Design and Implementation of Logical Database Languages

Jeffrey D. Ullman

Computer Science Department
Stanford University
Stanford, CA 94305-2140

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

Approved for public release; distribution unlimited.

Studies were made of the prototype GLUE/NAIL system which is a deductive database system, techniques for optimizing constraint maintenance in a distributed environment, approaches to nonmonotic reasoning in databases, efficient main-memory algorithms for essential database operations, especially join, multway join and transitive closure, the problem of maintaining an instantiated view of data, magic-sets implementation techniques, theory of logic programs, and object-oriented versus deductive database approaches.
1. ARO proposal number: 28354-MA
2. FOR PERIOD: 7/1/91–6/30/94
3. TITLE: Design and Implementation of Logical Database Languages
4. GRANT NUMBER: DAAL03–91–G–0177
5. INSTITUTION: Stanford University
6. AUTHOR: Jeffrey D. Ullman
7. MANUSCRIPTS PUBLISHED:


The NAIL System

Early in the grant period, we completed the prototype GLUE/NAIL system, which is a deductive database system. Geoff Phipps, who wrote his thesis, Phipps [1992], under the grant, completed the implementation of a number of optimizations for the GLUE language. He developed a suite of benchmarks, including GLUE code written by himself, Ashish Gupta, and some undergraduates, and has measurements of performance improvements for each of these contained in the thesis.

The fundamental paper on the system architecture was published: Derr, Morishita, and Phipps [1993].

Also, Tiwari and Gupta [1993] describes an early application of the GLUE/NAIL system in a construction engineering application.

Constraint Management

Ashish Gupta is completing his thesis on techniques for optimizing constraint maintenance in a distributed environment. One important goal is to determine that a constraint remains unviolated after an update to the local database, without having to look at any remote or inaccessible data. There are some surprising opportunities to do so. For example, sometimes when we insert a tuple \( t \) we can argue that if \( t \) participates in a constraint violation, then there is another local tuple \( t' \) that also participates in a violation. Since we assume no violations before the insertion of \( t \), we know that \( t \) does not cause a violation, and we need not look remotely.

Gupta and Widom [1993] gives a general framework for telling whether we can be assured of no constraint violation without looking at any remote data, when an update is performed at a given site.

Gupta and Ullman [1992] specialize this question to conjunctive queries with a single local subgoal and develop an efficient solution to the question. (Conjunctive queries are expressions that are the logical AND of subgoals; each subgoal is in effect a requirement that a tuple of a certain form be in a particular relation.)

Levy and Sagiv [1993] examine the problem of determining whether a "query is independent of an update." The question is central to constraint management as well as other forms of active elements in databases such as the instantiated views discussed below. They give tests for containment of generalized conjunctive queries that have some negated subgoals.

Gupta, Sagiv, Ullman, and Widom [1994] look at "complete tests" for determining whether a constraint holds by looking at only a limited amount of information. Only when the complete test fails do we have to make a second test, looking at both local and remote data. The two most interesting cases are when we are allowed to look only at constraints
and an update, and when we are also allowed to look at local data, as above.

For the update-only case, we show that classical results on containment of logic programs carry over and in some cases give algorithms of acceptable efficiency (i.e., they are exponential only in the length of constraint expressions, not the size of the database).

For the local-and-update-only case, we have results for conjunctive queries with one local subgoal. When there are no subgoals involving arithmetic, we can find the complete test in time that is polynomial in both the constraint size and the database size. We have made some progress on conjunctive queries with arithmetic, and in some cases can make the complete test in time that is exponential in the constraint size but linear (or less if there are the right indexes) in the size of the data.

Nonmonotonic Reasoning

Alberto Torres has been working on approaches to nonmonotonic reasoning in databases, which is essentially the problem of finding the most appropriate model for a collection of logical rules that are satisfied by more than one minimal model. His completed thesis (Torres [1994]) gives an elegant 3-dimensional view of approaches to defining appropriate models. One dimension represents whether we are “skeptical” or “credulous,” i.e., whether we favor believing facts or rejecting them if they are not well substantiated. A second dimension has to do with subtle mechanics of defining models, but is roughly the difference between the two most important approaches: well-founded and stable models. The third dimension is the matter of “linearity”: whether a model is constructed by stages or is constructed all at once.

Torres shows that all the stable-like approaches share certain anomalies, such as models changing in response to the addition of irrelevant facts, and that all their corresponding well-founded approaches cure these anomalies. He also shows that for the well-founded semantics itself, which has always been defined in a “linear” way, there is an equivalent “all at once” definition. These results tie together a number of competing proposals that have appeared in the recent literature.

Earlier publications of parts of this work appear in Torres [1992, 1993a–d]. A survey of work in the area was written: Ullman [1994].

Main-Memory Join Algorithms

Hakan Jakobsson completed his thesis, Jakobsson [1993], on efficient main-memory algorithms for essential database operations, especially join, multiway join, and transitive closure. Jakobsson [1992a, c] shows how joins of more than two relations can be speeded by partitioning relations into parts and joining parts of relations in different orders. He then gives an algorithm that performs at least as well as any strategy that works by partitioning relations.

Incremental View Update

Gupta, Mumick, and Subrahmanian [1993] and Gupta, Katiyar, and Mumick [1992] look at the problem of maintaining an instantiated view of data. See also Gupta [1993]. In this paper they use counts of “proofs” to aid in finding incremental view updates in response to updates to the underlying database.
Jakobsson [1992b] applies the techniques of his papers mentioned above to the view update problem.

Also, Gupta and Blakeley [1993] patches up an error in an earlier algorithm by Tompa and Blakeley for maintaining instantiated views.

**Magic-Sets Implementation Techniques**

Gupta and Mumick [1992] shows an interesting result about "magic sets," which is a key optimization technique used in the NAIL system for handling recursive queries. It was known that the technique applies to nonrecursive queries as well. However, sometimes the magic-sets transformation turns nonrecursive logic into recursive logic, which is a problem since recursive rules at the least require a termination test that can be avoided for nonrecursive rules.

They show is that a simple additional transformation takes the result of magic sets applied to nonrecursive rules and produce an equivalent set of rules that is guaranteed not to be recursive. It now looks like magic-sets is the preferred technique for almost all nonrecursive as well as recursive queries.

**Theory of Logic Programs**

In Chaudhuri and Vardi [1992] there is an algorithm to decide whether the result of a "logic program" (= set of recursive, logical rules) is contained in the result of a single logical rule, that is, whether a recursion is equivalent to some first-order logical formula.

Ullman [1991b] is a survey of optimization techniques, such as magic sets, for improving the running time of logical queries. Ullman [1991c] discusses some techniques for parallelizing logic programs that follow from the earlier body of knowledge developed for formal languages.

**Object-Oriented Versus Deductive Database Approaches**

Ullman [1991a] shows that there are certain incompatibilities between the deductive (logical) and object-oriented paradigms. In particular, you cannot have a deductive database system that takes object identity seriously, or that permits dynamic type creation. The conclusion is that the community trying to combine these paradigms (a worthwhile endeavor), need to back off from the more extreme visions of what "object-oriented" means.