TR-302

Partial Evaluation of Queries in Deductive Databases

by
C. Sakama and H. Itoh

September, 1987

Partial Evaluation of Queries in Deductive Databases

CHIAKI SAKAMA* AND HIDENORI ITOH

Institute for New Generation Computer Technology
Mita Kokusai Building 21F
1-4-28 Mita, Minato-ku, Tokyo, Japan

Abstract

This paper presents two strategies for using the partial evaluation method to compile an intensional database (*idb*) in the form of a set of Horn clauses by a set of queries. The least generalized query (*LGQ*) method is suitable when there are many compatible queries, while the most generalized query (*MGQ*) method works better when the predicates in the queries have a hierarchical ordering in the *idb*. In both cases, partial evaluation is used to preprocess the *idb* in order to obtain a subquery to direct access to the extensional database (*edb*).

1 Introduction

A deductive database usually consists of a large extensional database (*edb*) and a comparatively small intensional database (*idb*). The *edb* is a set of base relations whose tuples are explicitly stored, while the *idb* is a set of function free Horn clauses which define virtual relations between tuples. The tuples of a virtual relation are derived from the clauses in the *idb* and the facts in the *edb*.

In such deductive database systems, a query is evaluated with an inference system by compiling the *idb* and with a relational database management system by retrieving the *edb* [Rei 78]. There are known two methods to perform this evaluation: the *interpretive* method and the *compiled* method [Gall 84]. The interpretive method evaluates a query in an *idb* and

* *uucp: {enea,inria,kddlab,mit-eddie,ukc}!icot!sakama*
csnet: sakama%icot.jp@relay.cs.net

interleaves search in an *edb* over and over again until the query is wholly evaluated, while the compiled method compiles a query wholly in advance in an *idb* and produces a set of subqueries evaluable only in the *edb*.

However, when there is a large extensional database, the compiled approach is considered more effective to partially evaluate a query wholly in advance in an *idb* and reduce the access cost to the *edb* rather than evaluate one after another in the interpretive approach [Chak 82]. For example, the deductive database systems such as [Yoko 84] and [Boc 86] have been designed based on this approach.

Partial evaluation (computation) [Futa 83] or mixed computation [Ersh 82] of a program is considered as a compilation technique for a program with incomplete data, and is useful for iterative computation, program specification and meta-programming optimization [Take 85] in practice.

The compiled method in a deductive database is considered as one of the applications of such a partial evaluation technique to a query optimization, but when there is a set of queries to be evaluated, it is inefficient to partially evaluate each query independently in an intensional database. To minimize the cost of such multiple query processing, it is effective to perform evaluation once that is common to some of the queries and use the common intermediate results of evaluation to obtain answers for those queries. In [Chak 86], it is achieved by using a connection graph, that is, grouping a set of queries, exploiting the common subexpressions and generating a single plan to evaluate these queries.

In this paper, it is achieved as a natural extension of the compiled method in a deductive database by using the generalization techniques. Section 2 presents the notion of the partial evaluation method for query processing in deductive databases. Section 3 represents an application of the partial evaluation method for multiple query processing in deductive databases. Performance evaluation and some discussion are given in section 4.

2 Partial evaluation in deductive databases

Partial evaluation of a query in a deductive database is defined as an evaluation of the query in the *idb*. It is the transformation of a query which includes virtual relations defined in the *idb* into subqueries which are the Horn clauses evaluable only in the *edb*.

This process is represented as follows:

$$Q_{idb} = c(idb, Q) \qquad (1)$$

where Q denotes a query to be evaluated, $c(idb, Q)$ denotes a query compilation in the *idb* with deduction, and Q_{idb} presents the result of the compilation which is the subquery for the *edb*.

After this compilation, Q_{idb} is evaluated in the *edb*:

$$Q_{idb,edb} = r(edb, Q_{idb}) \qquad (2)$$

where $r(edb, Q_{idb})$ denotes the evaluation of Q_{idb} in the *edb* with relational operations and $Q_{idb,edb}$ represents the result of the query evaluation in the *idb* and the *edb*, which is a set of answer tuples for the query.

As is well known, the central problem of such an evaluation is a termination condition and answer completeness for a recursive query in a deductive database, and many studies have been done so far [Ban 86]. In this paper, the *Horn Clause Transformation (HCT)* method [Miya 86] is used for treating such recursive queries, that is, compiling a query in an *idb* and generates some subqueries which define the query only with *edb* predicates appearing in the *edb* and recursive predicates which are evaluated iteratively in the *edb*.

Example 2.1 Suppose the following *idb*.

$p(X, Y) : -q(X, Z), r(Z, Y).$
$q(X, Y) : -s(X, Y).$
$q(X, Y) : -s(X, Z), t(Z, Y).$
$t(X, Y) : -u(X, Z), q(Y, Z).$
$v(X, Y) : -w(Y, X).$

Using the HCT method, a query $? - p(a, Y)$ is partially evaluated in the *idb* as:

Q_{idb}

$p(a, Y) : -q(a, Z), r(Z, Y).$
$q(X, Y) : -s(X, Y).$
$q(X, Y) : -s(X, Z), u(Z, W), q(Y, W).$

These are the subqueries to be evaluated in the *edb*, where q is a recursive predicate and r, s and u are *edb* predicates. Note that, in the above example, the recursive clauses are not instantiated for later usage. □

3 Application to multiple query processing

Suppose there is a set of queries to process in a deductive database. In this case, it is inefficient to partially evaluate each query independently as is presented in the previous section. This section gives two methods as applications of the partial evaluation for multiple query processing in deductive databases.

3.1 Generalized query

First, some basic definitions and terminology used in the following discussion are given.

Definition 3.1 A term and an atom (atomic formula) are defined as follows:

1. A variable or a constant is a term, and nothing else is a term. (As stated earlier, we consider only function free cases.)
2. If p is an n-ary predicate and $t_1, ..., t_n$ are terms, then $p(t_1, ..., t_n)$ is an atom.

In particular, atoms of the same predicate and the same number of arguments are called *compatible*. □

Definition 3.2

1. Given atoms P and Q, Q is *more general* than P iff there exists a substitution θ such that $P = Q\theta$, written $P \sqsubseteq Q$.
2. Let S be a set of atoms, then L is a *least generalization* of S iff

(i) $\forall T \in S, T \sqsubseteq L$

(ii) $\forall L_i,\ if\ \forall T \in S\ and\ T \sqsubseteq L_i\ then\ L \sqsubseteq L_i.$

3. Let S be a set of atoms, then L is a *most generalization* of S iff

(i) $\forall T \in S, T \sqsubseteq L$

(ii) $\forall L_i,\ if\ \forall T \in S\ and\ T \sqsubseteq L_i\ then\ L_i \sqsubseteq L.$

Here $\sqsubseteq$ denotes partial ordering over the atoms. □

Example 3.1 The least and most generalizations of the atoms $\{p(a,a), p(a,b)\}$ are $p(a,X)$ and $p(X,Y)$, respectively. □

The generalization of atoms has been applied to inductive reasoning [Plot 70], or-parallel search strategy [Fish 75] and so on. Here, these generalization techniques are applied to multiple query processing in deductive databases.

3.2 LGQ method

In all subsequent discussion, a query composed of a single atom is assumed. For example, a query composed of several atoms such as $? - p(X,Y), q(Y,Z), r(Z)$ is considered as a query $? - s(X,Y,Z)$ and a clause $s(X,Y,Z) : -p(X,Y), q(Y,Z), r(Z)$.

First, the notion of least generalized queries is defined.

Definition 3.3 Given a set of queries, the least generalization of compatible queries in the set is called the *least generalized query* for compatible queries and a set of the least generalized queries for all the maximal compatible sets of queries in the given set is called the *least generalized queries* (LGQ). □

Example 3.2 The LGQ of a set of queries $\{p(a,a), p(b,b), q(a,Y), q(X,b), r(a)\}$
is $\{p(X,X), q(X,Y), r(a)\}$. □

When a set of queries to be evaluated is given, an LGQ of the set can be obtained by the following steps.

1. Classifying a set of queries into subsets of compatible queries.
2. Deriving the least generalized query of each subset.

The algorithm for deriving the least generalization is given in [Rey 70] and [Plot 70]. We use this LGQ for the partial evaluation of queries.

Suppose a set of queries **Q** is given, then the LGQ of **Q** is presented as:

$$LGQ = \sqcup \mathbf{Q} \tag{3}$$

Partial evaluation of a LGQ in the idb is presented as follows:

$$LGQ_{idb} = c(idb, LGQ) \tag{4}$$

where $c(idb, LGQ)$ denotes a compilation of LGQ in the idb.

Such LGQ_{idb} is evaluated in the edb with the selection condition of the given queries:

$$\mathbf{Q}_{idb,edb} = r(edb, \sigma_Q(LGQ_{idb})) \tag{5}$$

where σ_Q denotes a selection operation under the condition of $\mathbf{Q}$, and $\mathbf{Q}_{idb,edb}$ presents a set of answer tuples for the given set of queries.

Example 3.3 Suppose the same *idb* as in *Example 2.1* and a set of queries $\mathbf{Q}$.

$\mathbf{Q} = \{p(b,Y), p(X,d), t(X,c), t(b,c), v(X,f)\}$

Then, the LGQ of these queries is:

$LGQ = \{p(X,Y), t(X,c), v(X,f)\}$

and is partially evaluated in the *idb* as follows:

LGQ_{idb}

$p(X,Y) :- q(X,Z), r(Z,Y).$
$q(X,Y) :- s(X,Y).$
$q(X,Y) :- s(X,Z), u(Z,W), q(Y,W).$
$t(X,Y) :- u(X,Z), s(Y,Z).$
$t(X,Y) :- u(X,Z), s(Y,W), t(W,Z).$
$v(X,f) :- w(f,X).$

These are the union of the evaluated results of each query in the LGQ and are evaluated in the *edb* with the following selection conditions, $\sigma_{X=b \vee Y=d}(p(X,Y))$, $\sigma_{(X=b,Y=c) \vee Y=c}(t(X,Y))$, and $\sigma_{Y=f}(v(X,Y))$. □

3.3 MGQ method

This section presents another optimization technique for multiple query processing. First, the notion of most generalized queries is defined.

Definition 3.4 Given a set of queries, the most generalization of compatible queries in the set is called the *most generalized query* for the compatible queries, and the set of the most generalized queries for all the maximal compatible sets of queries in the given set is called the *most generalized queries* (MGQ). □

Example 3.4 The MGQ of a set of queries $\{p(a,a), p(b,b), q(a,Y), q(X,b), r(a)\}$

is $\{p(X,Y), q(X,Y), r(X)\}$. □

An MGQ can be obtained from a given set of queries likely in the case of LGQ by simply assigning different variables to the arguments of the compatible queries.

Next, the partial ordering over predicates is defined.

Definition 3.5 Partial ordering over predicates appearing in a set of Horn clauses is defined as follows:

1. Suppose a set of Horn clauses S, and $\exists C_k(\in S)$. When the predicates in C_k, p_i and $p_j(p_i \neq p_j)$ satisfy the condition $p_i \in Head(C_k), p_j \in Body(C_k)$, then it is said that p_i is *higher* than p_j (p_j is *lower* than p_i) and written $p_j \preceq p_i$. ($Head(C_k)$ and $Body(C_k)$ denote the sets of predicates appearing in the head part and the body part of the clause C_k, respectively.)
2. If $p_j \preceq p_i$ and $p_i \preceq p_j$, then $p_i \sim p_j$. □

In the above definition, it was assumed that atoms with the same predicate are compatible.

Example 3.5 For the set of Horn clauses S,

$$S = \{p(X,Y) :- q(X,Z), p(Z,Y), \quad q(X,Y) :- r(X,Z), s(Z,Y), \quad s(X,Y) :- q(Y,X)\}$$

the ordering over predicate is defined as $r \preceq q \preceq p$ and $q \sim s$. □

Assume an *idb* is given as a set of Horn clauses, then the ordering over the predicates appearing in the *idb* is defined according to the previous definition. When a set of queries to be evaluated is given for this *idb*, some hierarchical ordering over the queries may be defined because the predicates of those queries are to be defined in the *idb*.

We use this hierarchical ordering over queries with the MGQ for the partial evaluation of queries.

Suppose a set of queries $\mathbf{Q}$ and its MGQ. When an ordering over the predicates of the MGQ is defined, they can be sorted according to the ordering:

$$s(MGQ) = \langle mgq_1, mgq_2, ..., mgq_n \rangle \tag{6}$$

where $s(MGQ)$ denotes the sorted MGQ and mgq_i is a most generalized query lower than mgq_j, if $i < j (1 \leq i, j \leq n)$.

A sorted MGQ should be partially evaluated in the *idb* from lower queries to higher queries, because the evaluated results of the lower queries can be used to evaluate the higher queries since the higher predicate is to be defined by the lower ones in the *idb*. Note that the most generalization is needed for this evaluation because the higher query evaluation may need more general results of the lower one during its evaluation.

By (6), partial evaluation of the sorted MGQ in the *idb* is presented as:

$$s(MGQ)_{idb} = c(idb, s(MGQ)) \tag{7}$$

where $c(idb, s(MGQ)) = \bigcup_i c(idb_i, mgq_i)$, $idb_i = c(idb_{i-1}, mgq_{i-1}) \cup idb^*_{i-1} (i \geq 2)$, $idb_1 = idb$, and idb^*_{i-1} denotes idb_{i-1} except for the clauses which have the same predicates with the results of the evaluation of mgq_{i-1} in the head. This means that mgq_i is partially evaluated in the *idb* using the evaluated results of the lower *mgqs* and $\bigcup_i c(idb_i, mgq_i)$ presents the union of the evaluated results of the sorted MGQ in the *idb*.

Finally, $s(MGQ)_{idb}$ is evaluated in the *edb* with the selection conditions of the given queries:

$$\mathbf{Q}_{idb,edb} = r(edb, \sigma_Q(s(MGQ)_{idb})) \tag{8}$$

where σ_Q denotes a selection operation under the condition of $\mathbf{Q}$, and $\mathbf{Q}_{idb,edb}$ presents a set of answer tuples for the given set of queries.

Example 3.6 Suppose the following *idb* and a set of queries $\mathbf{Q}$.

$p(X,Y) :- q(X,Z), r(Z,Y).$
$q(X,Y) :- s(X,Y).$
$q(X,Y) :- s(X,Z), t(Z,Y).$
$r(X,Y) :- t(X,Y), w(Y,Z).$
$t(X,Y) :- u(X,Z), q(Y,Z).$

$\mathbf{Q}$ = { p(a,Y), p(a,b), q(X,c), q(b,Y), r(X,f)}

In this case, the MGQ of $\mathbf{Q}$ is:

$MGQ = \{p(X,Y), q(X,Y), r(X,Y)\}$

and, according to the ordering over predicate in the *idb*, the MGQ is sorted as

$s(MGQ) = \langle \; (q(X,Y) \quad r(X,Y)) \; , \; p(X,Y) \; \rangle$

that is, $q \preceq p$, $r \preceq p$ and there is no ordering between q and r.

At first, $q(X,Y)$ and $r(X,Y)$ are evaluated in the *idb*,

$$q(X,Y) :- s(X,Y).$$
$$q(X,Y) :- s(X,Z), u(Z,W), q(Y,W).$$
$$r(X,Y) :- u(X,W), q(Y,W), w(Y,Z).$$

and the *idb* is transformed into the following idb_1 with these evaluated results:

$$p(X,Y) :- q(X,Z), r(Z,Y).$$
$$q(X,Y) :- s(X,Y).$$
$$q(X,Y) :- s(X,Z), u(Z,W), q(Y,W).$$
$$r(X,Y) :- u(X,W), q(Y,W), w(Y,Z).$$
$$t(X,Y) :- u(X,Z), q(Y,Z).$$

Secondly, $p(X,Y)$ is evaluated in this idb_1.

$$p(X,Y) :- q(X,Z), u(Z,W), q(Y,W), w(Y,U).$$

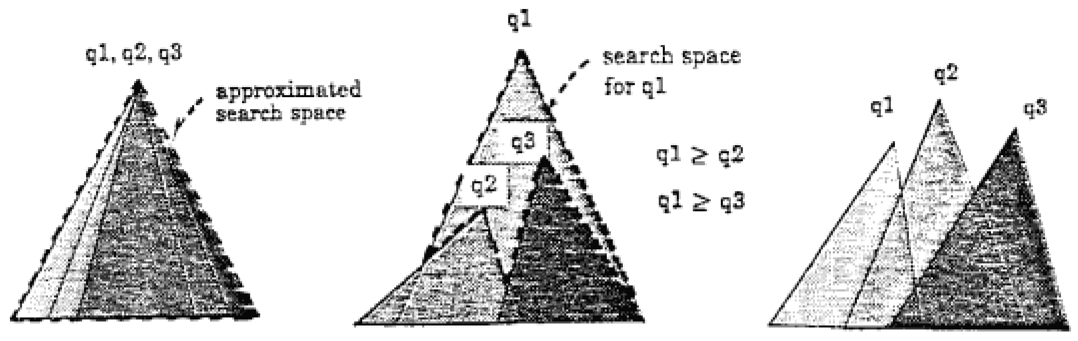
Finally, the partially evaluated MGQ in the *idb* can be obtained by the union of these results.

$s(MGQ)_{idb}$

$$p(X,Y) :- q(X,Z), u(Z,W), q(Y,W), w(Y,U).$$
$$q(X,Y) :- s(X,Y).$$
$$q(X,Y) :- s(X,Z), u(Z,W), q(Y,W).$$
$$r(X,Y) :- u(X,W), q(Y,W), w(Y,Z).$$

These subqueries are evaluated in the *edb* with the following selection conditions, $\sigma_{(X=a,Y=b)\vee X=a}(p(X,Y))$, $\sigma_{X=b\vee Y=c}(q(X,Y))$, and $\sigma_{Y=f}(r(X,Y))$. □

In the above example, $p(a,Y)$ and $p(a,b)$ need not have been most generalized for the evaluation of the queries, because they are the highest queries in the set and their evaluated results are not used by any other queries. (The least generalized query, that is $p(a,Y)$, can be used for compilation in this case.)



(a) LGQ method is effective (b) MGQ method is effective (c) LGQ and MGQ methods are not effective

Figure 1. Comparison of each method

4 Performance Evaluation

The previous section presented two strategies for the optimization of multiple query processing in deductive databases. Figure 1 shows the comparison of these methods.

The *LGQ* method is effective when there are many compatible queries in a given set of queries, because the same compilation for those queries can be approximated by their *LGQ* (Figure 1(a)). While the *MGQ* method is better when the predicates in the queries have a hierarchical ordering in the *idb*, because the compilation of queries can be performed incrementally from lower queries to higher queries (Figure 1(b)).

However, as in Figure 1(c), although the queries have common subqueries, both methods are useless because the queries are neither compatible nor hierarchical. In such a case, a bottom-up evaluation seems to be suitable rather than the top-down way, but this case is not discussed furthermore in this paper.

Now some experimental results of the effect of these methods are presented below.

The *LGQ* and *MGQ* processors, and *HCT* interpreter are implemented in DEC-10 Prolog. Table 1 gives the results of *LGQ* and *MGQ* processing. In the *LGQ* processing, an *LGQ* is generated from the given set of queries, and in the *MGQ* processing, an *MGQ* is generated from the given set of queries, which are sorted according to the ordering over predicates. In these experiments, the queries are assumed to be function free binary relations. Table 1(a) shows the execution time of the *LGQ* and *MGQ* processes for the different number of queries at the rate of $LGQ/Q = MGQ/Q = 0.6$, that is, the number of queries is reduced to 60% in

(a) LGQ / Q = MGQ / Q = 0.6

≠Q	5	10	15
LGQ	15	42	82
MGQ	14	36	74

(b) ≠Q = 10

/ Q	0.2	0.4	0.6	0.8
LGQ	31	37	43	44
MGQ	16	28	39	45

(msec)

Table 1. Execution time for the LGQ and MGQ processing

depth	5	10	15
time (msec)	863	5224	12647

Table 2. Execution time for HCT process

the *LGQ* or *MGQ*. Table 1(b) shows the time for the different rates of *LGQ/Q* and *MGQ/Q* with 10 queries.

Next, the performance improvement obtained by these strategies in compiling *idb* is shown. For measurement, it is used a sample *idb* which consists of function free Horn clauses, composed of binary relations without constants, and including linear recursive clauses at the rate of 60% for all clauses. The execution time of the *HCT* process for a single query is shown in Table 2. In this *idb*, the search space grows nearly exponentially, so does the costs increase with the depth.

For the evaluation of a set of queries in this *idb*, the following three ways are considered.

- Compiling each query iteratively in the *idb*.
- Using the *LGQ* method.
- Using the *MGQ* method.

We measured the execution time using each method for five and ten queries at the rate of $LGQ/Q = MGQ/Q = 0.6$. Figure 2 shows the comparison of the compiled execution time between these evaluations.

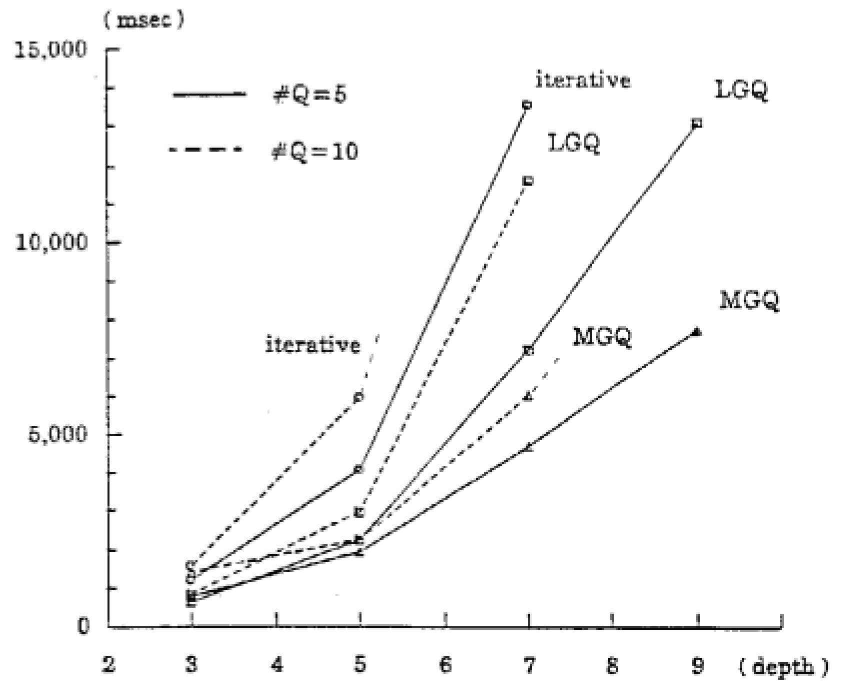


Figure 2. Comparison of performance evaluation

The set of queries used in this evaluation included some hierarchical queries, making the *MGQ* method more effective at deeper depths than other methods.

In these experiments, we assumed an *idb* which contains no constants, so the trade-off losses of efficiency due to the increased generality of the queries did not affect the results of the evaluation. However, when an *idb* contains many constants and the generalization techniques mentioned in this paper increase the search space for the queries, these methods may be less effective in practice.

5 Concluding remarks

This paper presented an application of a partial evaluation to multiple query processing in deductive databases. It introduced two methods: the *LGQ* method and the *MGQ* method. Both methods are a natural extension of the compiled approach for multiple query processing in deductive databases and can reduce the cost of compilation for a set of queries. Although function free Horn clauses were assumed according to the convention in deductive databases, these generalization techniques can be applied to the non function free cases in general. Further discussion will be needed in the real applications.

Acknowledgments

We would like to thank Yukihiro Morita, Nobuyoshi Miyazaki and our colleagues in the KBM project at ICOT for useful discussions on an earlier draft of this paper. We are also grateful to Toshiaki Takewaki for helpful comments and assistance in the performance evaluation in this paper.

References

[Ban 86] Bancilhon,F. and Ramakrishnan,R.: "An Amateur's Introduction to Recursive Query Processing Strategies", *Proc. ACM SIGMOD '86*, pp.16-52, 1986.

[Boc 86] Bocca,J., Decker,H., et al.: "Some Steps Towards a DBMS Based KBMS", *Proc. Information Processing Congress*, pp.1061-1067, 1986.

[Chak 82] Chakravathy,U.S., Minker,J. and Tran,D.: "Interfacing Predicate Logic Languages and Relational Databases", *Proc. 1st Int. Conf. on Logic Programming*, pp.91-98, 1982.

[Chak 86] Chakravathy,U.S. and Minker,J.: "Multiple Query Processing in Deductive Databases using Query Graphs", *Proc. 12th Int. Conf. on VLDB*, pp.384-391, 1986.

[Ersh 82] Ershov,A.P.: "Mixed Computation: Potential Applications and Problems for Study", *Theoretical Computer Science*, vol.18, pp.41-67, 1982.

[Fish 75] Fishman,D.H. and Minker,J.:"Π-Representation: A Clause Representation for Parallel Search", *Artificial Intelligence*, vol.6, pp.103-127, 1975.

[Futa 83] Futamura,Y.: "Partial Computation of Programs", *Lecture Notes in Computer Science*, vol.147, pp.1-35, 1983.

[Gall 84] Gallaire,H., Minker,J. and Nicolas,J.M.: "Logic and Databases: A Deductive Approach", *ACM Computing Surveys*, vol.16, no.2, pp.153-185, 1984.

[Miya 86] Miyazaki,N., Yokota,H. and Itoh,H.: "Compiling Horn Clause Queries in Deductive Databases: A Horn Clause Transformation Approach", *TR-183, ICOT*, 1986.

[Plot 70] Plotkin,G.D.: "A Note on Inductive Generalization", *Machine Intelligence*, vol.5, pp.153-163, 1970.

[Rei 78] Reiter,R.: "Deductive Question-Answering on Relational Data Bases", *Logic and Data Bases, Plenum Press*, 1978.

[Rey 70] Reynolds,J.C.: "Transformational Systems and the Algebraic Structure of Atomic Formulas", *Machine Intelligence*, vol.5, pp.135-151, 1970.

[Take 85] Takeuchi,A. and Furukawa,K.: "Partial Evaluation of Prolog Programs and its Application to Meta Programming", *Proc. Logic Programming Conference'85, ICOT*, 1985.

[Yoko 84] Yokota,H., Kunifuji,S., et al.: "An Enhanced Inference Mechanism for Generating Relational Algebra Queries", *Proc. 3rd ACM PODS*, pp.229-238, 1984.