ICOT Technical Report: TR-538

TR-538

Selection Propagation in Deductive Databases — From Pushing Selections to Magic Sets —

by
N. Miyazaki (Oki)

March, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191－5
Telex ICOT J32964

Institute for New Generation Computer Technology

Selection Propagation in Deductive Databases[1]

– From Pushing Selections to Magic Sets –

Nobuyoshi MIYAZAKI
(Oki Electric Industry Co., Ltd.)[2]

Abstract. This paper discusses the relationship between two optimization methods in deductive databases: the distribution of selections and the magic sets method. The former is a direct generalization of pushing selections in relational databases and the latter realizes a more general view of selection propagation. Characteristics of the generalized form of the distribution of selections are discussed and compared to other methods. It is shown that the distribution of selections corresponds to one of the least effective variations of the magic sets method. It is also shown that both methods have essentially the same power for non-recursive queries. Hence, the magic sets method can be regarded as a natural generalization of pushing selections in relational databases.

Keywords. Deductive databases, Relational databases, Query optimization.

1. Introduction

The combination of *pushing selections* and *selection-first evaluation* is one of the most important methods for efficient query processing in relational databases [25]. Pushing selections transforms queries based on the commutativity of selections with other operators of relational algebra. A generalization of pushing selections called the *distribution of selections* (DS) was proposed by Aho and Ullman for recursive queries [2]. It was the first query transformation (rule rewriting) method designed for the bottom-up evaluation. Several further generalizations were also proposed [1, 10, 14, 15, 18]. Queries can be efficiently processed by selection-first evaluation in the bottom-up computation [2, 3, 6] after transforming queries. However, it is known that DS can optimize only certain queries because selection is commutative only for specific positions of constants in goals even for simple recursive

[1] This work is a part of knowledge base research in the Fifth Generation Computer Systems Project.
This paper is a revised version of the following paper:
Distribution of Selections: The Missing Link between Strategies for Relational Databases and Deductive Databases, *ICOT Technical Report TR-474*, also in *Proc. DOOD*, Kyoto, 1989.
[2] Address: Systems Laboratory, Oki Electric Industry Co., Ltd. 4-11-22 Shibaura Minatoku, Tokyo, 108 Japan Email: miyazaki%okilab.oki.co.jp@uunet.uu.net

queries [5]. Hence, the direct extension of pushing selections is not considered as a part of a general framework of query processing strategy for deductive databases because of its limited applicability.

Meanwhile, many methods have been proposed for recursive query processing and two general frameworks have emerged from them. One is the query/subquery (QSQ) and its variations, which are based on top-down evaluation [24, 28, 29]. QSQ realizes the selection first processing in an integrated way, and does not separate transformation and evaluation phases. The other is the magic sets method and its variations (MS) [4, 8, 17, 20, 21, 23]. MS is designed as a transformation used before the bottom-up computation like DS. Works reported in [10, 13] can also be regarded as methods that combine the concept of MS with other optimization techniques in relational algebraic notation. QSQ and MS are considered as general frameworks because they are effective for a broad class of queries. It is known that there are strong connections between QSQ and MS [9, 23, 26, 29]. However, the relationship of these methods to DS (or pushing selections) is not clear, although MS realizes a more general view of selection propagation [7, 21]. Hence, DS and MS have been treated as independent methods [5, 11]. The situation is illustrated in Figure 1 which shows relationships among methods.

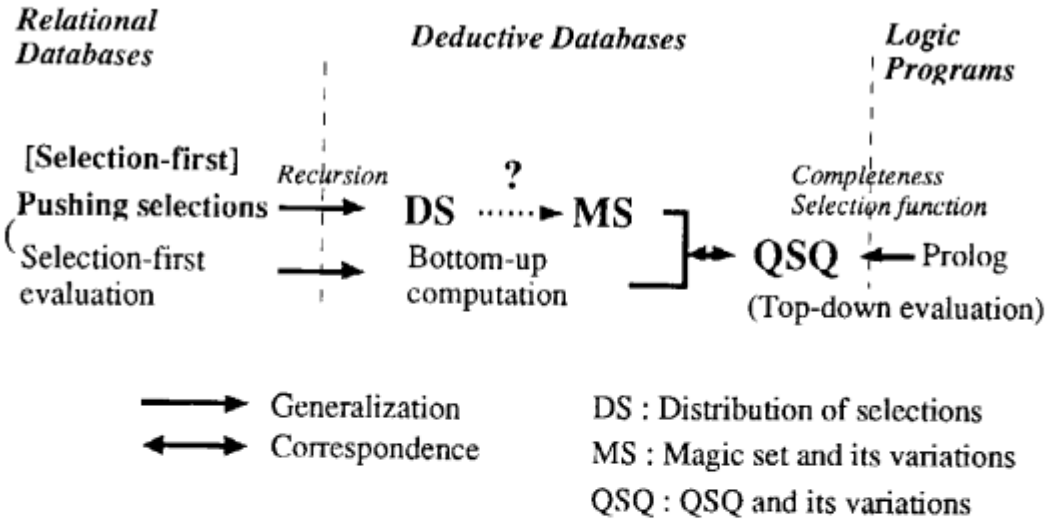

Figure 1 Relationships among Methods

Many methods for recursive query processing rely on relational database techniques to handle large relations. Therefore, it is possible to establish an integrated framework of recursive and

relational query processing strategies, if the recursive strategy is a generalization of the relational strategy. Even if the former is not a generalization of the latter, we can obtain a better criterion to choose methods for particular types of queries by clarifying the difference between methods. Figure 1 indicates that this problem has been almost solved for general purpose methods because MS (or QSQ) is effective for a broader class of queries than DS. However, this problem is not formally solved, because the precise relationship between MS and DS is not known. For instance, results of MS for non-recursive queries look different than the results of DS. Moreover, DS may be more efficient than MS for certain queries, because it does not introduce additional predicates. Thus, the corresponding link is shown with a question mark in the figure. This paper tries to find the missing link and to bridge this gap. The subject involves two issues:

(1) Formalization of DS in clausal form.
(2) Discussion on the relationship between DS and other methods.

Another motivation for this paper is to investigate properties of MS. The relationship between QSQ and MS is not symmetric although their correspondence is established. Because MS usually uses top-down concepts in its formalization and QSQ does not rely on bottom-up concepts, it is difficult to understand MS without top-down concepts but QSQ is self-explanatory. The relationship between QSQ and MS becomes more symmetric, if we can give an alternative explanation of MS by bridging the gap between MS and DS, and make optimization of bottom-up method self-explanatory.

The organization of this paper is as follows. Section 2 summarizes methods for DS. Section 3 discusses DS in clausal form. The relationship between DS and other methods is discussed in section 4. The results are summarized in section 5.

2. From Pushing Selections to the Distribution of Selections

The first generalization of pushing selections for recursive queries was proposed by Aho and Ullman [2]. The name of the method, the *distribution of selections (DS)*, is also used to refer to its various extensions in this paper. The following example shows how DS is performed by the Aho-Ullman method.

Example 1a

←ancestor(X,taro).
ancestor(X,Y)←parent(X,Y)

ancestor(X,Y)←parent(X,Z),ancestor(Z,Y).

It is known that this query can be expressed in relational algebra [25]:

answer = $\sigma_{2=taro}$(ancestor)

ancestor = f(ancestor)

= parent $\cup$ $\pi_{1,4}$(parent $|\times|_{2=1}$ ancestor).

The answer for this query can be obtained by first computing *ancestor* as the least fixpoint (lfp) of *ancestor = f(ancestor)* and then applying the selection $\sigma_{2=taro}$ to it. However, computing the ancestor relation as the lfp is a time consuming matter. The computation becomes more efficient, if the selection is distributed. The distribution is performed as follows. If the selection $\sigma_{2=taro}$ is applied to the fixpoint equation, it becomes:

$\sigma_{2=taro}$(ancestor) = $\sigma_{2=taro}$(parent $\cup$ $\pi_{1,4}$(parent $|\times|_{2=1}$ ancestor))

= $\sigma_{2=taro}$(parent) $\cup$ $\sigma_{2=taro}$($\pi_{1,4}$(parent $|\times|_{2=1}$ ancestor))

= $\sigma_{2=taro}$(parent) $\cup$ $\pi_{1,4}$(parent $|\times|_{2=1}$ $\sigma_{2=taro}$(ancestor)).

Thus, the fixpoint equation becomes the equation for $\sigma_{2=taro}$*(ancestor)* instead of the equation for *ancestor*. Hence, the answer can be obtained without computing the whole *ancestor*.

The Aho-Ullman method can be applied only to simple linear queries. Its generalizations for more general function-free queries were proposed in [1,10,14]. These methods use different notations: "the static filtering" uses graphs [14], "push of selection to variables" uses relational algebraic notation [10] and "moving selection" uses both relational algebraic notation and graphs [1]. They can be considered as variations of the following algorithm in relational algebraic notation. Let us suppose a query is expressed as follows to simplify discussion:

answer = σ_P(r), and r=f(r).

The algorithm transforms the query to the following form:

answer = σ_P(r), and r=σ_F(f(r)), where σ_F is another selection.[1]

It is obvious that the transformed form gives a sound but possibly incomplete answer. Selection σ_F which gives a complete answer can be computed by the following algorithm.

Algorithm 1: (computation of a commutative selection)

i := 0; P_0:=P; F := P_0;

repeat

distribute selection into function f in r=σ_{P_i}(f(r)) to obtain r=f_i($\sigma_{P_{i+1}}$(r));

F := F $\vee$ P_{i+1}; i :=i+1;

until F does not change;

[1] Selection σ_F should actually be distributed into function *f*. We use the above expression for the simplicity of discussion.

Algorithm 1 is obtained by the following observation. We need to compute $\sigma_P(f(r))$ in order to compute $\sigma_P(r)$. Since $\sigma_P(f(r))=f_0(\sigma_{P_1}(r))$, we need to compute $\sigma_{P_1}(r) = \sigma_{P_1}(f(r)) = f_1(\sigma_{P_2}(r))$. Then, we need $\sigma_{P_2}(r) = \sigma_{P_2}(f(r)) = f_2(\sigma_{P_3}(r))$, etc. Thus, the algorithm obtains a sufficiently weak condition, $F = P \vee P_1 \vee P_2 \vee ...$, in order to compute the complete answer. The Aho-Ullman method corresponds to a variation of algorithm 1 which transforms a query only when $\sigma_F=\sigma_P$.

Example 2a

$\leftarrow$p(v,X,Y,Z) /* This means P= (1=v). */

p(X,Y,Z,W)$\leftarrow$a(X,Y,Z,W) /* Relations *a* and *q* are in the database. */

p(X,Y,Z,X) $\leftarrow$ p(W,X,Y,Z),q(X,Z).

This query can be expressed in relational algebra as follows.

answer = $\sigma_{1=v}$(p)

p = a $\cup$ $\pi_{2,3,4,2}$(p $|\times|_{(2=1)\wedge(4=2)}$ q).

The Aho-Ullman method cannot optimize this query. However, the optimization is possible by transforming the query as follows:

answer = $\sigma_{1=v}$(p)

p = σ_F(a $\cup$ $\pi_{2,3,4,2}$(p $|\times|_{(2=1)\wedge(4=2)}$ q))

= σ_F(a) $\cup$ $\pi_{2,3,4,2}$(σ_F(p) $|\times|_{(2=1)\wedge(4=2)}$ q)),

where selection condition F = (1=v) $\vee$(2=v)$\vee$ (3=v) $\vee$(4=v) is computed by algorithm 1. Methods in [1, 10, 14] transform the query in essentially the same way.

An important contribution of extended methods is that the selection expressed as a disjunction of conditions is sometimes commutative with the lfp operator even if the original selection itself is not commutative. These methods assume function-free queries. A generalization of the static filtering for queries with function symbols was proposed in [15]. Although algorithms for DS always terminate for function-free queries, generalized algorithms may not terminate if there are function symbols. The generalized static filtering restricts the complexity of terms in order to guarantee the termination.

There have been some discussions on the relationship between DS and MS [1, 7, 10, 11, 14, 18]. It is known that their performance may be identical for queries such as example 1a [5], but the precise relationship is difficult to recognize because of different notations. The next section discusses DS in clausal form which enables us to compare it with other methods.

3. Distribution of Selections in Clausal Form

3.1. Fundamental Concepts

Before discussing the algorithm for DS in clausal form, fundamental concepts used in this paper are introduced.

Definition (Notations)
clause: A clause means a Horn clause in this paper. Function symbols may be used in clauses.
$\Rightarrow$: Let A be a clause and let B be a set of clauses. $A \Rightarrow B$ means B is a logical consequence of A.
$\supset$: Set inclusion.
iff: if and only if.
body(R): Let R be a set of clauses. *body(R)* is the set of all atoms that appear in bodies of clauses in R.

Definition Let B and C be definite clauses (or atoms). B is *more general* than C iff there exists a substitution θ such that $C = B\theta$. We also say that B *covers* C iff B is more general than C.[1] B is a *variant* of C iff B covers C and C covers B. We treat variants as if they are identical when we discuss a set of clauses (or atoms).

Let us consider examples for the above definition. Clause *ancestor(X,Y)* $\leftarrow$*parent(X,Z), ancestor(Z,Y)* covers *ancestor(X,taro)* $\leftarrow$ *parent(X,Z),ancestor(Z,taro)* because the latter is obtained from the former by substituting *taro* for *Y*. Notice that if B covers C then $\{B\} \Rightarrow C$. We treat {a(X,Y), a(V,W)} and {a(X,Y)} as identical sets, because a(V,W) is a variant of a(X,Y).

Definition A *database* (*DB*) is a finite set of ground unit clauses. A *query* is the set $Q = \{\leftarrow g\} \cup D$, where g is a *goal atom* and D is a set of definite clauses.[2] A predicate that appears in the head of a clause in Q is called a *q-predicate* and a predicate that appears in the database is called a *d-predicate*. Let g' be a ground instance of g, then the *answer* for the query is the set $G = \{g' \in B \mid D \cup DB \Rightarrow g'\}$ where B is the Herbrand base [16] of $D \cup DB$.

---

[1] The concept of *covering* is a special case of *subsuming* [12]. Clause B subsumes clause C if B covers a subclause of C.
[2] Usually, g (or $\leftarrow$g) is called a query and D is called an intensional database. The above definition is used in this paper to simplify discussion. Notice that this definition corresponds to the definition of a query in relational databases.

We assume that a given query is a finite set, although its transformed form may be an infinite set. We also assume that {q-predicate} $\cap$ {d-predicate} = $\varnothing$ to simplify discussion. The underlying domain of queries and databases is assumed to be fixed, i.e., the Herbrand base of every set of clauses is a subset of a given set.

Definition Let Q and Q' be queries, and let G and G' be their answers for a database respectively. $Q \geq Q'$ iff $G \supset G'$ for any database. Q and Q' are *equivalent*, denoted $Q \approx Q'$, iff $Q \geq Q'$ and $Q \leq Q'$. Note that the relation "$\approx$" is an equivalence relation.

Definition A *query transformation* is a mapping from the set of all queries to itself. The *composition* of mappings f and g is defined by $f*g(Q) = g(f(Q))$ for any Q.

Definition Query transformation f is *complete* iff $f(Q) \geq Q$ for any Q. It is *sound* iff $f(Q) \leq Q$ for any Q. It is *equivalent* iff $f(Q) \approx Q$ for any Q.

3.2. Formalization of Distribution of Selections in Clausal Form

DS transforms equation $r=f(r)$ to $r=\sigma(f(r))$ as discussed in section 2. Equation $r=f(r)$ corresponds to a set of clauses of the form $r \leftarrow B$ having a same head predicate, where B is a conjunction of atoms.. There are two clausal forms corresponding to the transformed equation:

(1) $(r \leftarrow B)\theta$, where θ is a substitution of variables.

(2) $r \leftarrow r^*, B$, where r^* is an atom.

The first form leads to DS in clausal form. The second leads to a variation of MS [17], which is used to investigate the relationship between DS and MS in section 4.

It is easy to see that the results of DS can be expressed by substituted clauses. For instance, the result of example 1a is equivalent to the following set of clauses [5, 21]:

ancestor(X,taro)←parent(X,taro)

ancestor(X,taro)←parent(X,Z),ancestor(Z,taro).

Let us consider a way to obtain a set of substituted clauses which is equivalent to the set of original clauses. First, we observe that the above set of clauses can be obtained by applying the most general unifiers (*mgu*) of the goal atom and the heads of original clauses to substitute variables in these clauses. If an atom $(r')\theta$ appears in a body of a substituted clause, this atom is used to obtain another set of substituted clauses. We can repeat this

process until it converges. The process obviously corresponds to a generalization of algorithm 1 in section 2.

We can formulate a transformation procedure based on the above observation. First, we define the set of all possible ground and non-ground instances of definite clauses in Q.

Definition Let $Q = \{\leftarrow g\} \cup D$ be a query. The *extension* of Q is defined as $S(Q) = \{\leftarrow g\} \cup \{C\theta \mid C \in D \wedge \theta \text{ is a substitution}\}$.

S(Q) is equivalent to Q, because each clause in S(Q) is covered by a clause in Q and $S(Q) \supset Q$. Thus, the problem of DS is to find the minimal subset of S(Q) that is equivalent to Q. First, we ignore the minimality problem.

Each step of the substitution in the above discussion can be formally defined as follows.

Definition Let $Q = \{\leftarrow g\} \cup D$ be a query and let R be a subset of S(Q). A mapping F_Q from the power set of S(Q) to itself is defined as follows.

$$F_Q(R) = \{\leftarrow g\} \cup \{C\theta \mid C \in D \wedge \exists r \in body(R) \wedge \theta = mgu(r, head(C))\}.$$

It is obvious that $F_Q(R)$ is a subset of S(Q). The mapping F_Q distributes selection conditions in body(R) to clauses whose heads are unifiable with elements of body(R). We can repeatedly apply it to distribute selections until an equivalent query is obtained.

Example 1b: Let us consider an ancestor query with goal ←*ancestor(jiro,taro)*. We compute a sequence of queries defined by $Q_{i+1} = F_Q(Q_i)$ with $Q_0 = \varnothing$.

$Q_1 = F_Q(\varnothing)$: ←ancestor(jiro,taro)

$Q_2 = F_Q(Q_1)$: The rule in Q_1 and the following clauses are obtained, because *ancestor(jiro,taro)* is in a body.

ancestor(jiro,taro)←parent(jiro,taro)

ancestor(jiro,taro)←parent(jiro,Z),ancestor(Z,taro).

$Q_3 = F_Q(Q_2)$: Rules in Q_2 and the following clauses are obtained, because *ancestor(Z,taro)* is in a body.

ancestor(X,taro)←parent(X,taro)

ancestor(X,taro)←parent(X,Z),ancestor(Z,taro).

$Q_4 = F_Q(Q_3) = Q_3$

It is clear that $Q_{i+1} = Q_i$ for $i > 3$. The result is equivalent to Q.

The process may not terminate if there are function symbols. However, we can prove that F_Q has a unique least fixpoint. The formalization is similar to the fixpoint semantics of logic programs [16, 27].

Lemma 3.1

(a) The power set $L = 2^{S(Q)}$ of the extension of Q is a complete lattice under set inclusion. The bottom ($\perp$) is $\varnothing$, and the top element is S(Q).

(b) F_Q is monotonic, i.e., $F_Q(R1) \supset F_Q(R2)$ for $R1 \supset R2$.

(c) F_Q is continuous, i.e., $F_Q(lub(X)) = lub(F_Q(X))$ for every directed subset X of L. Here, $F_Q(X)$ means $\{F_Q(C) / C \in X\}$. X is directed *if every finite subset of X has an upper bound in X.*

Proof (a) and (b) are obvious. (c) is proved as follows. Let X be a directed subset of L, and let $F'_Q(R) = \{C\theta \mid C \in D \wedge \exists r \in Body(R) \wedge \theta = mgu(r, head(C))\}$. If F'_Q is continuous, so is F_Q because $F_Q(R1) = \{\leftarrow g\} \cup F'_Q(R1)$. Now we have that

$C \in F'_Q(lub(X))$
iff $C = C'\theta \wedge (C' \in D \wedge \exists r \in body(lub(X)) \wedge \theta = mgu(r, head(C'))$
iff for $\exists I \in X$, $C = C'\theta \wedge (C' \in D \wedge \exists r \in body(I) \wedge \theta = mgu(r, head(C'))$
(because X is directed)
iff $C \in F'_Q(I)$ for $\exists I \in X$
iff $C \in lub(F'_Q(X))$. ■

Definition Let L be a complete lattice, and let T be a monotonic mapping from L to L. Then we define

$T\uparrow 0 = \perp$
$T\uparrow\alpha = T(T\uparrow(\alpha-1))$, if α is a successor ordinal
$T\uparrow\alpha = lub(T\uparrow\beta: \beta<\alpha)$, if α is a limit ordinal.

We obtain the next lemma from lemma 3.1. Please refer to proposition 5.4 in [16] for the proof.

Lemma 3.2 *F_Q has a least fixpoint, $lfp(F_Q)$. Furthermore, $lfp(F_Q) = F_Q\uparrow\omega$, where ω is the smallest ordinal next to 0.*

Next, we prove the equivalence of a fixpoint to original query Q.

Lemma 3.3 *Let $Q' \in 2^{S(Q)}$ be a fixpoint of F_Q, i.e., $Q' = F_Q(Q')$. Then $Q' \approx Q$.*

Proof $Q \geq Q'$ is clear because Q' is a subset of $S(Q) \approx Q$. $Q' \geq Q$ can be proved by inspecting SLD tree of Q, because there exists an atom in a body of a clause in Q' that is more general than each subgoal in the tree, and every clause in Q which is unifiable with the subgoal is included in Q' in a substituted form. ■

Theorem 3.4 *(equivalence of $lfp(F_Q)$ and Q)*
(a) $F_Q \uparrow \omega = lfp(F_Q) \approx Q$.
(b) $Q' \approx Q$ *for any* Q' *such that* $S(Q) \supset Q' \supset lfp(F_Q)$.
Proof Obvious from lemmata 3.2 and 3.3 because $Q \approx S(Q)$. ■

Thus, the procedure that computes $lfp(F_Q)$ is an equivalent transformation which distributes selections for general recursive queries. Note that $lfp(F_Q)$ is the limit of $Q_{i+1} = F_Q(Q_i)$ with $Q_0 = \varnothing$, and it is computed by the following simple iterative procedure.

Procedure for the distribution of selections:
$Q' = \varnothing$;
while "Q' changes" $Q' := F_Q(Q')$;

Since this procedure is identical to the *naive evaluation algorithm* [6] except for the mapping used, we can also consider a *semi-naive algorithm.*

Example 2b: (same query as example 2a)
We obtain the following sets of clauses by applying F_Q repeatedly. Non-recursive clauses are not shown.

Q: $\leftarrow p(v,X,Y,Z)$.
$p(X,Y,Z,X) \leftarrow p(W,X,Y,Z),q(X,Z)$.

$Q_1 = F_Q(\varnothing)$: $\leftarrow p(v,X,Y,Z)$.

$Q_2 = F_Q(Q_1)$: The clause in Q_1 and the following clause are obtained.
$p(v,Y,Z,v) \leftarrow p(W,v,Y,Z),q(v,Z)$.

$Q_3 = F_Q(Q_2)$: Clauses in Q_2 and the following clause are obtained.
$p(X,v,Z,X) \leftarrow p(W,X,v,Z),q(X,Z)$.

$Q_4 = F_Q(Q_3)$: Clauses in Q_3 and the following clause are obtained.
$p(X,Y,v,X) \leftarrow p(W,X,Y,v),q(X,v)$.

$Q_{i+1} = F_Q(Q_i)$ for $i > 3$.

This result corresponds to the disjunction of conditions in section 2.

The method corresponds to the straightforward generalization of the method discussed in section 2. However, the result obtained by the procedure may be infinite and the procedure may not terminate, if there are function symbols. The termination can be guaranteed by modifying the procedure in similar ways to those discussed in [15, 18].

The proposed method handles selection conditions embedded in predicates. In relational algebra, conditions are expressed as combinations of "attribute θ value" and "attribute θ attribute". If θ is equality "=", the conditions can be converted to those in the embedded form in clausal notation. For other types of conditions, we have to modify the definition of F_Q in order to distribute selections. If there are such conditions in bodies of clauses, we can modify F_Q to attach these conditions to substituted clauses along with conditions expressed as the mgu.

3.3. Optimization of the Transformed Result

The procedure in the previous section can distribute selections for general recursive queries, but the result, $lfp(F_Q)$, often contains redundant clauses. For instance, clauses generated in the second iteration are redundant in example 1b. Hence, we have to consider certain optimization procedure. We restrict our attention to a special case of redundancy elimination because the problem is difficult in general even for function-free queries [22].

Definition A query $Q = \{\leftarrow g\} \cup D$ *covers* another query $Q' = \{\leftarrow g\} \cup D'$ iff each element of D' is covered by a clause in D. We assume that ∅ is covered by any query. A subset Q' of Q is called a *minimum cover* of Q iff Q' covers Q and there is no proper subset of Q' that covers Q. Obviously there exists a unique minimum cover for any Q, and the covering problem is decidable for any finite Q. The function that maps a query Q to its minimum cover, *mcv(Q)*, can be regarded as a mapping from the set of all queries to itself. A query Q is *minimum* iff Q = mcv(Q).[1]

Note that D covers D' if $D \supset D'$. Hence, Q and mcv(Q) cover each other. We define a partial order in the set of minimum queries.

Definition A subset *MS(Q)* of $2^{S(Q)}$ for a given Q is defined by $MS(Q) = \{Q' | Q' \in 2^{S(Q)} \wedge Q' = mcv(Q') \wedge (\leftarrow g \in Q' \vee Q' = \emptyset)\}$. A relation $\leq_s$ in MS(Q) is defined as follows: $Q_1 \leq_s Q_2$ iff Q_2 covers Q_1 for $Q_1, Q_2 \in MS(Q)$.

[1] We can consider more general minimality based on subsumption. We discuss the optimization based on covering, because it corresponds to simplification of selection conditions in relational algebra.

Lemma 3.5

(a) $\leq_s$ is a partial order, and MS(Q) is a complete lattice under $\leq_s$. Let X be a subset of MS(Q). The lowest upper bound of X , $lub_s(X)$, in MS(Q) is identical to mcv(lub(X)), and the greatest lower bound, $glb_s(X)$, is identical to mcv(glb(X)). The top is mcv(S(Q)) = mcv(Q) and the bottom is $\varnothing$.

*(b) F_Q*mcv is monotonic in MS(Q).*

*(c) F_Q*mcv is continuous in MS(Q).*

*(d) There exists a least fixpoint of F_Q*mcv in MS(Q), and lfp(F_Q*mcv) = F_Q*mcv$\uparrow\omega$.*

The proof is shown in appendix.

Lemma 3.6 *Let Q' be a fixpoint of F_Q*mcv. Then, Q' ≈ Q.*

Proof Same as the proof of lemma 3.3. ■

Theorem 3.7 *(equivalence of lfp(F_Q*mcv) and Q)*

*(a) F_Q*mcv$\uparrow\omega$ = lfp(F_Q*mcv) = mcv(lfp(F_Q))*

*(b) Q' ≈ Q for any Q' such that S(Q) ⊃ Q' ⊃ lfp(F_Q*mcv).*

The proof is shown in appendix. The theorem implies the elimination of redundant clauses by covering can be performed any time during the transformation. The least fixpoint, lfp(F_Q*mcv), can be computed by the same procedure as the one given in section 3.2, if we replace mapping F_Q by F_Q*mcv. The modified procedure can be regarded as an improved procedure for DS.

Example 1c: (same query as example 1b)

Clauses generated in the first and second steps are identical to those in example 1b.

Q_3 = mcv(F_Q(Q_2)). F_Q(Q_2) consists of the following clauses.

```
←ancestor(jiro,taro)
ancestor(jiro,taro)←parent(jiro,taro)          /* redundant */
ancestor(jiro,taro)←parent(jiro,Z),ancestor(Z,taro)    /* redundant */
ancestor(X,taro)←parent(X,taro)
ancestor(X,taro)←parent(X,Z),ancestor(Z,taro).
```

Hence, Q_3 is given by

```
←ancestor(jiro,taro)
ancestor(X,taro)←parent(X,taro)
ancestor(X,taro)←parent(X,Z),ancestor(Z,taro).
```

$Q_{i+1} = Q_i$ for $i > 2$.

Notice that this optimization corresponds to the simplification of selection conditions in relational algebra. For instance, F = ((1=jiro ∧ 2=taro) ∨ 2=taro) = (2=taro).

4. From the Distribution of Selections to Magic Sets

4.1. The Distribution of Selections and Top-Down Methods

We compare the effectiveness of DS to top-down methods in this section. The following theorem is a direct consequence of the equivalence of DS.

Theorem 4.1 *(the distribution of selections is not more effective than a top-down method)*

Let $Q = \{\leftarrow g\} \cup D$, and DB be a database. Let $Q' = lfp(F_Q) = \{\leftarrow g\} \cup D'$, let $Q'' = lfp(F_Q{*}mcv) = \{\leftarrow g\} \cup D''$, let S_Ans be the set of all answers for subgoals in a SLD tree of $Q \cup DB$, and let model(R) denote the least Herbrand model of R. Then

$model(D \cup DB) \supset model(D' \cup DB) = model(D'' \cup DB) \supset S\_Ans$, for any Q and DB.

Proof $model(D \cup DB) \supset model(D' \cup DB)$ is obvious, because Q' is covered by Q. $model(D' \cup DB) = model(D'' \cup DB)$, because $lfp(F_Q{*}mcv) = mcv(lfp(F_Q))$ by theorem 3.7. $model(D' \cup DB) \supset S\_Ans$ is proved in the same manner as lemma 3.3. ■

This theorem implies that DS is not more effective than a top-down method based on SLD resolution in restricting computational space. Hence, it guarantees that a top-down method based on SLD resolution is not less effective than the combination of DS and the bottom-up computation provided selection-first is properly performed in the top-down method. QSQ can be more effective than (or as effective as) DS because it uses the *selection function* (computation rule in SLD resolution) to realize selection-first evaluation [28]. According to the correspondence between QSQ and MS [9, 17, 23, 26, 29], the theorem also implies that MS can be more effective than (or as effective as) DS if we ignore the overhead of magic predicates.

The theorem can be explained as follows. In a top-down method, there are two modes of information passing: *unification* and *sideways information passing* (sip) [8]. The distribution of selection performs information passing *only* by unification. Because a top-down method performs both unification and sip, it is more effective than DS.

Let us consider the ancestor example again. The goal is ←*ancestor(X,taro)* and we assume *parent* is defined in terms of *father* and *mother*. The transformed query by DS is:

←ancestor(X,taro)
ancestor(X,taro)←parent(X,taro)
ancestor(X,taro)←parent(X,Z),ancestor(Z,taro)
parent(X,Y) ← father(X,Y)
parent(X,Y) ← mother(X,Y).

The selection cannot be distributed for *parent*. Thus, DS is not effective even for certain non-recursive q-predicates. However, a top-down method with an appropriate computation rule (and MS) can effectively process this query. Note that we need the right-to-left evaluation strategy, and a Prolog-like top-down evaluation is not efficient or does not terminate for this query.

4.2. The Distribution of Selections and Magic Sets

First, we discuss the reason of the ineffectiveness of DS. The following descriptions summarize algorithms for DS.

(1) *Pushing selections* in relational databases: Pushing selections is performed using the commutativity of selections with other relational operators.

(2) *Aho-Ullman*: This method is based on the commutativity of given selections with the lfp operator.

(3) *Generalized distribution of selections* : For more complex queries, DS can be generalized by introducing *disjunctions of conditions*.

A common characteristic of algorithms for DS is that they extract selection conditions *only* from queries. On the other hand, top-down methods propagate binding conditions extracted from databases as well as from queries. This is considered the principal reason why top-down methods are more effective than DS as shown in theorem 4.1. Hence, DS may be improved if we can extract conditions not only from queries but also from databases. However, extracting conditions from a database before the bottom-up computation is difficult and time consuming. Moreover, the equivalency of the transformed result becomes dependent on the database, even if we can extract conditions from the database. A possible solution to this problem is to express conditions by definite clauses, and dynamically evaluate conditions during the bottom-up computation. As discussed at the beginning of section 3.2, there are two clausal forms to express the transformed equation $r=\sigma(f(r))$: $(r\leftarrow B)\theta$ and $r\leftarrow r^*,B$. The second form is more general than the first form because every selection in the first form can also be expressed in the second form. For instance, a clause $(r\leftarrow B)\theta$ can be expressed by a pair of clauses, $r\leftarrow r^*,B$ and $r^*\theta$. Moreover, we can express selection conditions by clauses having bodies. Thus, we can dynamically compute selection conditions

during the bottom-up computation by expressing conditions by atoms and clauses instead of substitutions. We proposed a transformation based on this idea, which was essentially a variation of MS [17].

The above discussion informally shows that MS can be regarded as a generalization of DS. If this observation is correct, there should be a precise correspondence between MS and DS. There are two possible ways for such a correspondence. First, MS may be as effective as DS for non-recursive queries for which DS is known to be effective. In other words, they may be essentially the same methods for relational databases. Second, DS may correspond to a special case of MS and there may exist a transformation that has the following properties.

(1) It is a variation of MS that extracts conditions only from queries, i.e., its magic sets can be computed without using the contents of databases.

(2) Its effect is the same as DS except for the existence of additional predicates and clauses.

We show that the above two conjectures are in fact true. First, we show that there exists a transformation that has above properties. Let us consider the ancestor query with the goal *←ancestor(jiro,X)*.

MS generates the following set of clauses as the transformed result [4, 8]. Here, "*" indicates a magic predicate.

Goal: $\leftarrow ancestor^{bf}(jiro,X)$

Modified rules: $ancestor^{bf}(X,Y) \leftarrow ancestor^{*bf}(X),parent(X,Y)$

$ancestor^{bf}(X,Y) \leftarrow ancestor^{*bf}(X),parent(X,Z),ancestor^{bf}(Z,Y)$

Seed: $ancestor^{*bf}(jiro)$

Magic rule: $ancestor^{*bf}(Z) \leftarrow ancestor^{*bf}(X),parent(X,Z).$

If the adornments of predicates are eliminated and implicit arguments of magic predicates corresponding to *f* are explicitly shown, the result can be rewritten as follows:

Goal: $\leftarrow ancestor(jiro,X)$

Modified rules: $ancestor(X,Y) \leftarrow ancestor^{*}(X,Y),parent(X,Y)$

$ancestor(X,Y) \leftarrow ancestor^{*}(X,Y),parent(X,Z),ancestor(Z,Y)$

Seed: $ancestor^{*}(jiro,X).$

Magic rule: $ancestor^{*}(Z,Y) \leftarrow ancestor^{*}(X,Y),parent(X,Z).$

Although this result is not bottom-up evaluable [5], it has an interesting property. By changing only its seed, the resulting set of clauses becomes equivalent to the original query for any types of goals. For instance, it is equivalent to the original query for goal

←*ancestor(X,taro)* if the seed is changed to *ancestor*(X,taro)*. It is also equivalent for goal ←*ancestor(jiro,taro)* if the seed is changed to *ancestor*(jiro,taro)*. Note that if the type of binding in the goal is changed then all clauses have to be changed in the usual magic sets method.

Now suppose that *parent(X,Z)* in the body of the magic rule is eliminated. Then the magic rule becomes:

ancestor*(Z,Y) ← ancestor*(X,Y).

This result is still equivalent to the original query, because the form of modified rules guarantees the soundness of the transformation. However, the resulting set of clauses has clearly a weaker effect than the original result in reducing the computation space because the binding propagated by *parent(Z,Y)* cannot be propagated. Moreover, the magic sets do not depend on the database and the effect is exactly the same as that of DS. For instance, binding on the first argument vanishes for goal ←*ancestor(jiro,X)* because the first argument of the head of the magic rule does not appear in the body. If the goal is ←*ancestor(X,taro)*, then the binding is propagated by the magic rule. If the goal is ←*ancestor(jiro,taro)*, then only the binding on the second argument is propagated. Thus, the above transformation has the same effect as DS for the ancestor query.

Let us define a variation of MS by generalizing the above example. We first summarize the basic concepts of the Horn clause transformation by restrictor (HCT/R) proposed in [17]. HCT/R is a transformation that maps a clause $r \leftarrow B$ to clause(s) of the form $r \leftarrow r^*, B$, where r^* is called a *restrictor*. The restrictor corresponds to the magic atom in the usual magic sets method. The transformation gives an equivalent query if clauses for restrictors are defined properly. The conditions for equivalence are found in [17]. There are two versions of HCT/R, the method that uses (full) restrictors and the method that uses partial restrictors. The full restrictor predicate has the same arity as the predicate in the head of a modified rule. The partial restrictor has smaller arity with proper adornment like the magic sets method. Ways to obtain semi-optimal results are also discussed in [17]. The framework of the partial restrictor version is not less general than those of any other variations of MS [4, 8, 20, 21, 23], because they can be formulated in the framework of HCT/R. A special case of the full restrictor version corresponds to DS.

Definition A transformation called *static HCT/R* is defined as follows:
Let $Q = \{\leftarrow g\} \cup D$ be a query. The transformed query Q' = st_hctr(Q) is obtained by the following steps. It does not change the goal.

(1) Let C = $r \leftarrow r_1,...,r_n$ be a clause in D. Replace each C by the following modified rule:
$r \leftarrow r^*,r_1,...,r_n$, where r^* is an atom having a new predicate symbol corresponding to r and the same arguments as the head atom.
(2) Add a seed which is a unit clause g^* having the same argument as the goal atom.
(3) For each $r \leftarrow r^*,r_1,...,r_n$, generate the following magic rules and add them:
For each i, generate a clause, $r^*_i \leftarrow r^*$, if the predicate of r_i is a q-predicate.

The effectiveness of MS depends on two factors: (1) numbers of atoms in bodies of magic rules and (2) arities of magic predicates. Static HCT/R is weakest in terms of the first factor among variations of MS. Its magic sets do not depend on databases because magic rules contain only restrictors. Thus, it is a variation of MS that extracts conditions only from queries. The equivalency of static HCT/R can easily be shown by the theorem for the equivalency of the transformation in [17] or can be directly proved by comparing SLD-trees of the original and the transformed queries.

We need several definitions to show the correspondence between DS and static HCT/R. The following is the definition of the mapping used to define the fixpoint semantics of logic programs [16, 27].

Definition Let D be a set of definite clauses. Let B be the Herbrand base of D, let 2^B be the power set of B, and let I be an element of 2^B. A mapping T_D from 2^B to 2^B is defined as follows.

$T_D(I) = \{r \in B \mid r \leftarrow r_1,r_2,...,r_n$ is a ground instance of a clause in D
and $r_1,r_2,...,r_n$ are elements of I$\}$.

We also need a generalized mapping that produces nonground instances.

Definition Let D be a set of definite clauses. Let U be the set of all unit clauses (that are possible in the underlying language). Let 2^U be the power set of U, and let I be an element of 2^U. A mapping T^+_D from 2^U to 2^U is defined as follows.

$T^+_D(I) = \{r' \in U \mid (r'$ is a unit clause in D$) \vee$
$(r \leftarrow r_1,r_2,...,r_n \in D \wedge \exists p_1,p_2,...,p_n \in I \wedge$
$\theta_i = mgu(r_i,p_i) \wedge \theta = \theta_1\theta_2,...,\theta_n \wedge r'=r\theta)\}$.

T^+_D is a generalization of T_D. The bottom-up computation algorithms should be extended, if we intend to evaluate queries which are not range-restricted [17, 20]. T^+_D can be used to define semantics of such algorithms. The following lemma shows the property of T^+_D.

Lemma 4.2

(a) 2^U is a complete lattice under set inclusion. The bottom is $\varnothing$ and the top is U.

(b) T^+_D is monotonic.

(c) T^+_D is continuous.

(d) There exists a least fixpoint of T^+_D, and $lfp(T^+_D) = T^+_D\uparrow\omega$.

Proof (a) and (b) are obvious. (c) is proved in the same manner as proposition 6.3 in [16]. (d) is a consequence of (a), (b) and (c). ■

The following lemma shows the correspondence between T^+_D and T_D. Hence, we can consider bottom-up computation using T^+_D instead of T_D.

Lemma 4.3 *Let I be an element of 2^U, and let ground(I) be the set of ground instances of elements of I.*

(a) $ground(T^+_D(I)) = T_D(ground(I))$ for any I.

(b) $ground(lfp(T^+_D)) = lfp(T_D)$.

Proof (a) is obvious. (b) can be proved using (a) as follows.

(I) $ground(T^+_D\uparrow 0) = T_D\uparrow 0 = \varnothing$.

(II) Assume $ground(T^+_D\uparrow i) = T_D\uparrow i$ for i=j. Then

$ground(T^+_D\uparrow j+1)=ground(T^+_D(T^+_D\uparrow j)) = T_D(ground(T^+_D\uparrow j)) = T_D(T_D\uparrow j) = T_D\uparrow j+1$.

Therefore, $ground(T^+_D\uparrow i) = T_D\uparrow i$ for $i<\omega$ by mathematical induction.

Hence, $ground(lfp(T^+_D)) = ground(T^+_D\uparrow\omega) = ground(lub(T^+_D\uparrow\beta: \beta<\omega))$

$= lub(ground(T^+_D\uparrow\beta): \beta<\omega) = lub(T_D\uparrow\beta: \beta<\omega) = T_D\uparrow\omega = lfp(T_D)$. ■

Now, we can prove the second conjecture given at the beginning of this section. The following theorem shows the precise correspondence between DS and static HCT/R.

Theorem 4.4 *(correspondence between the distribution of selections and static HCT/R) Let $Q = \{\leftarrow g\} \cup D$, and let $Q' = st\_hctr(Q)$. Q' can be divided to $Q' = \{g\} \cup D' \cup D^*$ where D' is the set of clauses obtained in step 1 of st_hctr and D^* is the set of clauses obtained in steps 2 and 3. Let DB be a database, let $D1 = (lfp(F_Q) - \{\leftarrow g\}) \cup DB$, and let $D2 = D' \cup D^* \cup DB$.*

(a) There exists a function f such that DS = st_hctr * f:

Let $D^{\#} = \{r' \mid \exists(r \leftarrow r^*, r_1, r_2, \ldots, r_n) \in D' \wedge \exists p^* \in lfp(T^+_{D^*}) \wedge$

θ *is* $mgu(r^*, p^*) \wedge r' = (r \leftarrow r_1, r_2, \ldots, r_n)\theta)\}$.

Then for any Q, $lfp(F_Q) = \{\leftarrow g\} \cup D^{\#}$.

(b) Equivalence of computational costs:

Let us consider bottom-up evaluation by T^+_D. There is a one to one correspondence between the executions of the query processing using DS and static HCT/R. More precisely, there are following correspondences:

Query processing using DS consists of two phases: transformation (DS-trans) and bottom-up evaluation (DS-exec). Query processing using static HCT/R also consists of two phases: transformation (ST-trans) and evaluation (ST-exec). Because restrictors do not depend on q-predicates, ST-exec can be executed in two sub-phases: evaluation of restrictors (ST-exec1) and evaluation of other predicates (ST-exec2). There is a one-to-one correspondence between the steps of DS-trans and ST-exec1, and between steps of DS-exec and ST-exec2.

(c) Equivalence of models:

Let us denote as model(D) instead of as $lfp(T_D)$ to simplify notation.

Then model(D1) = model(D2) − model(D), for any Q and DB.*

The proof is shown in appendix. Note that ST-trans does not have a corresponding phase in DS, but the computational cost of this phase is relatively small.

The first conjecture can be proved by theorem 4.4. The following corollary shows that MS is as effective as DS for non-recursive queries.

Corollary 4.5 *(equivalence of the distribution of selections and the magic sets method for non-recursive queries)*

Let Q be a query. Suppose that Q is non-recursive and there is only one q-predicate in Q. Note that any non-recursive query can be easily transformed to the one with one q-predicate. If MS generates a seed having all arguments that represent binding in the goal, then it is as effective as DS. Moreover, computations using two methods proceed in the same manner.

Proof There are no magic rules other than the seed generated by any variation of MS. If the seed preserves all bindings in the goal, then MS is as effective as static HCT/R. Thus, MS is as effective as DS by theorem 4.4. ■

Note that a variation of MS may eliminate arguments that represent bindings. For instance, earlier methods such as magic sets [4, 21] and generalized magic sets [8] ignore binding of the type $\leftarrow p(X,X)$. These methods may be less effective than DS for non-recursive queries. The corollary holds for more general variations of MS such as magic templates [20] and HCT/R [17].

We have proved two conjectures given at the beginning of this section. Hence, we can conclude that MS is a natural generalization of pushing selections in relational databases.

MS is usually more effective than static HCT/R because MS usually has more atoms in bodies of magic rules than static HCT/R. Thus, it is usually more effective than DS by theorem 4.4. When DS is performing most effective, it is as effective as MS. But, it has its own advantages:

(1) Because it does not introduce additional predicates, the evaluation of its result can be more efficient than MS when it is most effective. We observed about twenty percent difference in query processing time between these two methods in our experimental system [18].

(2) The arities of predicates can be reduced after the transformation. For instance, the result of example 1a can be rewritten as follows by eliminating the second argument:

$\leftarrow anc\_taro(X)$

$anc\_taro(X) \leftarrow parent(X,taro)$

$anc\_taro(X) \leftarrow parent(X,Z), anc\_taro(Z).$

Reducing arities of predicates improves the performance because it simplifies the computation of individual steps although it does not reduce the complexity of the whole processing [5].

Thus, the distribution of selection is more efficient than MS for certain queries, although the difference is not very large. We can improve MS using the above correspondence. The following are the possible but not exhaustive ways for the improvement:

(1) Elimination of magic predicates after the transformation.

(2) Application of DS before MS. The arities of predicates can also be reduced. The effectiveness of DS is difficult to check before the transformation, but the effectiveness of static HCT/R is easier to check using its partial restrictor (i.e., adorned) version.

There are other methods related to DS. First of these is left linear transformation, which is also a generalization of the Aho-Ullman method [19]. It is easy to see that DS is effective for left linear recursions and is a generalization of the left linear transformation. Another is the counting method [4] and its generalization. This method frequently fails to terminate where DS is effective, because the same subgoals repeatedly appear (see also theorem 10.3 in [8]). The simplest example is the ancestor query in example 1a.

5. Conclusion

This paper discussed the generalization of the selection-first principle to recursive queries. A generalized form of DS was proposed, and its relationships to other methods were discussed. We have shown that QSQ and MS are not less effective than DS as far as the selection propagation is concerned. We have also shown that DS corresponds to one of the least effective variations of MS called static HCT/R, and that MS is as effective as DS for non-recursive queries. Thus, MS can be considered as a generalization of DS. As a result, the relationship between general frameworks of query processing strategies for relational databases and deductive databases has become clearer. The revised relationships among methods are illustrated in Figure 2. The generalization of our result for stratified queries is possible because both DS and static HCT/R produce stratified results for a stratified query.

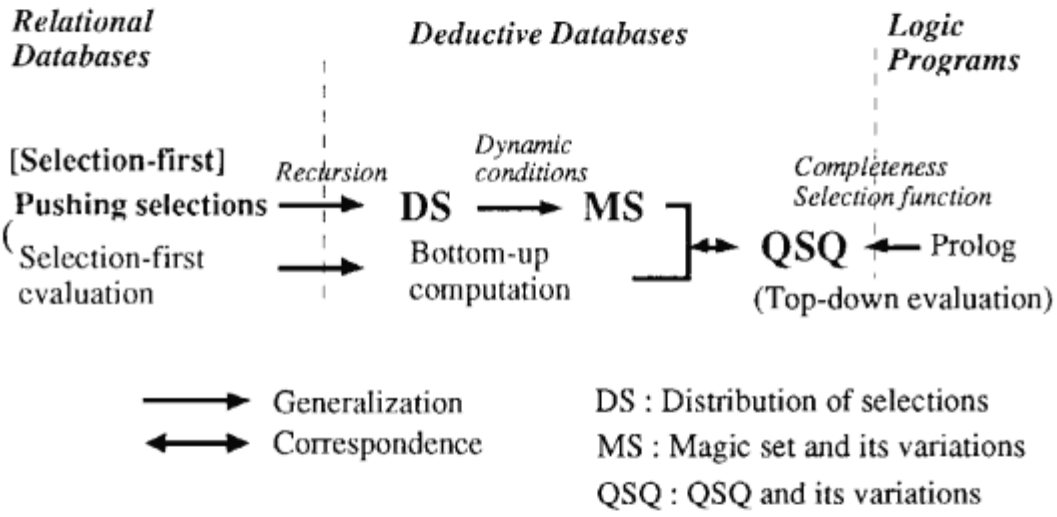

Figure 2 Revised Relationships among Methods

Acknowledgement

I would like to thank members of KBMS PHI group at ICOT and Oki for the development of HCTs. Special thanks are due to Catriel Beeri and Hirohisa Seki for valuable discussions on the frameworks of query processing, and to Kazumasa Yokota and Shojiro Nishio for helpful suggestions. I would also like to thank referees of DOOD89 for their valuable comments.

Appendix Proofs of lemma 3.5, theorem 3.7 and theorem 4.4

Proof of lemma 3.5

(a) It is obvious that $Q1 \leq_s Q1$ for all $Q1 \in MS(Q)$. $Q1 \leq_s Q2$ and $Q2 \leq_s Q1$ imply $Q1 = Q2$, because minimum cover is unique. $Q1 \leq_s Q2$ and $Q2 \leq_s Q3$ imply $Q1 \leq_s Q3$. Therefore, $\leq_s$ is

a partial order. It is easy to see that mcv(lub(X)) gives the lowest upper bound of X and mcv(glb(X)) gives the greatest lower bound.

(b) Let us assume $Q2 \leq_s Q1$ for $Q1, Q2 \in MS(Q)$. Then,

$S(Q1) \supset S(Q2)$

implies $F_Q(S(Q1)) \supset F_Q(S(Q2))$ because F_Q is monotonic

implies $F_Q(Q1)$ covers $F_Q(Q2)$

implies $mcv(F_Q(Q1))$ covers $mcv(F_Q(Q2))$

implies $mcv(F_Q(Q2)) \leq_s mcv(F_Q(Q1))$.

(c) Let X be a directed subset of MS(Q). Let $Y = \{Q1 | \exists Q2 \in X \wedge Q1 = S(Q2)\}$. Then, Y is a directed subset of $2^{S(Q)}$. F_Q*mcv is continuous because

$$F_Q{*}mcv(lub_s(X)) = mcv(F_Q(mcv(lub(X))))$$
$$= mcv(F_Q(lub(X))) \text{ and}$$
$$lub_s(F_Q{*}mcv(X)) = mcv(lub(F_Q{*}mcv(X)))$$
$$= mcv(lub(mcv(F_Q(X))))$$
$$= mcv(lub(F_Q(X)))$$
$$= mcv(lub(F_Q(Y)))$$
$$= mcv(F_Q(lub(Y))) \quad \text{(because } F_Q \text{ is continuous in } 2^{S(Q)})$$
$$= mcv(F_Q(lub(X))).$$

(d) This is a consequence of (a), (b) and (c). ■

Proof of theorem 3.7

The left hand side of (a) is given in lemma 3.5, and (b) is obvious by lemma 3.6. The proof of the remaining part of (a) is shown in the following two steps.

(I) $F_Q{*}mcv\uparrow\alpha = mcv(F_Q\uparrow\alpha)$ for $\alpha < \omega$. This is proved as follows.

(1) $F_Q{*}mcv\uparrow 0 = mcv(F_Q\uparrow 0) = \varnothing$ for $\alpha = 0$.

(2) Assume $F_Q{*}mcv\uparrow\alpha = mcv(F_Q\uparrow\alpha)$ for $\alpha = i$.

$F_Q{*}mcv\uparrow i+1 = F_Q{*}mcv(F_Q{*}mcv\uparrow i) = F_Q{*}mcv(mcv(F_Q\uparrow i)) =$

$mcv(F_Q(mcv(F_Q\uparrow i))) = mcv(F_Q(F_Q\uparrow i)) = mcv(F_Q\uparrow i+1)$.

Hence, $F_Q{*}mcv\uparrow\alpha = mcv(F_Q\uparrow\alpha)$ for $\alpha < \omega$ from (1) and (2) by mathematical induction.

(II) $$F_Q{*}mcv\uparrow\omega = lub_s\{F_Q{*}mcv\uparrow\alpha : \alpha < \omega\}$$
$$= lub_s\{mcv(F_Q\uparrow\alpha) : \alpha < \omega\}$$
$$= mcv(lub\{mcv(F_Q\uparrow\alpha) : \alpha < \omega\})$$
$$= mcv(lub\{F_Q\uparrow\alpha : \alpha < \omega\})$$
$$= mcv(F_Q\uparrow\omega)$$

$= mcv(lfp(F_Q))$ ■

Proof of theorem 4.4

(a) We note that step 3 of st_hctr generates following set of clauses.

Original clause: $r \leftarrow r_1,...,r_n$

Magic rules: $r_1^* \leftarrow r^*$

$r_2^* \leftarrow r^*$

:

$r_n^* \leftarrow r^*$ (except for non-q-predicates).

(I) First we prove the following property. Let body"(R) be the subset of body(R) that corresponds to predicates which also appear in the heads of clauses in R. Then, $lfp(T^+_{D^*}) =$ body"$(lfp(F_Q))$ except for the difference in corresponding predicate symbols. We prove this by showing $T^+_{D^*}\uparrow i =$ body"$(F_Q\uparrow i)$ for $i<\omega$ by mathematical induction. Because both lfps correspond to ω, the property holds if this equation holds.

(1) $T^+_{D^*}\uparrow 0 = \varnothing$.

$F_Q\uparrow 0 = \varnothing$.

$T^+_{D^*}\uparrow 1 = \{g^*\}$.

$F_Q\uparrow 1 = \{\leftarrow g\}$.

Thus, $T^+_{D^*}\uparrow 1 =$ body"$(F_Q\uparrow 1)$ if we ignore "$*$" attached to restrictor.

(2) Assume $T^+_{D^*}\uparrow i =$ body"$(F_Q\uparrow i)$ for $i=j$.

$$T^+_{D^*}\uparrow j+1 = T^+_{D^*}(T^+_{D^*}\uparrow j) =$$
$$\{r' \in U \mid (r' \text{ is a unit clause in } D^*) \vee (r^* \leftarrow q^* \in D^* \wedge \exists p^* \in T^+_{D^*}\uparrow j \wedge \theta = mgu(q^*,p^*) \wedge r'=r^*\theta)\}$$
$$= \{g^*\} \cup \{r' \in U \mid r^* \leftarrow q^* \in D^* \wedge \exists p^* \in T^+_{D^*}\uparrow j \wedge \theta = mgu(q^*,p^*) \wedge r'=r^*\theta)\}$$
$$= \{g^*\} \cup \{r' \in U \mid r^* \leftarrow q^* \in D^* \wedge \exists p^* \in \text{body"}(F_Q\uparrow i) \wedge \theta = mgu(q^*,p^*) \wedge r'=r^*\theta)\}.$$

$$F_Q\uparrow i+1 = F_Q(F_Q\uparrow i)$$
$$= \{\leftarrow g\} \cup \{C\theta \mid C \in D \wedge \exists r \in body(F_Q\uparrow i) \wedge \theta = mgu(r, head(C))\}.$$

Thus, $T^+_{D^*}\uparrow j+1 =$ body"$(F_Q\uparrow i+1)$.

(II) Next, we prove $lfp(F_Q) = \{\leftarrow g\} \cup D^{\#}$.

$$D^{\#} = \{r' \mid r \leftarrow r^*,r_1,r_2,...,r_n \in D' \wedge \exists p^* \in lfp(T^+_{D^*}) \wedge \theta \text{ is } mgu(r^*,p^*) \wedge r' = (r \leftarrow r_1,r_2,...,r_n)\theta)\}$$
$$= \{r' \mid r \leftarrow r^*,r_1,r_2,...,r_n \in D' \wedge \exists p^* \in \text{body"}(lfp(F_Q)) \wedge$$

θ is mgu(r^*,p^*) $\wedge$ $r' = (r \leftarrow r_1,r_2,...,r_n)\theta)\}$

$= \{r' \mid r \leftarrow r_1,r_2,...,r_n \in D \wedge \exists p \in$ body"(lfp(F_Q))$\wedge$

θ is mgu(r,p) $\wedge$ $r' = (r \leftarrow r_1,r_2,...,r_n)\theta)\}$

Therefore, lfp(F_Q) = $\{\leftarrow g\} \cup D^{\#}$.

(b) The correspondence between DS-trans and ST-exec1 is obvious by the proof of (a). The correspondence between DS-exec and ST-exec2 is proved as follows:

DS-exec starts from D2. ST-exec2 starts from D3 = D1 $\cup$ lfp($T^+_{D^*}$). Here, note that lfp($T^+_{D^*}$) is a set of unit clauses and it does not change in ST-exec2. By (a), we can show that each modified rule in D3 produces the same result as the corresponding clause in D2 in each iteration step during the bottom-up evaluation as shown below.

Let $r \leftarrow r^*$, B be the modified rule generated from $r \leftarrow B$. Let D_r be the subset of lfp($T^+_{D^*}$) that corresponds to predicate of atom r. Let $D2_r$ be the set of clauses that are generated from $r \leftarrow B$. Then, $r \leftarrow r^*$,B and elements of $D2_r$ produce identical sets of unit clauses in each step.

(I) Both sets are empty at the beginning of ST-exec2 and DS-exec.

Suppose a unit cause r1 is produced in ST-exec2. There exists a unit clause $r1^* \in D_r$ which is more general than r1. Let θ be mgu(r^*,$r1^*$). Then there exists a clause $(r \leftarrow B)\theta$ in $D2_r$ (see (a) above), and it also produces r1. The reverse is also easy to prove.

(II) Suppose both sets are identical at the i-th iteration.

The corresponding clauses shown in (I) produce identical results at the i+1th step. Therefore, the latter part of (b) can be proved by mathematical induction.

(c) model(D2) = model($D' \cup D^* \cup$ DB)

= model($D' \cup$ lfp($T^+_{D^*}$) $\cup$ DB).

model(D1) = model((lfp(F_Q) $-$ $\{\leftarrow g\}$)$\cup$ DB).

It is obvious that bottom-up computations of both clause sets using T^+_D produce identical results for q-predicates as shown in the proof of (b). Therefore, *model(D2) = model(D1) $\cup$ model(D*)* by lemma 4.3. ■

References

[1] R. Agrawal and P. Devanbu, Moving Selections into Linear Least Fixpoint Queries, *Proc. 4th Intl. Conf. on Data Engineering* (1988) 452-461.

[2] A.V. Aho and J.D. Ullman, Universality of Data Retrieval Languages, *Proc. 6th ACM Symp. on Principles of Programming Languages* (1979) 110-120.

[3] I. Balbin and K. Ramamohanarao, A Generalization of the Differential Approach to Recursive Query Evaluation, *J. Logic Programming*, Vol. 4, No.2 (1987) 259-262.

[4] F. Bancilhon, D. Maier, Y. Sagiv and J.D. Ullman, Magic Sets and Other Strange Ways to Implement Logic Programs, *Proc. of 5th ACM PODS* (1986) 1-15.

[5] F. Bancilhon and R. Ramakrishnan, An Amateur's Introduction to Recursive Query Processing Strategies, *Proc. ACM SIGMOD* (1986) 16-52.

[6] F. Bancilhon, Naive Evaluation of Recursively Defined Relations, in: M.L. Brodie and J. Mylopoulos (Eds.), *On Knowledge Base Management Systems* (Springer-Verlag, 1986) 165-178.

[7] C. Beeri, P. Kanellakis, F. Bancilhon and R. Ramakrishnan, Bounds on the Propagation of Selection into Logic Programs, *Proc. 6th ACM PODS* (1987) 214-226.

[8] C. Beeri and R. Ramakrishnan, On the Power of Magic, *Proc. 6th ACM PODS* (1987) 269-283.

[9] F. Bry, Query Evaluation in Recursive Databases: Bottom-up and Top-down Reconciled, *Proc. DOOD89* (1989) 20-39.

[9] S. Ceri and L. Tanca, Optimization of Algebraic Equations for Evaluating Datalog Queries, *Proc. 13th VLDB* (1987) 31-41.

[11] S. Ceri, G. Gottlob and L. Tanca, What You Always Wanted to Know About Datalog, *IEEE TKDE*, Vol.1, No.1 (1989) 146-166.

[12] C.L. Chang and R.C.T. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, 1973).

[13] G. Gardarin, Magic Functions: A Technique to Optimize Extended Datalog Recursive Programs, *Proc. 13th VLDB* (1987) 21-30.

[14] M. Kifer and E.L. Lozinskii, Filtering Data Flow in Deductive Databases, *Proc. ICDT* (1986) 186-202.

[15] M. Kifer and E.L. Lozinskii, SYGRAF: Implementing Logic Programs in a Database Style, *IEEE TOSE*, Vol. 14, No.7 (1988) 109-116.

[16] J.W. Lloyd, *Foundations of Logic Programs*, 2nd edition, (Springer-Verlag, 1987).

[17] N. Miyazaki, K. Yokota, H. Haniuda and H. Itoh, Horn Clause Transformation by Restrictor in Deductive Databases, *J. Inf. Process.*, Vol.12, No.3 (IPSJ, 1989) 266-279.

[18] Miyazaki, N., Haniuda, H., Yokota, K. and Itoh, H., A Framework for Query Transformations in Deductive Databases, to appear in *J. Inf. Process.*, Vol.12, No.4.

[19] J.F. Naughton, R. Ramakrishnan, Y. Sagiv and J.D. Ullman, Efficient Evaluation of Right-, Left-, and Multi-Linear Rules, *Proc. 8th ACM SIGMOD* (1989) 235-242.

[20] R. Ramakrishnan, Magic Templates: A Spell Binding Approach to Logic Programs, *Proc. 5th Intl. Conf/Symp. on Logic Programming* (1988) 140-159.

[21] D. Sacca and C. Zaniolo, Implementation of Recursive Queries for a Data Language Based on Pure Horn Logic, *Proc. 4th ICLP* (1987) 104-135.

[22] Y. Sagiv, Optimizing Datalog Programs, in: J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, 1988) 659-698.

[23] H. Seki, On the Power of Alexander Templates, *Proc. 8th ACM PODS* (1989) 150-159.

[24] H. Tamaki and T. Sato, OLD Resolution with Tabulation, *Proc. 3rd ICLP* (1986) 84-98.

[25] J.D. Ullman, *Principles of Database and Knowledge-Base Systems*, two volumes (Computer Science Press, 1988 and 1989).

[26] J.D. Ullman, Bottom-up Beats Top-down for Datalog, *Proc. 8th ACM PODS* (1989) 140-149.

[27] M.H. van Emden and R.A.Kowalski, The Semantics of Predicate Logic as a Programming Language, *JACM*, Vol. 23, No. 4 (1976) 733-742.

[28] L. Vieille, Recursive Axioms in Deductive Databases: The Query/Subquery Approach, *Proc. 1st Intl. Conf. on Expert Database Systems* (1986) 179-193.

[29] L. Vieille, From QSQ towards QoSaQ: Global Optimization of Recursive Queries, *Proc. of 2nd Intl. Conf. on Expert Database Systems* (1988) 421-436.