will write

```
? all_mol(Mol, 4, 4).
```

Furthermore, if the relation `alk(Name, Str)` associates the names of alkanes with their structure, the following rule will compute the number of carbons for an alkane given its name (assuming that 10000 is a large enough number for all molecules to have a lower carbon complexity):

```
find(Name, Cs) ← alk(Name, Str), all_mol(Str, Cs, 10000).            (1)
```

The first two examples can be supported through a forward-chaining computation, which in turn translates naturally to the least-fixpoint computation that defines the model-theoretic semantics of recursive Horn-clause programs [Llo, NaTs]. The least-fixpoint computation amounts to an iterative procedure, where partial results are added to a relation until steady state is reached. Because of its simplicity, this execution model is more suitable for handling data on secondary store than backward-chaining, which leads to main-memory-based, stack-oriented implementations. Thus deductive databases support well the first two examples, whereas Prolog and other backward-chaining systems would fail. In the last example, however, the first argument, `Str`, of `all_mol` is bound to the values generated by the predicate `alk`. Thus a computation, such as Prolog's backward-chaining which recursively propagates these bindings, is significantly more efficient than forward chaining. Deductive databases solve this problem equally well by using techniques such as the *Magic-Sets Method*, or the *Counting Method* that simulate backward-chaining trough a pair of least-fixpoint computations [BMSU, SaZ1, SaZ2]. For rule (1), the magic-sets method will produce the following modified definition of `all_mol`:

```
all_mol(h, 0, Max) ← m_all_mol(h, Max).
all_mol(carb(M1, M2, M3), N, Max) ←
                                    all_mol(M1, N1, Max),
                                    all_mol(M2, N2, Max),
                                    all_mol(M3, N3, Max),
                                    N= N1+N2+N3+1, N <= Max,
                                    m_all_mol(carb(M1, M2, M3), Max)).
```

Thus, the magic-sets method rewrites the original rules by adding the magic predicate `m_all_mol`. This is defined on the arguments that would be bound in a backward-chaining execution. (In rule (1) the first argument and the third one of `all_mol` are bound, and these bindings propagate in the backward-chaining execution of the original `all_mol` rules.) Then, our magic predicate is defined as follows (in actual systems the determination of bound arguments and the resulting generation of magic rules and modified rules are done at compile-time [Ull]):

```
m_all_mol(Str, 10000).
m_all_mol(M1, Max) ← m_all_mol(carb(M1, M2, M3), Max).
m_all_mol(M2, Max) ← m_all_mol(carb(M1, M2, M3), Max).
m_all_mol(M3, Max) ← m_all_mol(carb(M1, M2, M3), Max).
```

Once the variable `Str` in the first rule is initialized with a value passed down from rule (1), the recursive magic rules construct all subcomponents for such a `Str`. For an ethane molecule `Str = carb(h, h, carb(h, h, h))`, the magic rules produce the molecule itself, and its subcomponents: `carb(h, h, h)` and `h`. Thus the magic predicate in the modified rules ensures that no molecule is returned unless it is one of these three. Therefore the fixpoint computation of `all_mol` using these rules completes in three iterations.

Deductive databases handle cycles automatically and efficiently. This is a most useful feature since cyclic graphs are often stored in database relations, and derived relations can also be circular. In our alkane example there are many equivalent representations for the same alkane. To generate these, equivalence-preserving operations are used, such as rotation and permutation on the molecules—but repeated applications of these operations bring back the initial structure. When using a language such as Prolog, the detection of cycles must be built into the program at the price of complications and inefficiency (e.g., by carrying along a bag with all solutions). In deductive databases, the checking of new solutions against the set of old ones is automatically performed as part of the fixpoint computation.

Research on deductive databases has also contributed to areas such as nonmonotonic reasoning and knowledge representation by extending the declarative semantics of Horn Clauses (based on the concepts minimal model and least-fixpoint [Llo, NaTs]) to nonmonotonic constructs such as negation and sets. Concepts, such as stratification [VG1, ApBW, Prz2, Naq], well-founded models [VRS, VG2], and stable models [GeLi] have shed new light on various aspects of nonmonotonic reasoning and knowledge representation [MaSu, Prz2], and have also provided formal semantics to seemingly unrelated concepts such as nondeterminism [SaZ3]. Many of these theoretical contributions had a practical impact: current deductive database systems provide efficient support for stratified negation, which is more powerful than Prolog's negation-by-failure; work is progressing on finding efficient ways to support more powerful semantics (e.g., well-founded models).

One of the most interesting aspects of programming with a declarative language is debugging. The trace-based approach taken by debuggers of procedural languages and Prolog is not applicable here, since the actual execution is controlled by the system and takes place in an order that might not resemble that of the original program. On the other hand, the declarative semantics makes it possible to build a truly logical debugger. For example, the $\mathcal{LDL}$ system provides a `why` and `whynot` explanation capability, whereby the system carries out a conversation with the user explaining why a certain tuple was part of the answer, while another was missing [ShTs]. Again, a person's attention can focus on the logical correctness of a program rather than on its physical behavior.

3 Systems and Applications

Space constraints preclude us from discussing various prototypes, such as [Boc, Ceta, CeGT, KiMS, LeVi, Meta, RaSh], and comparing their architectures. Many of these systems [Ceta, Meta], however, share a common trend, namely tight coupling with existing relational databases and SQL servers. Thus, most deductive database systems position themselves as extensions to and improvements of existing relational databases rather than as their replacement. (However, it has become normal for deductive database systems to trade some of the requirements of conventional database systems, such as absolute resilience, for performance improvements.) This evolutionary approach, combined with the ability to access existing databases, has been found to be critical for making successful inroads into real-life applications. These include traditional database applications, such as bills of materials and various inventory control functions that are poorly served by current relational systems, as well as applications from new areas. The latter include scientific applications, e.g., in the molecular biology area [Tsur], semantic prototyping from E-R based specifications [Teta], and data dredging and complex analysis [Tsur]. The many applications that emerged in the short while since a viable prototype was completed, suggest that deductive database systems offer significant practical benefits in several areas, and that their use can spread rapidly as systems become more usable and their role is better understood.

A second ingredient found critical in many applications is an open and extensible architecture. For instance, $\mathcal{LDL}$ applications compile into equivalent C programs that are then linked with external routines with close coupling of data structures. This makes it easier to build on existing software, to enhance performance by coding critical rules or predicates in a procedural language, and to extend the language by introducing meta-level predicates as externals [CGK].

4 Future Directions

Deductive databases have made great strides in the last five years, in terms of theory, systems and applications, and their technology is now mature enough for commercial deployment. They also remain a vibrant field of research marked by a close interaction between theoretical and practical problems. Current work, for instance, addresses the problem of finding an attractive confluence of the declarative logic-based paradigm with the object-oriented and procedural paradigm to support a superior environment for the next generation of database applications. For instance, works, such as [KiLa, KiWu, Zani] have demonstrated the feasibility and/or desirability of merging the O-O paradigm— with notions such as object identifiers, inheritance and methods—with logic. Also, for all its merits, a declarative formulation cannot compete with the cogency and optimality of textbook algorithms for specific problems. These situations call for a mixed programming mode, and for the harmonious cooperation between the two modes at the language and system levels. We are looking forward to a new generation of deductive database systems that embody these new advances, along with the know-how acquired in building the first generation of research prototypes.

5 Conclusion

Aiming to extend relational databases while preserving their declarative programming style, deductive databases support a rule-based language capable of expressing complete applications. We believe that this technology will improve substantially the ability of database systems to cope with future demands, such as expressing very complex exploratory queries to identify elaborate patterns in large databases, and dealing with heterogeneous distributed databases.

Acknowledgments

The alkane application was written by Fosca Giannotti and based on an example by David Turner.

References

[ApBW] Apt, K., H. Blair, A. Walker, "Towards a Theory of Declarative Knowledge," in *Foundations of Deductive Databases and Logic Programming*, (Minker, J. ed.), Morgan Kaufman, Los Altos, 1987.

[BMSU] Bancilhon, F., D. Maier, Y. Sagiv, J. Ullman, "Magic Sets and other Strange Ways to Implement Logic Programs", *Proc. 5th ACM SIGMOD–SIGACT Symp. on Principles of Database Systems*, 1986.

[Boc] Bocca, J., "On the Evaluation Strategy of Educe," *Proc. 1986 ACM–SIGMOD Conference on Management of Data*, pp. 368–378, 1986.

[Ceta] Chimenti, D. et al., "The LDL System Prototype," *IEEE Journal on Data and Knowledge Engineering*, March 1990.

[CGK] Chimenti, D., R. Gamboa and R. Krishnamurthy. "Towards an Open Architecture for LDL," *Proc. 15th VLDB*, pp. 195-203, 1989.

[CeGT] Ceri, S., G. Gottlob and Tanca, "*Logic Programming and Databases*," Springer Verlag, 1990.

[GeLi] Gelfond, M., Lifschitz, V., "The Stable Model Semantics for Logic Programming", *Proc. 5th Int. Conf. and Symp. on Logic Programming*, MIT Press, Cambridge, Ma, pp. 1070-1080, 1988.

[KiLa] Kifer, M. and Lausen, G., "F-Logic:A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme" *Proc. ACM SIGMOD Int. Conference on Management of Data*, pp.134-146, 1989.

[KiMS] Kiernan, G., C. de Maindreville, and E. Simon "Making Deductive Database a Practical Technology: a step forward," *Proc. ACM–SIGMOD Conference on Management of Data*, pp. 237-246, 1990.

[KiWu] Kifer, M. and J. Wu, "A Logic for Object-Oriented Programming (Maier's O-logic Revisited)", *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Programming*, 1989.

[KrN1] Krishnamurthy and S. Naqvi, "Non-Deterministic Choice in Datalog," *Proc. 3rd Int. Conf. on Data and Knowledge Bases*, June 27–30, Jerusalem, Israel, 1988.

[LeVi] Lefebvre, A. and Vieille, L. "On Deductive Query Evaluation in the DedGin System," *Proc. 1st Int. Conf. on Deductive and O-O Databases*, Dec. 4-6, Kyoto, Japan, 1989.

[Llo] Lloyd, J. W., *Foundations of Logic Programming*, Springer Verlag, (2nd Edition), 1987.

[MaSu] Marek, V. and V.S. Subramanian, "The Relationship between Logic Program Semantics and Non-Monotonic Reasoning," *Proc. 6th Int. Conference on Logic Programming*, pp. 598-616, MIT Press, 1989.

[Meta] Morris, K. et al. "YAWN! (Yet Another Window on NAIL!), *Data Engineering*, Vol.10, No. 4, pp. 28–44, Dec. 1987.

[Naq] Naqvi, S. "A Logic for Negation in Database Systems," in *Foundations of Deductive Databases and Logic Programming*, (Minker, J. ed.), Morgan Kaufman, Los Altos, 1987.

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

[Prz1] Przymusinski, T., "On the Semantics of Stratified Deductive Databases and Logic Programs", in *Foundations of Deductive Databases and Logic Programming*, (Minker, J. ed.), Morgan Kaufman, Los Altos, 1987.

[Prz2] Przymusinski, T., "Non-Monotonic Formalism and Logic Programming," *Proc. 6th Int. Conference on Logic Programming*, pp. 656-674, MIT Press, 1989.

[RaSh] Ramamohanarao, K. and J. Sheperd, "Answering Queries in Deductive Databases", *Proc. 4th Int. Conference on Logic Programming*, pp. 1014-1033, MIT Press, 1987.

[SaZ1] Saccá D., Zaniolo, C., "Implementation of Recursive Queries for a Data Language based on Pure Horn Logic," *Proc. Fourth Int. Conference on Logic Programming*, Melbourne, Australia, 1987.

[SaZ2] Saccá D., Zaniolo, C., "The Generalized Counting Method for Recursive Logic Queries," *Journal of Theoretical Computer Science*, 61, 1988.

[SaZ3] Saccá D., Zaniolo, C., "Stable Models and Non-Determinism in Logic Programs with Negation," *Proc. 9th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1990.

[ShTs] Shmueli O. and S. Tsur, "Logical Diagnosis of LDL Programs", in *Logic Programming: Proc. of the Seventh International Conf.*, pp. 112-129, The MIT Press, 1990.

[Teta] Tryon, D., et al. "System Analysis for Deductive Database Environments: an Enhanced role for Aggregate Entities," *Proc. 9th Int. Conference on Entity-Relationship Approach*, Lausanne, CH, Oct. 8-10, 1990.

[Tsur] Tsur S., "Applications of Deductive Database Systems," *Proc. IEEE COMCON Spring '90 Conference*, San Francisco, Feb 26-March 2, 1990.

[Ull] Ullman, J.D., *Database and Knowledge-Based Systems, Vols I and II*, Computer Science Press, Rockville, Md., 1989.

[vEKo] van Emden, M.H., Kowalski, R., "The semantics of Predicate Logic as a Programming Language", *JACM 23*, 4, pp. 733–742, 1976.

[VG1] Van Gelder, A., "Negation as failure using tight derivations for general logic programs," *Proc. IEEE Symp. on Logic Programming*, pp. 127-139, 1986.

[VG2] Van Gelder, A., "The Alternating Fixpoint of Logic Programs with Negation", *Proc. 8th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, pp. 1-10, 1989.

[VRS] Van Gelder, A., Ross, K., Schlipf, J.S., "Unfounded Sets and Well-Founded Semantics for General Logic Programs", *Proc. 7th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, pp. 221-230, 1988.

[Zani] Zaniolo, C. "Object Identity and Inheritance in Deductive Databases: an Evolutionary Approach," *Proc. 1st Int. Conf. on Deductive and O-O Databases*, Dec. 4-6, 1989, Kyoto, Japan.