

Prolog and deductive databases

M H Williams, G Chen, D Ferbrache, P Massey, S Salvini,
H Taylor and K F Wong

The logic programming language Prolog has been shown to be a very suitable language for implementing database concepts. However, current Prolog implementations are limited, and the database examples used have all consisted of relatively small sets of clauses. The reason for this is that existing Prolog implementations do not scale up to handle large databases. This paper describes a project whose aims are to develop the links between logic programming and databases. The prime aim is to develop a Prolog system which is capable of handling large sets of clauses. The second aim is to implement a deductive database management system in Prolog, while a third is to evaluate the effectiveness of parallel logic languages for implementing database applications.

Keywords: Prolog, Parlog, logic programming, deductive databases

Within the field of database systems several different approaches have been used, but of these the relational database has emerged as the most popular. The clean, uniform approach of the relational data model, and the power and simplicity of relational query languages have been the main reasons for the success of relational databases. These, in their turn, owe much to logic, as the relational calculus is a restricted form of first order predicate calculus.

With the growth of interest in knowledge processing, and in expert systems in particular, a need has been developing for more powerful database systems - systems which can store and retrieve knowledge; such systems are generally referred to as 'knowledge base management' systems, or 'expert database' systems. One of the formalisms used for representing knowledge is first order predicate calculus.

In view of the role of logic in underpinning relational databases and in describing knowledge, a natural successor to relational database systems, in the form of logic or deductive databases, has been proposed. Such systems are natural extensions of relational databases based on a more general form of first order logic. One particular form, known as a definite deductive database, has been studied in some detail¹. This form of database is

based on a firm theoretical foundation, providing the facility to handle rules as well as facts, and offers considerable promise as a successor to relational databases. The main focus of research at this stage is the development of efficient and safe query processing strategies.

Various approaches to achieving such systems are currently under investigation by different groups. This paper describes work being directed towards a two-level strategy which is being implemented as part of a programme of research funded by the Alvey initiative.

PROLOG AS AN INTERMEDIATE LANGUAGE

The suitability of Prolog for implementing various aspects of databases has been known for some while². Not only is it ideal for analysing and translating queries into internal form, but it is also a natural candidate as the internal form into which queries are translated and then executed³. For this reason it was selected as the intermediate language for this project.

However, although Prolog has proved its suitability for realizing databases on a smaller scale, current Prolog implementations suffer from the drawback that they do not scale up. In order to assess the performance of current systems in this regard, a set of benchmarks has been drawn up which are specifically geared to database applications⁴. Initially, a fairly complex set of tests was devised, but in the end this had to be reduced to a more limited set because of the poor responses recorded.

When the benchmarks were applied to different Prolog systems, their shortcomings were clearly revealed. In a study involving three Prolog systems, C-Prolog⁵, Prolog-X⁶ and NIP⁷ on a 4Mbyte Sun3 system the following conclusions were reached: .sp .75

- Given a large set of simple ground facts, each with three arguments, all three systems were unable to cope when the number of clauses reached some limit between 30K and 70K. In each case, the total space required by the clauses and the associated symbol table exceeded the available space at this level.
- The time taken by each system to compile 30K to 70K clauses from source into their respective internal forms is considerable.
- The time taken by each system to load 30K to 70K compiled clauses from a file on disc into main memory is significant.

Computer Science Department, Heriot-Watt University, 79
Grassmarket, Edinburgh EH1 2HJ, UK

Since a real database might be expected to handle millions of clauses, the limitations of current Prolog systems are unrealistic. To handle a million clauses by simply extending the main memory would be impractical, since the amount of memory required would be excessive.

If a database is to be accessible to more than one user, and may be updated by any of them, then it cannot be retained in the main memory. In this case the time taken to load it into the main memory for each user becomes an intolerable overhead. For these two reasons one needs to retain the database on disc, and load only those clauses which satisfy the query or subquery. Because compilation time is high, the clauses need to be stored in compiled form, and a facility to perform incremental updates is essential.

OVERALL ARCHITECTURE

There are various possible ways in which the power of logic programming languages and the storage capabilities of database systems may be brought together. The simplest of these is the idea of a loosely coupled system in which a Prolog compiler or interpreter is linked to a conventional database system. This approach has been followed by various people, e.g. Cerie et al.⁸, but generally is very inefficient and clumsy.

A more efficient alternative is to connect a Prolog translator to a database system using a tight coupling. In this case one or both of the two components will be modified to effect a direct linkage. This approach has been followed in the EDUCE system developed at ECRC⁹. A survey of different approaches is given in Reference 10.

Although the idea of coupling two existing systems together has advantages in terms of rapid development, this is not a long term solution to the problem of deductive databases. First, the database system is not geared to storing and retrieving clauses; second, the Prolog system will still be limited in terms of the size of the set of clauses it can manipulate; and third, the interface to the system is the language Prolog itself, which is not suitable as a query language for inexperienced users.

The strategy adopted in the current project is a two level approach in which different aspects of the problem are tackled at different levels. The project has three prime objectives:

- 1 To develop a Prolog system based on special purpose hardware which is capable of handling large data sets efficiently. The system being developed, referred to as the Prolog Database Machine (PDBM), extends conventional Prolog with efficient clause retrieval from large data sets stored on disc.
- 2 To implement a Deductive Database Management System (DDBMS) in Prolog which will provide a simple query language by which a user can access clauses in the database, and facilities to assist the user in the management of his data.
- 3 To evaluate the effectiveness of parallel logic programming languages for implementing deductive databases. To this end, a compiler for the language Parlog will be linked to the special purpose hardware outlined in (1) and used to implement a database application.

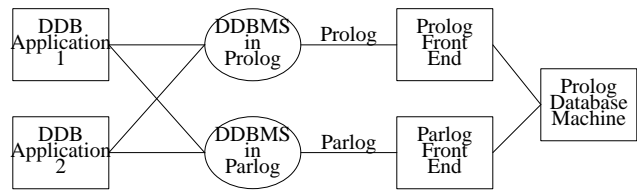


Figure 1. Overall system structure

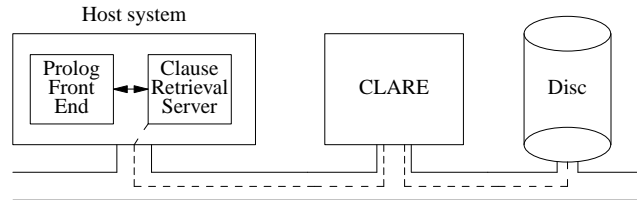


Figure 2. Prolog Database Machine (PDBM) architecture

To demonstrate the DDBMS executing on the PDBM various applications are being considered, and two in particular are being implemented. Thus, the overall structure of the system is as illustrated in Figure 1. The version of Prolog chosen for this purpose is Prolog-X. It was originally developed by Clocksin as an abstract machine code compiler and emulator with a view to microcoding the instruction set on a suitable host⁶. The PDBM is based on a C version of the Prolog-X system, and this is currently running on Sun3 systems.

THE PDBM

The Prolog database machine is basically a Prolog system which has been extended to handle large sets of clauses. The version of Prolog on which it is based has a module facility which enables clauses to be grouped together into modules. If a module is small in size, the whole module can be loaded into the main memory and used directly; if it is large, it will be disc resident, and clauses will be taken into the main memory as and when they are needed.

Thus, the Prolog system has been extended as detailed below. The front end of the system executes Prolog programs. Provided that the modules it requires are in the main memory, the front end operates as a normal Prolog system. If it requires access to a disc resident module, it invokes the Clause Retrieval Server (CRS). This is a separate sub-system which is responsible for communicating with and controlling some special purpose hardware known as the Clause Retrieval Engine (CLARE), an associative retrieval engine for clauses. The CRS is responsible for setting up and sending the appropriate parameters to CLARE, and receiving the results returned. The architecture of the system is illustrated in Figure 2.

As depicted in Figure 2, CLARE is a special purpose engine which links to the bus connecting the processor and its disc(s). When a clause or a set of clauses is to be retrieved from the disc, the appropriate set of data is passed to CLARE. It then sifts through the clauses as fast as they are delivered by the disc and passes any matching

clause or clauses to the processor.

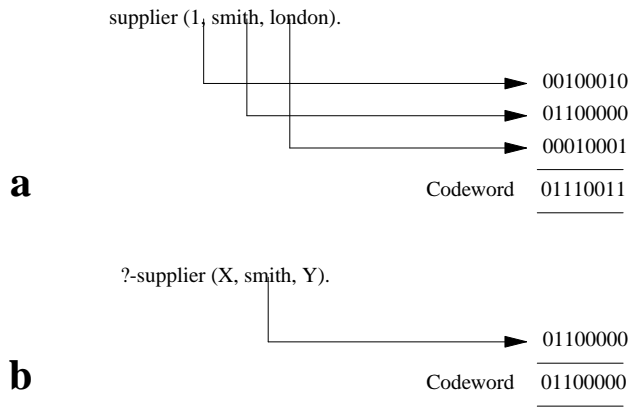


Figure 3. Generation of a simple codeword for (a) a fact, (b) a query

The clause retrieval engine itself is based on a two stage filtering process. The first stage makes use of superimposed codeword indexing". This is a form of hashing, in which a special hashing function is applied to each constant argument of a clause and the resulting bit strings are combined together by a logical 'or' operation to form a codeword (see Figure 3). When a query is issued, it is subjected to the same process to produce a corresponding codeword. The query codeword is then compared with the codeword for each clause in turn following a special matching procedure in order to determine possible hits.

Any possible hits obtained by the first phase are passed to the second phase of the clause retrieval engine. This is a partial unification filter, which performs most of the checks carried out by a software unification routine. It was decided not to extend this to full unification checking, since the part that has been omitted is a time consuming process which accounts for only a very small proportion of queries. Realization of full checking would impede on-the-fly clause retrieval and complicate the hardware considerably. The clauses which satisfy this second phase of the engine are passed back to the CRS.

On receiving clauses from CLARE, the CRS returns these results to the Prolog front end where they are subjected to the normal unification process by software. An additional level of clause management is incorporated into the CRS to make the system simple to use and nearly transparent to the user.

Since Prolog-X supports the incremental compilation of clauses, it is possible to store clauses in compiled form on disc. This avoids the overhead of compiling and decompiling clauses whenever they are transferred into and out of main memory.

THE DDBMS

The deductive database management system fits hand in glove with the Prolog Database Machine. On the one hand the DDBMS needs the PDBM in order to provide adequate performance for database applications; on the other hand, the DDBMS provides a useful yardstick by which the performance of the PDBM can be judged

against other Prolog systems.

The DDBMS is a natural extension of a relational database system which handles both facts and rules. A fundamental decision in the design of the DDBMS is the form of clauses to be supported by the system. At one extreme one has full clausal form, in which each clause in the database may have the form

$$P_1 V \dots V P_n \leftarrow Q_1 \wedge \dots \wedge Q_m \quad n, m \geq 0.$$

Full clausal form has the expressive power of first order predicate calculus, but is impossible to implement efficiently and leads to problems with regard to satisfying the Closed World Assumption for negative information. At the other extreme, one has Horn clauses, which have the form

$$P \leftarrow Q_1 \wedge \dots \wedge Q_m \quad m \geq 0.$$

Horn clause resolution can be implemented efficiently, and for Horn clauses the Closed World Assumption is directly equivalent to negation as failure. However, the expressive power of Horn clauses is limited.

Since work is progressing on finding suitable intermediate representations between Horn clause and full clausal form^{1,2} it was decided to proceed with the design of a query language which can handle full clausal form. Restrictions can then be placed on the form of clauses permitted by any particular implementation depending on the internal representation used.

A DDBMS based on this assumption is currently under development. In particular, it will provide the following facilities:

- The query language on which it is based is an extension of SQL, which caters for clauses as well as tuples. In order to achieve this, instead of treating the tuple as the basic unit of the database (as is the case for relational databases), the basic unit is taken to be the clause. With this extension the tuple variable in SQL is extended to a clause variable which represents each clause in the database. The resulting query language is considerably simpler to use than the programming language Prolog, and enables a user with relatively little training to query a database containing both facts and rules. In addition to facilities to query the database, the system also provides facilities for the user to define and manipulate data and rules. These too are simple to use.
- The query evaluation sub-system must support different external models of the data. Thus, in order to enable different knowledge-based system applications to store their knowledge in the deductive database, the query evaluation system must provide the necessary transformations between the external and internal formats. By introducing the notion of a clause variable into the language, the user can access and manipulate the knowledge stored in the deductive database in a unified manner, and at the same time cope with the knowledge from different models. It can also deal with the situation where a database user wishes to treat rules as representing implicit data, and to use them to deduce the data implied by the database, whereas a database administrator may wish to treat them as a set of first order sentences from which some structural

- properties or integrity constraints might be derived.
- The initial implementation of the query evaluation sub-system is based on the use of Horn clauses as the means of internally representing knowledge. However, a more general internal representation is currently being considered, and it is hoped that this will be incorporated into the final system. This does mean that, initially, all clauses which are stored as objects in the database must be converted to Horn clause form.
- A facility to handle general integrity constraints is being provided whereby the user may specify and enforce conditions imposed upon facts as well as rules. This facility can be used by other knowledge-based systems to specify and check the integrity of the knowledge base. Integrity constraints are also expressed as clauses, but are not restricted to Horn clauses. Thus they may have the general form

$$P_1 \vee \dots \vee P_n \leftarrow Q_1 \wedge \dots \wedge Q_m$$

$$\text{or } \leftarrow Q_1 \wedge \dots \wedge Q_m \wedge \neg P_1 \wedge \dots \wedge \neg P_n$$

In addition to normal static integrity constraints, provision has also been made for transition constraints¹³.

APPLICATIONS

In order to demonstrate the usefulness of a database system which can store, retrieve and manipulate rules as well as facts, two specific applications are being developed. The first involves adapting several different expert system shells so that for each shell, instead of storing the knowledge in its own internal knowledge base, it stores it in the deductive database. Each shell will then be able to fetch and store rules via the query language.

The rationale behind this application is that, as the number of expert systems in an organization grows, the amount of common knowledge used by the different systems will increase. For example, in a commercial environment, common business practice may underlie a number of different expert systems; in an engineering firm, knowledge of good design practice, safety rules and basic engineering principles may be common to a number of systems, and so on. As the number of systems proliferates, and more and more knowledge is duplicated, there will be growing pressure to structure the knowledge into local and global knowledge, and to store knowledge rules in such a way that some sets of rules may be shared by two or more expert systems, whereas others are private to a specific expert system.

The second application involves the storage of a database of medical records, together with various types of rules. This is then used by an automatic learning system to extract and formulate knowledge from the raw data and store this knowledge back in the same database.

PARLOG

The parallel logic programming language Parlog is executed by a committed choice resolution strategy over guarded Horn clauses that makes it unsuitable for obtaining more than one solution to a database query. Accordingly, Parlog has been defined to have two Prolog-

like extensions to handle all-solution searches over a Horn clause database stored separately from the executed Parlog program¹⁴.

This project is developing the first serious implementation of these all-solutions constructors, along with a clause update facility that will be linked via the Clause Retrieval Server to the PDBM. A new sequential implementation of Parlog based upon a kernel guarded Horn clause emulator, will incorporate these database search engines into a full Parlog system with a Prolog-style interface. On top of this implementation a DDBMS will combine the distinctive systems programming virtues of Parlog with this facility for efficient database retrievals and updates, enabling it to be compared with a Prolog DDBMS and enabling their relative abilities for handling database applications to be assessed.

CONCLUSION

The project described in this paper has various objectives. First, it aims to produce a Prolog system which can deal with large sets of clauses efficiently: this is needed urgently as a vehicle for prototyping database applications, although many other applications will require large clause sets in the future too. It is too early to provide performance figures for the final system, but some preliminary results for the PDBM obtained using an ICL 3930 are encouraging. It must be stressed that the PDBM is not a database management system, but a Prolog system extended to deal with large sets of clauses and coupled to special hardware to enable this to be done efficiently.

The second objective of the project is to implement a deductive database management system with a suitably powerful yet simple query language. Although the initial implementation is based on Horn clauses, a more general internal representation is being studied. The resulting system will be similar to a relational database management system, except that it will have the capability of dealing with rules as well as facts. Although Prolog is being used as the language for developing the prototype, it could clearly be re-engineered in a language such as C if it proves successful.

To demonstrate the usefulness of such a system, two particular applications are being developed:

- 1 The connection of several different expert systems to the deductive database management system in order to store the knowledge of the expert systems in a central knowledge base.
- 2 The use of the deductive database management system by an automatic learning program to store both raw data and processed rules.

Finally, the potential for implementing database systems in the parallel logic language Parlog will be explored.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the support received from ICL, and from the Science and Engineering Research Council under the Alvey initiative, who are currently funding the project.

REFERENCES

- 1 **Gallaire, H, Minker, J and Nicolas, J-M**, "Logic and databases: a deductive approach" *ACM Comput. Surveys* Vol 16 No 2 (June 1984) pp 153-185
- 2 **Kowalski, R A** *Logic as a Database Language* Internal Report, Department of Computer Science, Imperial College, London, UK (1981)
- 3 **Neves, J C, Anderson S O and Williams, M H** "A Prolog implementation of query-by-example" *Proc. 7th Int. Comput. Symp.* B.G. Teubner, Stuttgart. FRG (1983) pp 318-332
- 4 **Williams, M H, Massey, P A, Crammond, J A** *Benchmarking Prolog for Database Applications* Internal Report, Computer Science Department, Heriot-Watt University, Edinburgh, UK (1987)
- 5 **Warren, D, Bowen, D, Byrd, L and Pereira, L C** *Prolog User's Manual* 1.4d.edai (September 1985)
- 6 **Clocksink, W F** "Design and simulation of a sequential Prolog machine" *New Generat. Comput.* Vol 3 (1985) pp 101-120
- 7 **Bowen, D L, Byrd, L, Chung, P W H, Pereira, F C N, Pereira, L M, Rae, R and Warren, D H D** *Edinburgh Prolog (The New Implementation) User's Manual version 1.5*, AIAI, Edinburgh University, (1st June, 1987)
- 8 **Ceri, S, Gottlob, G, and Wiederhold, G** "Interfacing relational databases and Prolog efficiently" *Proc. 1st Int. Conf. on Expert Database Systems* Charleston, USA (1986)
- 9 **Bocca, J** "EDUCE a marriage of convenience: Prolog and a relational DBMS" *Proc. 3rd Symp. of Logic Programming* Salt Lake City, Utah, USA (1983)
- 10 **Gallaire, H** "Logical data bases vs deductive data bases" *Proc. Logic Programming Workshop* Universidade Nova de Lisboa, Portugal (1983) pp 608-622
- 11 **Ramamohanarao, K, Shepherd, J** "A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases" *Proc. 3rd Int. Conf. on Logic Programming* London, UK (1986) pp 569-576
- 12 **Chisholm, P, Chen, G, Ferbrache, D, Thanisch, P and Williams, M H** "Coping with indefinite and negative data in deductive databases: a survey" *Data and Knowledge Engineering* (to appear)
- 13 **Gaebler, A G and Williams, M H** *An Integrated System of Static and Transition Constraints for a Deductive Database* Internal Report, Computer Science Department, Heriot-Watt University, Edinburgh, UK (1987)
- 14 **Gregory, S** *Parallel Logic Programming in Parlog - The language and its Implementation* International Series in Logic Programming, Addison-Wesley, London, UK (February 1987)