COMPUTATIONAL INTELLIGENCE, III
N. Cercone & F. Gardin (Editors), G. Valle (Co-Editor)
© Elsevier Science Publishers B.V. (North-Holland), 1991

Integrating a deductive database system with a Warren Abstract Machine

Stefan Böttcher
IBM Deutschland GmbH
Scientific Center
Institute for Knowledge Based Systems
P.O.Box 80 08 80
D – 7000 Stuttgart 80
West Germany *

Abstract

This paper describes a special purpose inference engine for deductive databases and how it is integrated with a Warren Abstract Machine within the PROTOS-L system. Furthermore, we outline some typical cases where this special purpose inference engine is superior to the standard Prolog evaluation strategy based on backtracking, and we compare the performance of the special purpose inference engine to that of the standard Prolog evaluation strategy.

1 Introduction

1.1 An overview of PROTOS-L

PROTOS-L is a logic programming language extended by the following features. PROTOS-L embeds a module concept similar to that of Modula-2 [Wirth, 1983], provides read access to external databases, allows to program deductive databases [Böttcher, 1990a], and combines order-sorted types with polymorphism [Beierle and Böttcher, 1989], [Beierle, 1989]. PROTOS-L has been implemented at IBM Stuttgart on an IBM-RT 6150 workstation.

The PROTOS-L system contains a compiler, an abstract machine [Semle, 1989] (which is an extension of the Warren Abstract Machine [Warren, 1983] by subtypes, polymorphism and database access), a run time code retrieval system [Böttcher and Beierle, 1989], [Garidis and Böttcher, 1990] and a deductive database system. The PROTOS-L deductive database system supports an efficient evaluation of function free logic programs[1] which are equivalent to DATALOG [Bancilhon and Ramakrishnan, 1986].

PROTOS-L offers two evaluation strategies for logic programs: backtracking as in Prolog and set-oriented retrieval as in deductive databases. For this purpose, the module concept of PROTOS-L supports the following two kinds of module bodies (the PROTOS-L programmer can choose between both kinds of bodies in order to implement an interface): program bodies (which are evaluated by tuple unification and backtracking) and database bodies (which are evaluated by set-oriented proof techniques). A PROTOS-L database body contains only function free rules which additionally have the property that after every bottom-up (forward chaining) application of such a rule every variable occurring in the head is bound.

*The research reported here has been carried out within the international EUREKA project PROTOS (EU56): Prolog Tools for Building Expert Systems. Project partners are BIM, IBM Stuttgart, Sandoz AG, Schweizerische Bankgesellschaft, University of Dortmund, and University of Oldenburg.

[1] We also use the term *deductive database* as a synonym for a function free logic program accessing a relational database.

1.2 The language for the PROTOS-L deductive database inference engine

PROTOS-L supports the programming of deductive databases because the PROTOS-L database bodies may contain any function free logic program which is equivalent to DATALOG. We give a short overview of DATALOG because the expressive power of the language and its typical applications determine the requirements for the inference engine.

DATALOG has become the most commonly used programming language for deductive databases, as Prolog is the best known logic programming language. As logic programs, a DATALOG program can be considered as a collection of facts and horn rules, i.e. the rules have one unnegated literal. However, different from logic programs DATALOG programs contain only function free facts and rules, i.e. they do not contain function symbols. Further, DATALOG programs can access relations of a relational database in order to get ground facts for a certain predicate, i.e. ground facts are not listed in the program (as e.g. in Prolog), but are retrieved from a database relation. Typical DATALOG programs access database relations which contain very many ground facts (some thousand) compared to Prolog programs. Additionally, the sequence of facts and rules is not considered to be relevant in a DATALOG program, but it is relevant for Prolog.

There are many applications for DATALOG programs because relational databases are a widely used tool in order to share large data collections.

To summarize: The difference of DATALOG programs compared to logic programs is as follows. DATALOG programs are used for applications with function free rules which typically consider a large number of ground facts stored in database relations.

1.3 Inference engines of the PROTOS-L system

The PROTOS-L inference system consists of two inference engines which are implemented as abstract machines. The upper inference engine (called the PROTOS Abstract Machine (PAM)) is an extension of the Warren Abstract Machine by types and polymorphism. This upper inference engine uses backtracking. The lower inference engine (called deductive database inference engine (DDBIE)) is a special purpose inference engine for function free logic programs. Its set-oriented proof technique and some performance results are described in section 3. The results of section 3.3 show that this inference engine is adapted for proofs which contain large sets of ground facts.

Alltogether, PROTOS-L provides backtracking on top of set-oriented proofs. Note that it is the decision of the PROTOS-L programmer, in which cases he assumes that solutions are easy to find and therefore prefers backtracking, and in which cases he prefers set-oriented retrieval of facts because he assumes that a large search space has to be searched in order to find a solution. Whenever the PROTOS-L programmer prefers tuple-oriented unification with backtracking he programs his rules in program bodies (the code of which is evaluated by the upper inference engine). Otherwise, if he prefers set-oriented retrieval he programs his rules in database bodies (the code of which is evaluated by the lower inference engine).

Note further, that the syntax for rules in program bodies is identical to the syntax for rules in database bodies. Therefore, the PROTOS-L programmer has to learn only one single language.

1.4 A motivating example

In this section we demonstrate the advantages of program bodies and the advantages of database bodies and outline how the programmer can choose between both in order to determine the evaluation strategy of his program.

The following example is taken from the domain of production planning systems. It is a modified version of a larger production planning example given in [Böttcher, 1990c] which describes the reimplementation of a scheduling algorithm for a chemical production planning system. A first version of this scheduling algorithm has been implemented at Sandoz AG in Basel [Slahor *et al.*, 1990], [Sauer, 1990].

The production planning algorithm has to obey the constraint that every product can only be produced at some machines. Furthermore, the planning algorithms has to care about whether or not a machine is available at the time it shall be used for production.

Which product can be produced on which machine is stored in a specific relation `can_be_produced_on`. The relation `available` computes whether a machine is available at a given time interval. The predicate `producable` checks whether a given product is producable in a given time interval and returns the machine on which a product is producable. The predicate `producable` can be implemented by the predicates `can_be_produced_on` and `available` as follows.

```
rel producable :  string x   int x int x    ?string .
%                 product     time interval  on machine

producable( Product, From, Until, Machine ) <-
      can_be_produced_on( Product, Machine ) &
      available( Machine, From, Until ) .
```

PROTOS-L offers the programmer the choice between two alternative evaluation strategies for the predicate `producable`. If the programmer assumes that it is difficult to find an available machine on which the product can be produced, then he may prefer to evaluate the predicate `producable` in a set-oriented way. In this case he will declare the module body in which the predicate `producable` is implemented, say `producable_orders`, to be a database body because database bodies are evaluated by the PROTOS-L deductive database system, i.e. the join of `can_be_produced_on` and `available` is evaluated in a set-oriented way.

However, if the programmer assumes that it is easy to find an available machine on which the product can be produced, then he may prefer to evaluate the predicate `producable` by backtracking. In this case, he will declare the module body in which the predicate `producable` is implemented to be a program body because program bodies are evaluated by the PROTOS Abstract Machine, i.e. elements of `can_be_produced_on` are taken one by one and the evaluation of the predicate `producable` can be stopped when no more solutions are needed.

The two system implementation alternatives are shown in figure 1. In the alternative shown in the left half of figure 1, the module body `producable_orders`, in which the predicate `producable` is implemented, is a database body, in the alternative shown in the right half of figure 1, the module body is a program body. The example shown in figure 1 assumes that the relations `can_be_produced_on` and `available` are implemented in database bodies computing the `production_data`.

2 The PROTOS-L deductive database system

2.1 Advantages of the PROTOS-L deductive database system

The following outlines why the PROTOS-L system contains two inference engines, i.e. why the Prolog evaluation strategy in general, and thereby the PROTOS Abstract Machine in particular, is not sufficient for query evaluation in deductive databases which are programmed in database bodies.

A first reason is the computation of large joins. Consider the rule implementing the predicate `producable` which is given in section 1.4. In production planning systems the relation `can_be_produced_on` may contain about 5000 ground facts depending on the factory for which it is planned. The size of the relation

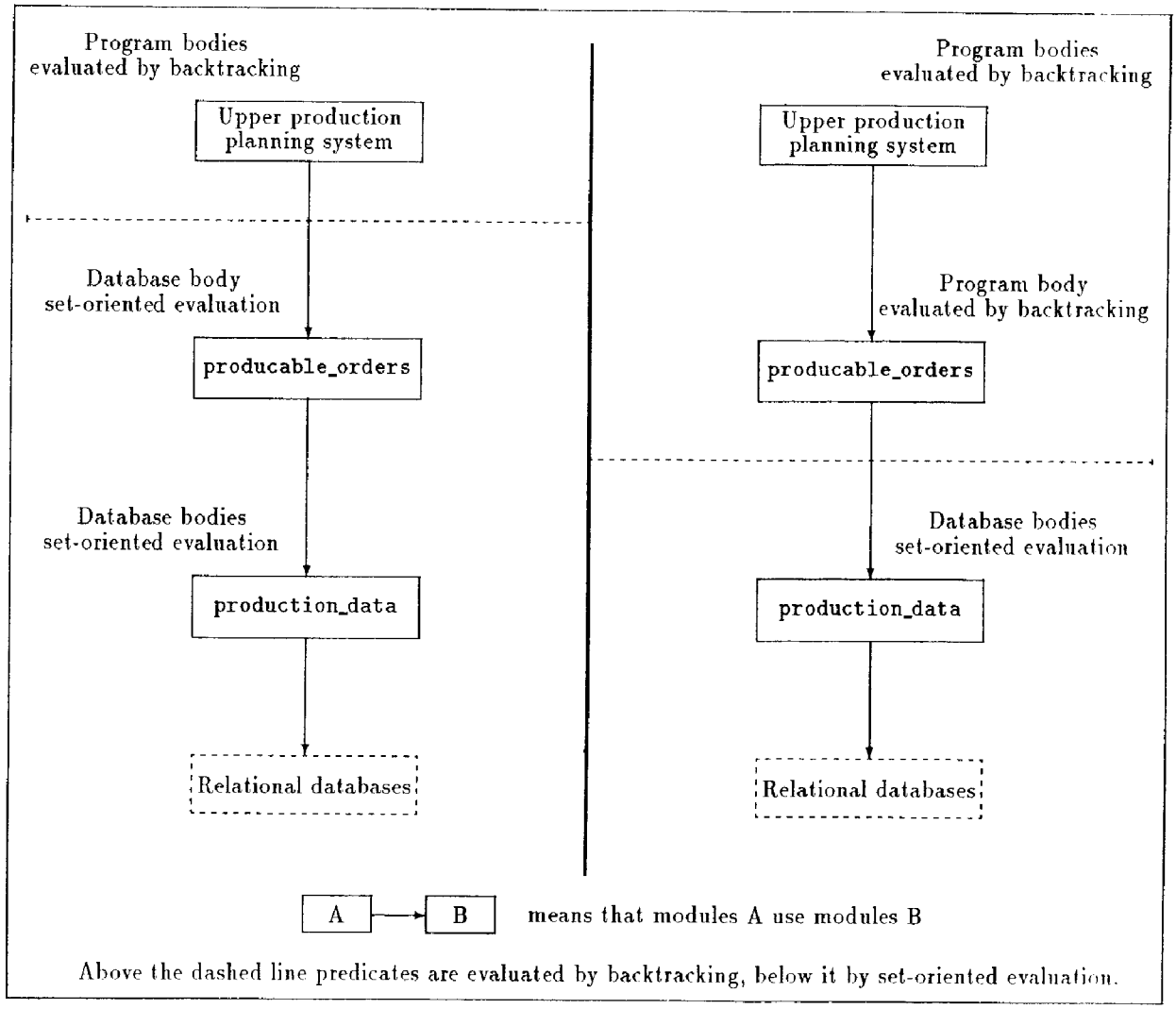

Figure 1: Two implemetation alternatives for the predicate `producable` both using backtracking on top of set-oriented retrieval from relational databases.

`available` depends on the size of the time interval for which the production is planned. Assume this relation contains 2000 facts. In the worst case, the Prolog strategy proves 5000 goals `can_be_produced_on` and 5000×2000 goals `available` in order to compute the answers to the predicate `producable`. Of course, efficient join algorithms implemented in the underlying SQL/RT database system are much faster.

A second reason not to use the Prolog evaluation strategy is the requirement to avoid duplicate computations. This idea is known as *lemma generation* in the field of automated theorem proving. The idea of lemma generation is that all intermediate results are stored for further usage and are used instead of recomputing the answers to a goal. This can not be done by a Warren Abstract Machine because it does not contain any stack for intermediate results. For the same reason, an extension of the Warren Abstract Machine has been proposed recently [Warren, 1989]. This extension has the property of storing intermediate results in common with the DDBIE of the PROTOS-L system.

Furthermore, during the evaluation of left recursive programs, the Prolog evaluation strategy may run into an infinite loop, although there exists a proof for a goal. This is avoided by the DDBIE of the PROTOS-L system, because it uses a bottom-up deduction strategy and because all rules in database bodies are free of function symbols.

2.2 The process model of the DDBIE

This section summarizes the inference technique used in the PROTOS-L deductive database inference engine. A comprehensive description of the algorithms is given in [Meyer, 1989] and [Böttcher, 1990b].

A PROTOS-L deductive database contains only function free rules which additionally have the prop-

erty that after every bottom-up (forward chaining) application of such a rule every variable occurring in the head is bound. The PROTOS-L DDBIE combines bottom-up (forward) deduction with a goal-oriented deduction as used in the Warren Abstract Machine. Because the PROTOS-L DDBIE uses this goal-oriented deduction it is superior to naive or semi-naive inference engines [Bancilhon and Ramakrishnan, 1986].

The DDBIE combines the magic set approach [Bancilhon and Ramakrishnan, 1986] with the QoSaQ approach [Vieille, 1988] and with ideas outlined in [Hulin, 1989]. The basic idea of the PROTOS-L deductive database inference engine is the process model. Every predicate occurring in a PROTOS-L database body is implemented by its own process. For didactic purposes, it is easier to think of independent processes, although the whole computation is implemented as a single PROTOS-L process. All rules of a predicate together with all goals of the rule are considered to be a part of the process. A global scheduler coordinates the processes.

Each process can perform the following activities:

- it can submit an answer,
- it can submit a subquery,
- it can signal to the global scheduler that it is idle.

Furthermore, each process has two kinds of memories:

- a memory for received goals containing also their environments,
- a memory for the computed answers.

The processes are lazy, i.e. no process computes any subquery twice. Instead, the process submits stored answers to newly received goals and it submits each newly received answer to all stored goals which can be satisfied by the answer, thereby reactivating other processes. That is how PROTOS-L supports lemma generation for every predicate occurring in a database body.

Query processing is completed when all processes signal to the global scheduler that they are idle.

The process model allows various degrees of freedom [Böttcher, 1990b]. For example, every process can decide by its own in which sequence it submits answers or subqueries. Furthermore, the global scheduler can control the evaluation strategy by assigning priorities to the processes, e.g. bottom-up evaluation is enforced by assigning higher priorities to processes at the leaves of a proof tree.

The advantages of the process model over the Prolog deduction strategy can be summarized as follows. The process model embeds a fast computation of large joins which is provided by the underlying database system. The process model uses lemma generation for every predicate. Finally, the process model avoids infinite left recursion.

We do not outline the process model in more detail here because it is described in [Böttcher, 1990b], which additionally discusses lemma generation in the process model and presents the implementation of the process model in the PROTOS-L system.

2.3 Performance results

As described for the module `producable orders` in section 1.4 several modules have been implemented in one version as a database body and in another version as a program body. Then both systems have been linked together as shown in figure 1 and the performance of both systems has been compared.

This performance comparison was done for the reimplemtation of the production planning system [Böttcher, 1990c] and for a travel information system [Böttcher, 1990a].

The queries to the production planning system are to plan a given set of production orders under some given production constraints. The computation which products could be run on which machines has been implemented in one version in a database body and in the other version in a program body. Depending on the set of constraints, the version using the database body is 2 to 4 times faster than the version using the program body.

The queries to the travel information system are:

- Compute the shortest travel time between two cities.
- Find a path with a minimum number of stopovers.
- Compute the latest departure time that is sufficient in order to reach a city at a given time.
- Find the earliest arrival time if a city can be left at a given time.

The result of the performance comparison is that the version implementing as much as possible in the database body (i.e. using the DDBIE as inference engine) is in most cases superior to the version implementing as much as possible in the PAM (i.e. using the Protos Abstract Machine). How much the first version is superior to the second depends on the size of the base relations and on the distance of the cities. For example, we get the following results in a test where the database relations of the travel information system contains about 100 facts. If the distance of the cities is 2 (i.e. the search tree has a depth of 2), then the first version is about as fast as the second (from 10% slower up to 80% faster depending on the query). However, if the distance of the cities is 4 (i.e. the search tree has a depth of 4), then the first version is between two and four times faster than the second version.

Further evaluations have generalized this performance result: The larger the relations are and the deeper the proof tree is, the more is the first version (using the database body which is evaluated by the DDBIE) superior to the second version (using the program body which is evaluated by the PAM).

3 System architecture and implementation

3.1 The integration of the DDBIE into the Warren Abstract Machine of the PROTOS-L system

The architecture of the PROTOS run time system can be summarized as follows. The PROTOS-L system consists of a compiler and a hierarchy of two inference engines, which are implemented as abstract machines (c.f. figure 2).

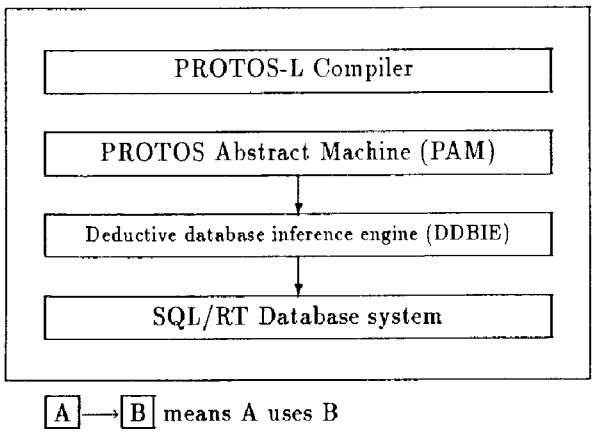

Figure 2: Architecture of the PROTOS-L system

The upper inference engine (called the PROTOS Abstract Machine (PAM)) is an extension of the Warren Abstract Machine [Warren, 1983] and uses backtracking [Semle, 1989], [Böttcher and Beierle, 1989]. The lower inference engine is the deductive database inference engine (DDBIE) described above.

As shown in section in section 1.4, the PROTOS-L system supports backtracking on top of set-oriented retrieval of facts. Therefore, set-oriented results are not computed and stored completely but instead are computed part after part as required by the upper inference engine. In order to support this incremental set-oriented computation, the DDBIE is embedded in the PAM as follows. The PAM-DDBIE interface contains besides others two functions: by one, the PAM submits a query to the DDBIE, by the other the PAM asks for further solutions for a given query goal. The queries are evaluated by communicating processes of the DDBIE as described in the previous section, however only a first part of the solution-set to the goal is computed. The procedure which asks for further solutions may trigger the computation of a further part of the solutions to the given query goal, if each tuple of a current part has already been returned to the PAM.

3.2 Implementation

Since the PAM is the upper inference machine and the only abstract machine directly used by the code of compiled PROTOS-L programs, the PAM code contains also code for the evaluation of queries to database bodies. This code is passed through the PAM-DDBIE interface to the DDBIE. The interface of PAM and DDBIE contains only the following functions.

- A set of functions for constructing at run time a graph structure which represents the PROTOS-L program coded in a database body, i.e. functions constructing predicates, rules, goals and built-in goals, arguments, variables and constants. The function calls are part of the PAM code which is produced by the compiler during the compilation of the database body.
- A function `dbquery` which is used to solve a goal when the corresponding predicate is implemented in a database body. Each call of `dbquery` passes the name of the goal predicate and the argument bindings from the PAM to the DDBIE. The argument bindings are used by the DDBIE in order to reduce the search space by the goal-oriented query submitting technique described in section 2.2.
- A function `dbfacts` which is used by the PAM in order to read the next fact to a goal submitted by dbquery. The DDBIE submits the next fact for the given goal to the PAM or informs the PAM that there is no next fact for the given goal.
- A function `dbcutfacts` is used by the PAM in order to tell the DDBIE that no more answer to a given goal is needed and that the DDBIE can do garbage collection.
- A function initializing a connect link to the underlying relational database system (which is in our case the SQL/RT database system).

These functions are used in the PAM in the following way. The functions which construct a graph representing a copy of the database body are executed once for each query to a database body. After this graph construction the argument bindings are passed by a call of `dbquery` and the DDBIE computes a result. Thereafter, the DDBIE returns a single fact for each call of `dbfacts` until there are no more facts for the given goal or the PAM called the function `dbcutfacts` in order to tell the DDBIE that no more facts of a given goal are needed.

4 Summary and conclusion

PROTOS-L provides two kinds of module bodies: program bodies and database bodies. Program bodies and database bodies may both contain function free logic programs, however they evaluate function free logic programs in a different way.

The PROTOS Abstract Machine (PAM) which is an extension of the Warren Abstract Machine uses backtracking for rules contained in program bodies, and the deductive database inference engine (DDBIE) uses the process model for rules contained in database bodies. The PROTOS-L system integrates the DDBIE into the PAM and implements the interface of both inference engines with a rather small set of communication functions.

Since program bodies may call predicates implemented in database bodies, but not vice versa, PROTOS-L alltogether supports backtracking on top of set-oriented retrieval from relational databases. It is the decision of the PROTOS-L programmer to determine the boarder line between upper system modules which shall be evaluated by backtracking and lower system modules which shall use set-oriented evaluation. Even more, the programmer can easily change the evaluation strategy for modules at the border line: he only has to change a database body into a program body or vice versa.

The process model evaluates predicates programmed in a database body in a set-oriented way and incrementally on demand. It was adapted for queries searching in a large search space of ground facts. The process model is in superior to the Prolog evaluation strategy, whenever a large search space has to be searched in order to find a solution to a given goal. If, however, the search space needed in order to find a first solution to a given goal is smaller and further solutions are not required, then backtracking may be preferable. PROTOS-L provides both evaluation strategies for function free logic programs.

In summary, the PROTOS-L system which integrates both inference engines supports adaptive query evaluation of small search spaces as well as of large search spaces.

References

[Bancilhon and Ramakrishnan, 1986] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington D.C., 1986.

[Beierle, 1989] C. Beierle. Types, modules and databases in the logic programming language PROTOS-L. In K. H. Bläsius, U. Hedtstück, and C.-R. Rollinger, editors, *Sorts and Types for Artificial Intelligence*, Springer-Verlag, Berlin, Heidelberg, New York, 1989. (to appear).

[Beierle and Böttcher, 1989] C. Beierle and S. Böttcher. PROTOS-L: Towards a knowledge base programming language. In *Proceedings 3. GI-Kongreß Wissensbasierte Systeme, Informatik Fachberichte*, Springer-Verlag, Berlin, Heidelberg, 1989.

[Böttcher, 1990a] S. Böttcher. Development and programming of deductive databases with PROTOS-L. In L. Belady, editor, *Proc. 2nd International Conference on Software Engineering and Knowledge Engineering*, Skokie, Illinois, USA, 1990.

[Böttcher, 1990b] S. Böttcher. An inference engine for function free logic programs. In W.T. Tsai, editor, *Proc. 2nd International Conference on Tools for Artificial Intelligence*, Washington D.C., 1990. (to appear).

[Böttcher, 1990c] S. Böttcher. A tool kit for knowledge based production planning systems. In M. Tjoa, editor, *Proc. International Conference on Data Base and Expert System Applications*, Vienna, Austria, 1990.

[Böttcher and Beierle, 1989] S. Böttcher and C. Beierle. Data base support for the PROTOS-L system. *Microprocessing and Microcomputing*, 27(1–5):25–30, August 1989.

[Garidis and Böttcher, 1990] C. Garidis and S. Böttcher. A clustering concept for incremental loading of large PROLOG programs. In H.-J. Appelrath, A.B. Cremers, and O. Herzog, editors, *The EUREKA Project PROTOS*, Springer-Verlag, 1990. (to appear).

[Hulin, 1989] G. Hulin. Parallel processing of recursive queries in distributed architectures. In *Proceedings of the 15th International Conference on Very Large Data Bases*, Amsterdam, 1989.

[Meyer, 1989] G. Meyer. *Regelauswertung auf Datenbanken im Rahmen des PROTOS-L-Systems.* Diplomarbeit Nr. 630, Universität Stuttgart, December 1989.

[Sauer, 1990] J. Sauer. Design and implementation of a heuristic planning algorithm. In H.-J. Appelrath, A.B. Cremers, and O. Herzog, editors, *The EUREKA Project PROTOS*, Springer-Verlag, 1990. (to appear).

[Semle, 1989] H. Semle. *Erweiterung einer abstrakten Maschine für ordnungssortiertes Prolog um die Behandlung polymorpher Sorten.* Diplomarbeit Nr. 583, Universität Stuttgart und IBM Deutschland GmbH, Stuttgart, April 1989.

[Slahor *et al.*, 1990] L. Slahor, F. Reuter, and H. Schildknecht. Scheduling problems: a user's perspective. In H.-J. Appelrath, A.B. Cremers, and O. Herzog, editors, *The EUREKA Project PROTOS*, Springer-Verlag, 1990. (to appear).

[Vieille, 1988] L. Vieille. From QSQ towards QoSaQ: global optimization of recursive queries. In *Proceedings of the 2nd International Conference on Expert Database Systems*, Virginia, 1988.

[Warren, 1983] D. Warren. *An Abstract PROLOG Instruction Set.* Technical Report 309, SRI, 1983.

[Warren, 1989] D.S. Warren. *The XWAM: A Machine that Integrates Prolog and Deductive Database System Query Evaluation.* Technical Report 89/25, CS Department, SUNY Stony Brook, 1989.

[Wirth, 1983] N. Wirth. *Programming in Modula-2.* Springer, Berlin, Heidelberg, New York, 1983.