A DEDUCTIVE DATABASE ARCHITECTURE BASED ON PARTIAL EVALUATION

Li Lei
Ji Lin University
Dept. of Computer Science
Chang Chun
CHINA

G.-H. Moll
Université Laval
Departement d'informatique
G1K 7P4 Ste-Foy, Québec
CANADA
Tel: 418-656 5189
e-mail: moll@lavalvm1.bitnet

J. Kouloumdjian
INSA de Lyon
43, Bd. du 11 Novembre 1918
F-69622 Villeurbanne Cedex
FRANCE
tel: 78-94-80-99
e-mail: koulou@frcpn11.earn

This work was partially sponsored by the European Communities, as an ESPRIT project (P-530 EPSILON)

Abstract

The implementation of a logic programming language for database management systems is a possible way to build a knowledge base management system. It allows to re-use the know-how in the fields:

-of inference engines,

-of data management.

However, the strategy of each component is very different. SLD resolution leads to a one tuple at a time access to facts, opposed to the set oriented approach of databases. To face this problem, a new strategy, has been designed to keep the advantages of each part avoiding the drawbacks. The kernel of the system is a **Partial Evaluator**, designed as a metaprogram, which allows to bridge the impedance mismatch gap.

I-INTRODUCTION

There has been a growing interest in the past few years for systems having more extended functionalities than classical database management systems (DBMS): Extensions of modelling power, of knowledge representation languages, greater deductive capabilities /GALL-81/GALL-84/FURU-84/, various reasoning paradigms. Systems having such features are called Knowledge Base Management Systems (KBMS).

Using logic programming (Prolog-like languages) as an implementation tool seems a promising way for realizing such systems /KUNI-82/. There are several arguments for such a choice: well defined syntactic and semantic basis for the language, existence of a standard proof procedure (SLD resolution) for a subset of clauses, implementation of interpreters and compilers for the language.

A possible way is to use the cooperation between existing systems having complementary functionalities and a number of recent research works have been devoted to building systems coupling logic programming and data base technology. However, to be efficient, these systems need to use sophisticated techniques such as parallelism between processors /CERI-85/, syntactic and semantic query simplifications from Prolog to data manipulation languages /VASS-84/JARK-84/ or logic programs' compiling in order to minimize data flow between Prolog and DBMS. This last technique, called partial evaluation /KOMO-82/VENK-84/ is a source to source Prolog compilation technique, and as so, produces more specialized programs than the user's one for a particular query. To cope with such a technique, a control of Prolog's inference process is needed and can be achieved by using metaprogramming /KOWA-79/.

This paper describes a KBMS based on the cooperation of Prolog and DBMSs, and the underlying techniques, namely partial evaluation and meta-programming, which allow a great design flexibility, and easy and efficient strategy modifications. The paper is organized as follows: Section 2 presents some of the recent works using the same approach. In section 3, metaprogramming and partial evaluation which are the basic implementation tools of the prototype are described. Section 4 details the treatments of built-in Prolog predicates during partial evaluation. The architecture of the resulting KBMS is given in section 5.

II-STATE OF THE ART

The cooperative approach for building KBMSs consists in making work together:

- A deductive system: Prolog /COLM-83/
- A fact manager: one (or more) relational DBMS(s) /CODD-70/.

... through an interface.

Of course the choice of relational databases is due to the easy model mapping between logic predicates and relations, which appear as facts fully instantiated by atoms. Prolog and DBMSs can go on being available in standalone.

From a technical point of view, this constraint implies efficiency problems. The researches on this subject are oriented to developing optimization strategies.

Demolombe and Cuppens /DEMO-86/ propose an interface between Prolog and DBMSs via a delayed evaluation. In this approach, the SLD resolution tree is explored a first time just propagating the instantiations, and freezing the evaluation of the databases predicates. This method and the one that we have used (partial evaluation), are very similar. Let's point out, however, some differences:

The "delayed evaluation" technique is powerful for the treating and optimizing formulas including nested conjunctions and disjunctions.

The partial evaluation technique of the so called "Epsilon" (ESPRIT project P-530) prototype allows an efficient treatment of built-ins, particularly those that can be evaluated by the databases.

In /BOCC-86/ECRC-86/, Bocca describes the EDUCE system architecture, which is an integration of Prolog with the INGRES DBMS. The EDUCE strategy is mid-way between a "one tuple at a time" strategy and the "set oriented" access technique. The automated way in and out

between those two extreme strategies, at the evaluation time, is piloted by efficiency considerations for the recursive rules. Those last ones seem to be more efficiently solved one tuple at a time /VIEL-88/.

Vassiliou's paper /VASS-84/ insists on the power of the metaprogramming technique as a syntactic and semantic optimization tool, via the use of domains and integrity constraints meta-information, typically functional dependencies. A query is syntactically transformed depending on the functional dependencies it is involved in. Its internal form is then a table structure. The database is accessed by the SQL data manipulation language (loose coupling).

The DEDUCE2 /CHAN-78/ system includes a deductive system with quantifiers, which allows querying as well as virtual views definition and expressing integrity constraints. An user's request is transformed into an expression where only real data relations occur, then the consistency is checked out with the constraints. The data extraction is done by the DBMS. Request compiling is allowed.

TECHNOLOGUE, a prototype from the IBM France scientific center /MAAM-84/, is a loose coupling between a deductive system based on production rules with variables, and a relational database. Among its features, let us note at the logical level the use of forward, and then backward chaining, a non monotonous reasoning strategy, a consistency control and an explanation module.

SABRE /MASI-85/ is a tree level architecture system. First one is an external PROLOG interface including DML features expressed by built-in predicates. Second one is a PROLOG inferential level with facilities to access data in mass memory. Third one is the physical level. It allows to see the database as a PROLOG facts manager, and remains invisible for the user.

This coupling approach has been adopted for the Epsilon prototype.

III-IMPLEMENTATION TOOLS

III-1 Metaprogramming

Logic Programming can be considered as the use of an inference engine to solve problems /STER-84/. The use of built-in predicates justifies the terminology "programming". However, standard Prolog classical strategy is not always the best suited one.
Implementing a strategy can of course be done in any language. But using Prolog as the implementation language has many advantages:

-First one is that unification is a standard Prolog feature, and has not to be rewritten.

-Second one is the possible use of standard Prolog backtracking.

So meta-programming is an elegant strategy control method, and also well suited for prototyping.

III-2 Partial Evaluation

This technique consists in transforming an initial sentence, or group of sentences, expressed in a particular language, into a new group of sentences expressed in the same language, but the execution of which will be more efficient. So it is a source to source compilation /KOMO-82/.

The price to pay for the improvement of efficiency is a diminution of the generality of the program. Let F be a function with parameters $x_1, x_2, \ldots, x_n$. If $x_1=a_1$, $x_2=a_2, \ldots, x_k=a_k$ ($k \leq n$) where the a_i are constants, one can replace F by a particular version F' verifying:

$$F'(x_{k+1},\ldots,x_n)=F(a_1,\ldots,a_k,x_{k+1},\ldots,x_n)$$

for any xj (j=k+1 à n).

In the case of Prolog (our case) ,the partial evaluator will /VENK-84/

◊1 **collect and delay edb-predicates** to be sent to database system, to minimise the number of calls to the database.

◊2 **instantiate the variables** in the edb-predicates as much possible, that is to transfer less tuples from the database.

◊3 **evaluate the builtins** whenever possible. That is, in the system, the user is allowed to use a large number of primitives in his program.

As it is an evaluation strategy, Partial Evaluation can be implemented as a Metaprogram /TAKE-86/.

IV- BUILTINS HANDLING IN PARTIAL EVALUATION

The goal of our approach is to allow users to use a large number of primitives in their programs (cut, fail, and, < >, write ...). However , assert and retract will not be allowed.

IV-1 Builtins' classification

A classification of the Prolog (C-PROLOG) builtins (also called pimitives) has been made according to the kind of treatment they have during partial evaluation.

•The first class called **L-primitive** (Logic-primitive) is the set of primitives without argument: true, fail

•The cut operator, !, is not a member of the class, though it is a primitive without argument. It is called **C-primitive** (Control- primitive).

•The third class called **V-primitives** (Variable-primitive) is the set of the primitives the arguments of which are variables. This class can be divided into two groups :

◊Group1 is the *arithmetic primitives* :
Y is X
and the *comparative arithmetic primitives* :
=:=, ==,<, >, =<, >=, .

◊Group2 is the *term control primitives* :
var, nonvar, atom, number,integer, fonctor, arg, =.., name.

•The fourth class called **I/O-primitive** is the set of input/output primitives. This class can be also divided into two groups:

◊Group1 is the *side-effect primitives* :

tell, write, put , told, see, seen, close, seeing, telling, rename, read, display, writeq, print, nl , geto, get, skip, tab, system.

◊Group2 is *I/O test primitive* :

exist.

•The fifth class : **P-primitives** (P for predicate) are the primitives the arguments of which are predicates:

not ,or (";" in C-PROLOG), and ("," in C_PROLOG).

IV-2 The treatment of builtins during partial evaluation

All the primitives described above are right literals in the sense that they appear only on the right side of clauses. Except side-effect primitives, they can be treated as edb-predicates and delayed until final evaluation. But this way of doing is too conservative. The partial evaluator is not only a tool for collecting edb-predicates, but also a tool for obtaining a program P' optimized. So, it is worthwhile evaluating as many primitives as possible during the partial evaluation phase.

Firstly, we deal with primitives which can be evaluated during the procedure of partial evaluation in our system.

_x is called a **free variable**, if it is a variable in a literal in the user's program and it has not been instantiated.

Clearly:

•L-primitives are independent primitives.

•A V-primitive (except "is") is an independent primitive if all the variables in it have been instantiated, even the variables in the functions.

In the case of " _x is E" where E is an arithmetic expression, "_x is E" is independent if there is no free variable in E.

•the value of an independent primitive is fixed during or after the partial evaluation.

•For P-primitives, see next page.

So we can ever evaluate independent primitive during the partial evaluation phase.

For ***example***, in the program :

```
P(_x, _y) :-
   A(_x1), B(_y1),
   <(_x, _y), D(_x, _y).
```

with the query P(1,2) the result is :

```
P(1, 2):-
   A(_x1), B(_y1),D(1, 2).
```

but with the query

P(2,1)

the result will be :

```
P(2, 1) :- A(_x1), B(_y1), fail.
```

P-primitive are rather difficult to treat, because of the nesting problem. For example, the primitive :

```
not ( and (P (_x, _y),
    not ( Q (_x, _y) ) ) ).
```

An argument is called **final-argument**, if it is **in** a P-primitive and it is **not itself** a P-primitive.

• A P-primitive is an **independent primitive** if all the final arguments occurring in it are independent primitives.

A final-argument may be a primitive but not an independent primitive, or a prolog predicate. In this case, the P-primitive is not an independent primitive, the evaluation of the primitive will be delayed. In the second case, the final-argument is treated as an initial literal.

For ***example***, consider the program :

```
P(_x, _y)  :-
   A (_x, _y),
   or( <(_x, 100), Q (_x, _y) ).
Q(_x, _y)  :-
   B (_x, _y),
   C (_x, _y).
C(a, b).
A(_x, _y) :- D (_x, _y).
```

With the query P(_x,_y), the result is :

```
P(_x, _y)  :-
   D (_x, _y),
   or( <(_x, 100), Q (_x, _y) ).
Q (a, b)  :-  B (a, b).
```

In this example, " or (<(_x, 100), Q (_x, _y))" is not an independent primitive because the final argument Q(_x,_y) is not an independent primitive.

We will see how the independent primitives are evaluated during the step of partial evaluation in the system.

The treatment of cut has been discussed in /LIJK-86/.

IV-3 The treatment of side-effect primitives

The side-effect primitive problem is a well-known problem in the techniques of partial evaluation /VENK-84/. Let us consider the ***example*** described in /VENK-84/:

```
acceptable (_class, _status ) :-
   equal(_class, highquality).
acceptable(_class,_status):-
   equal(_status, approved).
program ( _x, _y) :-
   write(anything),
   product ( 380, _x, _y),
   acceptable(_x, _y).
```

With the query, Program(_x,_y), it is transformed into :

```
program( _x, _y) :-
   write(anything),
   product(380, _x, _y),
   equal(_x, highquality).
program( _x, _y) :-
   write(anything),
   product(380, _x, _y),
   equal(_y, approved).
```

Equal defines equality:

Example1: equal(_x,_x).

(explicit unification)

Example2: equal(_x,_y):- _x == _y.
(term syntactic matching)
Because of backtracking, the primitive write would be executed twice here, when it is only once in the initial program .

We propose an approach to solve this problem: replace the conjunction of literals following a side effect primitive by a new literal having the same variables. Then go on with partial evaluation with this new head of clause.

With previous example, the result in our system will be:

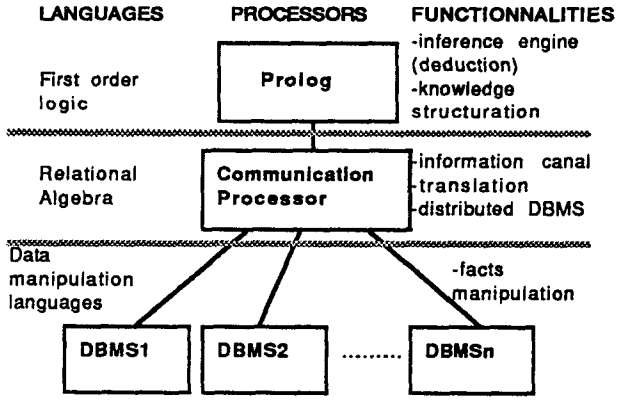
```
program(_x,_y):-
   write(anything),
   prog0(_x, _y).
prog0(_x,_y):-   product(380,_x,_y),
   equal(_x, highquality).
prog0(_x,_y):-
   product(380,_x,_y),
   equal(_y, approved).
```

V-IMPLEMENTATION

V-1 Architecture

As previously said, the architecture is based on the communication between Prolog and relational DBMSs.

Next schema presents the three level architecture, with the languages and the functionalities of each level.



The basic hypothesis is to allow the user to write Prolog programs that refer to facts that are stored as relations in the database, as if they were in Prolog main memory zone, in a transparent way.

The heart of the prototype is the communication processor (CP), the role of which is twofold:
-It is a bidirectional transfer canal.
-But it is also a distributed DBMS performing the multibase operations (typically multibase joins)

This architecture was designed to propose the following objectives:

-give the user an unique data model and a unique query language, namely Prolog. This is justified by the Prolog's success in knowledge representation among others languages.

-In the framework of this architecture, the Prolog inference engine performs deduction on the whole system, that is upon Prolog data (rules and facts) as well as bases data (facts). Note that this is only deduction and never updating, since Partial Evaluation, as a source to source compiler, cannot treat "assert" and "retract".

-Prolog plays the additional role of a programmable component. This allows developers to build at the Prolog level, knowledge structuration mechanisms, user's interfaces and so on.

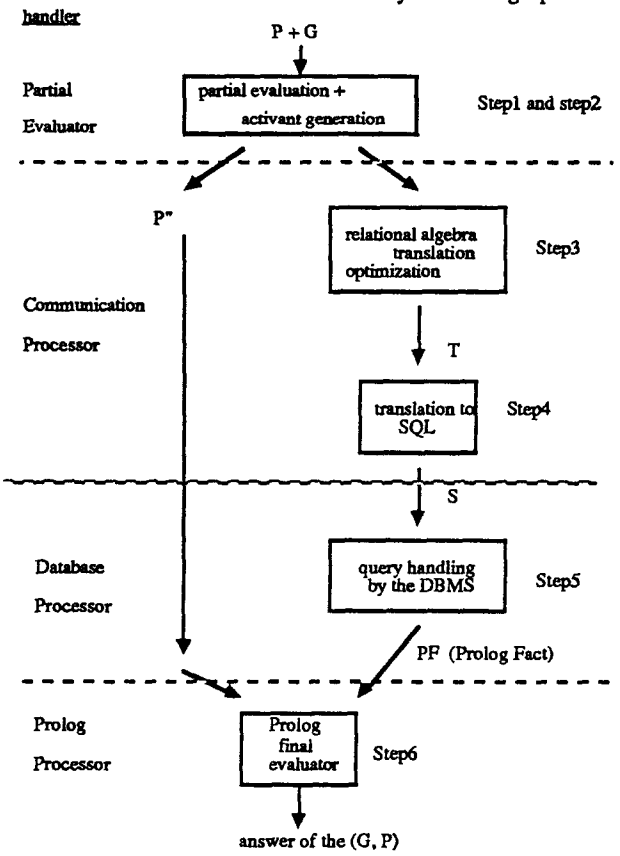
-This architecture makes clear the repartition of functionalities:

* great amounts of facts management at the level of the databases
* multibase operation at the level of the communication processor.
* deduction in Prolog.

V-2 Evaluation strategy

Since Prolog and database have not the same strategy, one tuple at a time for Prolog, i.e. backward chaining, and set oriented for databases, i.e. forward chaining, the, Evaluation Strategy has been modified, that is the SLD tree is not explored in a standard way. In order to minimize the inference engine's calls to databases, partial evaluation techniques has been used to gather elementary calls into global calls and to have a maximal instantiation in the calls to the database /VENK-84/.

The communication is illustrated by a linear graph:



•STEP 1 (which is a partial evaluation) uses the source program P and a goal G as input (P is a full Prolog

program, with recursive predicates, builtins and functions). The result of the partial evaluation is still a Prolog program P', but the edb-predicates in user's program have been collected. The variables in the edb-predicates have been partially instantiated, and several primitives have been evaluated. That is, the source program has been optimized.

•STEP 2 is the activants generator /MOLL-87/ which uses the program P', that is the result of STEP1, as input. The task of the activants generator is to group the dbcalls and to rename the dbcall-groups by so called "activants" predicates. The activants are the predicates which refer to facts brought from the external data base to the inference engine work bases via the communication processor. The activants generator separates the input program P' into two, the program P" where the conjunction of edb-predicates are changed to activants (a collection of dbcalls), and a set of activants. For example , if the program P' is:

```
P(_x,_y):-
   edb(f(_x,_z)),
   edb(m(_z, _y)),
   P(_z, _y).
P(_x, _y):-
   edb(f(_x,_y)),
   edb(m(_x, y)).
```

Where "edb" marks the predicates referring to database relations. "edb" has been added at partial evaluation time.

The activant generator transforms the program P' into two parts named P" and A.

P" :

```
P(_x, _y):-
   activant0(_x,_y,_z),
   P(_x, _z).
P(_x, _y) :-
   activant1(_x, _y).
```

A :

```
activant0(_x,_y,_z):-
   edb(f(_x, _z)),
   edb(m(_z, _y)).
activant1(_x, _y) :-
    edb(f(_x, _y)),
   edb(m(_x, _y)).
```

In the case of Venken's example (§ III-2), the output is

```
prg(_x,violette):-    activant1(_x).
prg(_x,jan) :-        activant2(_x).
prg(_x,stanis):-      activant2(_x).
prg(_x,henry):-
   activant3(_x).
prg(_x,_y):-
   activant4(_x,_y).
```

With the definitions of activants:

```
activant1(_x):-
   f(_x,anna).
activant2(_x):-       f(_x,violette).
activant3(_x):-       f(_x,henriette).
activant4(_x):-
   f(_x,_z),f(_z,_y).
```

Note that this the solution allows to process every selection operation by the database management system, which is much more efficient than PROLOG for this purpose. Facts returned from the Data Base systems will be put in various work bases, decreasing the work of inference engine at unification time.

•STEP 3 is the activant translation /MOLL-87/ which only uses the set of activants as input. In this step, the set of activants is translated into relational algebra. Some optimization work is done at this step, namely getting down unary operators in the relational trees /DELO-82/GARD-84/.

•In STEP 4, the program is then transformed into a program called PDB program written in a database system language, namely SQL in the case of the prototype.

•STEP 5 is a general database system which uses PDB program as input. Then a DB result translator will translate the result of DB system into a set of facts in Prolog form. Then, the set will be loaded to the work-space of Prolog.

•STEP 6 is done by the Prolog interpreter which uses the facts loaded from the database system and the program P" generated by partial evaluation and activant generation (STEP1 and STEP2), as input.

The present state of the prototype is a coupling of C-Prolog (Edinburgh, version 1.5) with the unique database Informix (running with the SQL DML). It has been implemented on a SUN 3/50 machine under Unix operating system. The prototype has been reviewed in the framework of P-530 ESPRIT project.

VI-CONCLUSION

Partial Evaluation is a powerful tool to implement a suitable resolution strategy for coupling Prolog to databases. The obtained strategy keeps the advantages of each component:

-at the level of the database, facts are processed in a set oriented way, which is the more efficient way for big amounts of facts. The results of this processing is a set of facts that are inserted in Prolog zone.

-at the level of Prolog, the inference is processed in two steps: before the access to the databases, extensional database predicates are frozen, but also the global program is optimized by the Partial Evaluation technique. After the access to the bases, Prolog standard strategy can run.

This strategy is a mixing of forward and backward chaining. As so, Partial Evaluation is an automatical dispatcher of functionalities: instantiations (pushed down by the PE) and reference of variables are treated by the bases as selections and joins whenever edb predicates are concerned. Projections are generated when all the attributes of the resulting relational are not useful to the inference mechanism. The rest of the inference is processed by Prolog.

Another prototype for the same objective has been realized with a new technique named the "half interpreted,

half compiled" approach (Lei, Moll et Kouloumdjian 1988) which is also based on the EPSILON partial evaluator. With this new approach inspired from (Venken 1984) and (Ceri, Gottlog et Wiederhold 1985), only facts which are relevant to the SLD resolution are loaded from database, which saves space in Prolog and produces a more efficient evaluation.

VII-BIBLIOGRAPHY

/BOCC-86/ J.B.Bocca, H.Decker, J.M.Nicolas, L.Vielle, M.Wallace - Some steps towards a DBMS based KBMS. - information Processing (North Holland)-1986-pp1061-1067 -

/BOWE-85/ K.A. Bowen - Meta-level programming and Knowledge representation. - Computing, September '85 -

/CERI-85/ S. Ceri, G. Gottlob, L. Lavazza - "Transformation and optimization of logic queries: the algebraïc approach". - Internal report- Politecnico di Milano. -

/CHAN-78/ C.L. Chang - Deduce 2: Further investigations on deduction in relational Data Bases. - In Logic and Dara Bases, Plenum Press, 1978, pp. 201-236 -

/CODD-70/ E.F. Codd - A relational model of Data for Large shared Data Banks. - Comm. of the ACM; vol 13 N°6 june 1970 pp 377-387 -

/COLM-83/ A. Colmerauer, H. Kanoui, M. Van Caneghem - Prolog, bases théoriques et développements actuels. - TSI, vol 2, n°4 1983 pp 271-311 -

/DELO-82/ C. Delobel, M. Adiba - Bases de Données et Sytèmes Relationnels. - Collection Dunod Informatique - 449 pages - 1982 -

/DEMO-86/ R. Demolombe - A PROLOG-relational DBMS interface using delayed evaluation. - in /WILP-86/ -

/ECRC-86/ European Computer-Industry Research Centre: -J. Bocca: On the Evaluation Strategy of EDUCE. - Proceedings of 1986 ACM-SIGMOD - International Conference on Management of Data - May 1986, Washington, USA - -J. Bocca: EDUCE, a Marriage of convenience: - Prolog and a Relational DBMS. - Third Symposium on Logic Programming - September 86, Salt Lake City, USA -

/FURU-84/ K. Furukawa, A. Takeuchi, S. Kunifuji, H. Yasukawa, - M. Ohki and K. Ueda, Mandala: - A logic based knowledge programming system. - Proc. Int'l Conf. on Fifth Generation Computer Systems (1984), - 613-622. /GALL-81/ H. Gallaire - Impacts of Logic on data bases. - Proc. VLDB '81, Cannes 1981 - pp 248-259 -

/GALL-84/ H. Gallaire, J. Minker, J.M. Nicolas - Logic and data bases, a deductive approach. - Computing Survey vol. 1, n°2, June 1984 pp 153-185. -

/GARD-84/ G. Gardarin - Bases de données : les systèmes et leurs langages. - Ed. EYROLLES 1984. -

/JARK-84/ M. Jark, J. Clifford; Y. Vassiliou - An optimized front end to a relational query system - ACM 84- 0-89 791-128/84/006/296 - pp 296-306 /KLM-88/ J. Kouloumdjian, L. Lei, G.H. Moll - Vers une architecture répartie des bases de connaissances - TSI, n° 2 vol 7 '88 -

/KOMO-82/ J.Komorowski - Partial evaluation as a means for inferencing data structures - in an applicative language: a theory and implementation in the case of - PROLOG. - LINKOPING University-S-581 83-1982 -

/KOUL-85/ J. Kouloumdjian, P. Lebaube, B. Lepape - The interface between the data base management system and the inference machine. - EPSILON technical report 4, ESPRIT Project P-530, November '85 -

/KOWA-79/ R. Kowalski - Logic for problem solving (287 pages) - Elsevier Science Publishing Co., Inc. -

/KUNI-82/ S. Kunifigi, H. Yokota - PROLOG and relational data bases for fifth generation computers - Workshop on logical basis for data bases-TOULOUSE-1982 -

/LIJK-86/ L. Lei, J. Kouloumdjian - An implementation of a partial evaluation system. - Internal report - University of Lyon (1986). -

/LKM-88/ L. Lei, G.H. Moll , J. Kouloumdjian , - Prolog-DBMS coupling: a hybrid approach, half - interpreted, half compiled - 3rd int. conf. on data and knowledge bases - Jerusalem '88 -

/MAAM-84/ A. Manga, A. Mouline - L'utilisation des systèmes experts en informatique de gestion: un - exemple d'aide à la décision dans l'interrogation d'une base de données. - Thèse de 3ième cycle, Université de Paris IX Dauphine, 1984 -

/MASI-85/ M. Bouzeghoub, M. Jouve - PROLOG-SABRE: 3 possibilités de couplage. - MASI-Paris VI, Rapport de recherche n° 75, 1985 -

/MOLL-87/ G.H. Moll, - Un langage pivot pour le couplage de Prolog avec des bases de données: - formalisation et environnement opérationnel - Thèse de Doctorat, Octobre '87 -

/STER-84/ L. Sterling, - Expert system = Knowledge + Meta-interpreter - Tech. Rep.CS84-17, Weizmann Institute of Science, Rehovot, - Israël (1984). -

/TAKE-86b/ A. Takeuchi, - Affinity between meta interpreters and partial evaluation. - Information Processing-86, H.-J. Kugler, Ed. - (Elsevier Science Publishers B.V., 1986). -

/VASS-84/ Y. Vassiliou, J. Clifford, M. Jarke - Access to specific declarative knowledge by expert systems: the impact of logic programming - Decision Support Systems-Vol 1-NO 1-1984 -

/VENK-84/ R. Venken - A PROLOG meta-interpreter for partial evaluation and its - application to source to source transformation and query - optimization - ECAI-1984 -

/VIEL-88/ L. Vieille - Recursive query Processing: Fundamental algorithms and the DedGin system - Université d'été 88 - "Organisation et traitement des connaissances en intelligence artificielle", Lyon, France 4-8 july 1988 -

/WARR-84/ D. Warren, D. Bowen, L. Byrd, L. Pereira - C-Prolog user's manual, Version 1.5 - February 22, 1984 -