ICOT Technical Report: TR-474

TR-474

Distribution of Selections: The Missing Link between Strategies for Relational Databases and Deductive Databases

by
N. Miyazaki

April, 1989

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

Distribution of Selections:

The Missing Link between Strategies for Relational Databases and Deductive Databases

Nobuyoshi Miyazaki
Oki Electric Industry Co., Ltd.
4-11-22 Shibaura Minato-ku Tokyo, 108 Japan

Abstract

The distribution of selections is a method to realize the selection-first principle in relational databases. Although its direct extensions have been proposed for recursive queries, they are effective only for certain queries. On the other hand, two methods are emerging as general frameworks of the query processing strategy in deductive databases. One is the top-down method, query/subquery and its generalizations (QSQs). The other is its counterpart in the bottom-up method, the magic set and its generalizations (MSs). The correspondence between QSQs and MSs is also being established. However, the relationship between these frameworks and the distribution of selections is not clear, although they all realize the selection-first principle.

This paper discusses the relationship between the distribution of selections and other methods. A method to distribute selections for general queries is proposed, and the relationship between this method and other methods is discussed. It is shown that there is a guarantee that QSQs (and MSs) are at least as effective as the distribution of selections as far as the selection propagation is concerned. It is also shown that there exists a variation of MSs that precisely corresponds to the distribution of selections. Hence, MSs can be regarded as natural generalizations of the selection-first principle in relational databases.

1. Introduction

The selection-first principle is one of the most important principles for query optimization in relational databases [Ull82]. It is realized in two steps:

(1) Transformation of queries by distribution of selections (or pushing selections) based on the commutativity of selections with other operators of relational algebra.

(2) Performing selection-first evaluation of transformed queries.

The extension of this principle was proposed in the framework of bottom-up method for transitive closure queries in [AU79]. Extensions of the distribution of selections for more general queries have been proposed in [AD88] [KL86] [MHYI88]. Selection-first evaluation can also be performed in methods for bottom-up computation proposed in [Ban86] [BaR87]. It is known that the selection is commutative with the least fixpoint operator only for certain positions of constants

in goals even for simple recursive queries. Hence, the direct extension of this principle is not considered as a general framework of query processing strategy because of its limited applicability, although it is simpler and more efficient than more general methods for certain queries.

Meanwhile, two general frameworks of the recursive query processing are emerging. One is the query/subquery (QSQ) and its generalizations (referred QSQs in this paper) based on top-down evaluation [Vie86,87,88] [TS86]. The other is its counterpart in bottom-up method, magic set and its generalizations (referred MSs) [BMSU86] [RLK86] [SZ87] [BeR87] [Ram88] [MYHI88]. MSs are designed as transformations used before bottom-up computation, and selection-first evaluation is performed during bottom-up computation. QSQs and MSs are considered as general frameworks because they are effective for a broad class of recursive queries. It is known that there are strong connections between QSQs and MSs, and their correspondences were discussed in [Vie88] [Sek89] [Ull89]. They perform selection-first either in themselves (QSQs) or in the evaluation phase (MSs). However, the relationship of these methods to the classical distribution of selections is not clear, although they realize a somehow more general selection propagation. A framework for the selction propagation that includes both the distribution of selections and MSs is discussed in [BKBR87]. Although it gives a complete characterization of the problem for a class of queries called *chain programs*, its applicability is limited. The situation is illustrated in Figure 1 that shows the relationship of these methods.

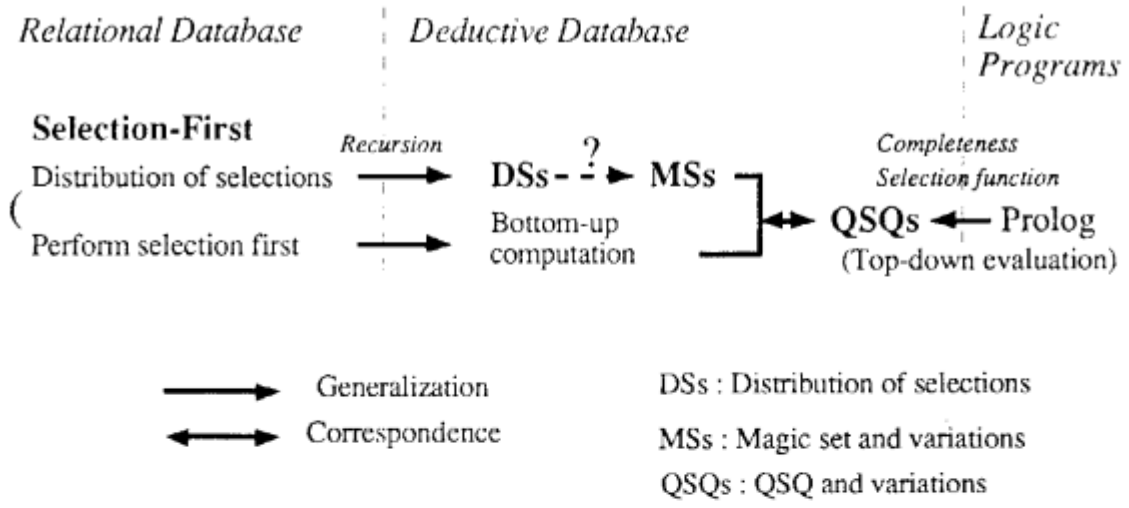

Figure 1 Relationships between Methods

The framework of query processing strategies for deductive databases should be a generalization of the framework of those for relational databases. Figure 1 shows that this requirement is practically satisfied, because MSs (and QSQs) are effective for a broader class of queries than the distribution of selections. However, this requirement is not formally satisfied, because the precise relationship between MSs and the distribution of selections is not known. For

instance, the result of MSs for non-recursive queries are not same as the result of the distribution of selections. Moreover, the distribution of selections is more efficient than MSs for certain queies, 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 bridge this gap. The subject involves two issues:

(1) Generalization of the distribution of selections for a broader class of queries.

(2) Discussion on the relationship between the generalized distribution of selections and other methods.

The organization of this paper is as follows. Section 2 summarizes methods for the distribution of selections. Section 3 discusses the generalization and optimization of the distribution of selections. The relationship between the proposed method and other methods is discussed in section 4. The result is summarized in section 5.

2 Distribution of Selections for Datalog Queries

The extension of the distribution of selections for recursive queries was first proposed in [AU79]. This method is referred as AU in this paper. The following example shows how the distribution of selections are performed by AU.

Let consider a transitive closure query, the ancestor example:

The goal of the query is

:-ancestor(X,taro).

The clauses defining the ancestor are:

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:

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 the ancestor relation as the least fixpoint (lfp) of ancestor = f(ancestor) and then applies the selection, $\sigma_{2=taro}$ to the ancestor relation. However, computing the ancestor relation as the lfp is time consuming. 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 computed directly without computing the whole ancestor

relation. Note that this kind of optimization is always possible for non-recursive queries but it is not possible to distribute the selection for certain recursive queries. For instance, if the constant in the goal appears in the first argument instead of the second argument, the selection can not be distributed.

AU can be applied only for queries that satisfy well-typed condition [AD88]. A generalization of the method for more general linear queries was proposed to overcome this problem in [AD88]. This method, referred as AD in this paper, transforms linear queries in the following way. Suppose a query is expressed as:

σ_P (lfp (r=f(r)) where *lfp* is the least fixpont operator.

AD transforms this into the optimized form:

σ_P (lfp (r=f(σ_F(r))) where the function *f* remains unchanged.

Note that the selection inside of the lfp operator is not same as the selection outside. The condition F can be expressed as:

$F = P \vee \{\vee_i \ Cond_i\}$ where $Cond_i$ refers to a simple condition.

An algorithm that determines the strongest condition F was proposed based on the analysis of a graph that represents how arguments are related in the query. For instance, consider a clause:

p(X,Y,Z,X) :- p(W,X,Y,Z),q(X,Z).

Because this clause is not well-typed, AH is not able to distribute the selection. However, the distribution is possible by AD by choosing F as follows:

Suppose that P= (1=v), i.e., the goal is :-p(v,X,Y,Z). Then,

$F = (1=v) \vee (2=v) \vee (3=v) \vee (4=v)$.

The improvement by AD can be seen even for simple transitive closure queries. For instance, suppose that both arguments are constants in the goal in ancestor query mentioned above. AU fails to transform this query, because the selection itself is not commutative with the lfp operator [AD88]. However, AD finds that the selection for the second argument is commutative and distributes that part into the lfp operator.

A method called the static filtering, referred as KL in this paper, was proposed based on a graphical notation in [KL86]. Selections are considered as filters that blocks the data flow during the computation using a graph called the system graph similar to the rule/goal graph in [Ull85]. Selections are repeatedly pushed into the graph to obtain every condition that is necessary to compute the complete answer. Conditions are also expressed in disjunctive forms in KL. KL can be applied to more general queries than linear queries.

An important contribution of AD and KL is that the selection expressed as a disjunction of conditions is sometimes commutative with lfp operator when the original selection itself is not commutative. However, all methods mentioned above have following problems.

(1) They cannot handle queries with function symbols. Methods other than KL are difficult to generalize for more complex queries, because they check commutativity before transforming a query.

(2) They can be used only with certain evaluation algorithms. Hence, it is difficult to combine these methods with other optimization methods. Moreover, it is difficult to recognize the relationship of these methods to other methods.

Therefore, a method using clausal notation is desirable. A method called Horn clause transformation by substitution (HCT/S) [MHY188] gives a partial solution for this problem. A simpler and more general algorithm than HCT/S is proposed in the next section.

3. Distribution of Selections for General Recursive Queries

3.1 Fundamental Concepts

Definition (Notations)
clause: A clause means a Horn clause in this paper. It is denoted h:-B where h is an atom and B is a conjunction of atoms. Function symbols may be used in clauses.
$\Rightarrow$: $A \Rightarrow B$ means B is a logical consequence of A.
$\supset$: Set inclusion.
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 $C = B\theta$. B is *less general* than C iff C is more general than B. B is a *variant* of C iff B is both more general and less general than C. We treat variants as if they are identical when we discuss a set of clauses (or atoms). For instance, {a(X,Y), a(V,W)} = {a(X,Y)}. ■

Definition Let B and C be clauses. B *subsumes* C iff there exists a substitution θ such that $B\theta$ is a sub-formula of C. It is obvious that if B subsumes C, then $\{B\} \Rightarrow C$. ■

Definition A *database* (DB) is a finite set of ground unit clauses. A *query* is the set $Q = \{:\text{-}g\} \cup D$ where g is a *goal atom* and D is a finite set of definite clauses. A predicate that appear in the head of a clause in Q is called a *derived predicate*, and a predicate that appears in the database is called a *base predicate*.

Let g' be a ground instance of g. The *answer* of the query is the set $G = \{g' \in B \mid D \cup DB \Rightarrow g'\}$ where B is the Herbrand base of $D \cup DB$. ■

We assume that the set of base predicates and the set of derived predicates are disjoint to simplify discussion.

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 $Q \leq Q'$. ■

The relation "≈" is an equivalence relation. A slightly more general definition of equivalence that allows different arities of goal predicates is used in [BeR87] [BKBR87]. Although we recognize the advantage of the generalized definition, the above definition is used to simplify discussion. The generalized equivalence may be used in informal discussions in this paper.

Definition A *query transformation* is a mapping from the set of all queries to itself. Let f be a query transformation. f is *complete* iff f(Q) ≥ Q for any Q. f is *sound* iff f(Q) ≤ Q for any Q. f is *equivalent* iff f(Q) = Q for any Q. ■

3.2 Generalized Distribution of Selections

In [BR86], it is pointed out that AU and KL can be explained using clausal notation. For instance, the result of AU and KL for ancestor example for the goal :-ancestor(X,taro) is equivalent to the following set of clauses:

```
ancestor(X,taro):-parent(X,taro)
ancestor(X,taro):-parent(X,Z),ancestor(Z,taro).
```

Because all instances of the predicate ancestor is essentially the same, these clauses are equivalent to the original clauses for the goal :-ancestor(X,taro). This set of clauses can be directly obtained by applying the most general unifier (mgu) of the goal atom and the head of original clauses to substitute variables in these clauses.

What happens if the goal atom is ancestor(jiro,taro) instead of ancestor(X,taro)? By applying mgu as a substitution, we obtain the following set of clauses:

```
ancestor(jiro,taro):-parent(jiro,taro)
ancestor(jiro,taro):-parent(jiro,Z),ancestor(Z,taro).
```

However, this set of clauses is not equivalent to the original query, because ancestor(Z,taro) appears in the body of the second clause. In such case, we can apply the substitution again using the mgu of ancestor(Z,taro) and the heads of original clauses to obtain:

```
ancestor(X,taro):-parent(X,taro)
ancestor(X,taro):-parent(X,Z),ancestor(Z,taro).
```

Clearly this is equivalent to the original query. The first set of clauses obtained by substitution is subsumed by the second set and can be eliminated. Corresponding results for this query can be obtained by AD and KL.

Next, let us consider again the non-well-typed example in section 2.

```
:-p(v,X,Y,Z).
p(X,Y,Z,X) :- p(W,X,Y,Z),q(X,Z).
```

By applying substitution, we obtain:

```
p(v,Y,Z,v) :- p(W,v,Y,Z),q(v,Z).
```

Because p in the body is not less general than the head, we apply substitution again to obtain:

p(X,v,Z,X) :- p(W,X,v,Z),q(X,Z).

We have to further apply substitution to obtain:

p(X,Y,v,X) :- p(W,X,Y,v),q(X,v).

We do not obtain other clauses even if this process is further continued. Moreover, the resulting set of clauses is equivalent to the original query, because subgoals more general than atoms in the bodies of these clauses do not appear in SLD resolution of the original query. This result corresponds to the disjunction of conditions in the result of AD.

We can formulate a transformation procedure based on the above observation. The fixpoint operator can be defined for a set of definite clauses as well as for relational algebra expressions [Llo87]. Hence, the procedure defined below can be regarded as a generalization of the distribution of selections. First, we observe that all clauses appeared in the above discussion are obtained by substitution of clauses in the query.

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

It is obvious that S(Q) is equivalent to Q, because each clause in S(Q) is subsumed by a clause in Q. Thus, the problem of the distribution of selections is to find the minimal subset of S(Q) that is equivalent to Q. First, we ignore the minimality problem.

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

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

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

It is obvious that if R is a subset of S(Q), so is $F_Q(R)$. 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.

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 $\{F_Q(C) \mid C \in X\}$.

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 $F_Q(R1) = \{:\text{-}g\} \cup F'_Q(R1)$.

$C \in F'_Q(lub(X))$

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

$T\uparrow 0 = \bot$

$T\uparrow \alpha = T(T\uparrow(\alpha-1))$, 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. The proof is the same as proposition 5.4 in [Llo87] and is omitted.

Lemma 3.2 There exists the least fixpoint of F_Q denoted $lfp(F'_Q)$, and $lfp(F_Q) = F_Q\uparrow\omega$ where ω is the smallest ordinal next to 0. Hence, $lfp(F_Q)$ is the limit of $Q_{i+1} = F_Q(Q_i)$, $Q_0 = \varnothing$.

■

Lemma 3.3 Let Q' be a fixpoint of F_Q. Then, $Q' = Q$.

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 that can be unified with the subgoal is included in Q' in a substituted form. ■

The following theorem is obtained from lemmata 3.1, 3.2 and 3.3.

Theorem 3.4

(a) $F_Q\uparrow\omega = lfp(F_Q) \approx Q$.

(b) $Q' \approx Q$ for any Q' such that $S(Q) \supset Q' \supset lfp(F_Q)$. ■

Thus, the procedure that computes $lfp(F_Q)$ distributes selections for general recursive queries. This procedure is also a generalization of the distribution of selections in relational databases, because F_Q distributes selections based on essentially the same principle as commutativity of selection with other relational algebraic operators and the procedure produces essentially the same result as the algebraic transformation for non-recursive queries. The result of the procedure may be infinite and the procedure may not terminate, although the termination is guaranteed for Datalog queries. We can modify the procedure by using the concept of the least generalization in order to guarantee the termination for general queries [MHYI88].

The above procedure distributes selection conditions embedded in predicates. In relational algebra, conditions are expressed as combinations of "attribute θ value" and "attribute θ attribute". If θ is =, the conditions can be converted to those in 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 attached to an element of body(R), 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, $lfp(F_Q)$, may contain redundant clauses as shown in the ancestor example at the beginning of the section. The result even contains Q itself, if the procedure fails to produce a better result. For instance, the result includes original clauses for the goal, :-ancestor(jiro,X), in ancestor example. Hence, we have to consider some optimization procedure.

We may define the optimal form of a transformed query as follows:

(1) It is non-redundant, i.e. it does not contain a clause that is a logical consequence of other clauses.

(2) It gives the smallest least model for any database. Or, it does not contain a clause that does not contribute to the answer.

Although we can obtain optimal results for linear queries [AD88], the problem is undecidable for general recursive queries [Sag87]. Hence, we restrict our attention to special cases of redundancy.

Definition A query $Q = \{:\text{-}g\} \cup D$ *subsumes* another query $Q' = \{:\text{-}g\} \cup D'$ iff each element of D' is subsumed by a clause in D. A subset Q' of Q is called a *subsumption cover (s-cover)* of Q iff Q' subsumes Q. An s-cover of Q is called a *minimum s-cover* of Q iff there is no proper subset of it that is an s-cover of Q.

Clearly there exists a unique minimum s-cover for any Q, and the covering problem is decidable for finite Q. The function that maps a query Q to its minimum s-cover, *mscv(Q)*, can be regarded as a mapping from $2^{S(Q)}$ to itself. ■

Lemma 3.5

(a) $mscv(S(Q)) = mscv(Q)$.

(b) mscv is monotonic.

(c) The composition of F_Q and mscv, i.e. $(F_Q{*}mscv)(R) = mscv(F_Q(R))$ is monotonic.

(d) $F_Q{*}mscv$ has a least fixpoint $lfp(F_Q{*}mscv)$.

(e) $lfp(F_Q{*}mscv) \supset F_Q{*}mscv\uparrow\alpha$ for any ordinal α. Furthermore, there exists an ordinal β such that $\beta \leq \gamma$ implies $F_Q{*}mscv\uparrow\gamma = lfp(F_Q{*}mscv)$.

Proof (a), (b) and (c) are obvious by the definition of mscv. (d) and (e) are consequences of (c) as shown in propositions 5.1 and 5.3 in [Llo87]. ■

The composition, $F_Q{*}mscv$, can be used to define an improved procedure for the distribution of selections.

Lemma 3.6 Let Q' be a fixpoint of $F_Q{*}mscv$. Then, $Q' \approx Q$.

Proof Same as proof for lemma 3.3. ■

Unfortunately, mscv is not continuous, because $mscv(Q1 \cup Q2) \neq mscv(Q1) \cup mscv(Q2)$. However, we can prove that $F_Q{*}mscv\uparrow\omega = lfp(F_Q{*}mscv)$.

Theorem 3.7

(a) $F_Q{*}mscv\uparrow\omega = lfp(F_Q{*}mscv) = mscv(lfp(F_Q))$

(b) $Q' = Q$ for any Q' such that $S(Q) \supset Q' \supset lfp(F_Q*mscv)$. ■

The proof is shown in appendix 1. The procedure that computes $lfp(F_Q*mscv)$ can be regarded as the improved procedure that distributes selections. The above theorem also implies the elimination of redundant clauses by subsumption can be performed any time during the transformation.

4 The Relationship of the Distribution of Selections to Other Methods

4.1 The Relationship of the Distribution of Selections to Top-Down Methods

The following theorem is a direct consequence of the equivalence of the distribution of selections.

Theorem 4.1 Let $Q = \{:-g\} \cup D$, and DB be a database. Let $Q' = lfp(F_Q) = \{:-g\} \cup D'$, let $Q'' = lfp(F_Q*mscv) = \{:-g\} \cup D''$, 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 subsumed by Q. $model(D' \cup DB) = model(D'' \cup DB)$, because $lfp(F_Q*mscv) = mscv(lfp(F_Q))$ by theorem 3.7. $model(D' \cup DB) \supset S\_Ans$ is proved by the same way as lemma 3.3. ■

This theorem guarantees that a top-down method based on SLD resolution is at least as efficient as the combination of the distribution of selections and the bottom-up computation, if performing selection-first is properly used in the top-down method. Because QSQs use the *selection function* (corresponds to computation rule in SLD resolution) to realize selection-first evaluation [Vie86], they are at least as efficient as the distribution of selections. Because of the correspondence between QSQs and MSs [Vie88] [Sek89] [Ull89], the theorem also implies that MSs is at least as efficient as the distribution of selections if we neglect the overhead introduced by magic predicates. Note that it does not guarantee that all top-down methods are at least as efficient as the distribution of selections.

The following examples illustrate the meaning of the theorem. Let consider the transformed result of ancestor example.

```
:-ancestor(X,taro)
ancestor(X,taro):-parent(X,taro)
ancestor(X,taro):-parent(X,Z),ancestor(Z,taro).
```

The query can be efficiently evaluated by bottom-up computation if the second clause is evaluated from right to left (selection-first evaluation). A top-down evaluation of the original query is as efficient as bottom-up computation if the body of the second clause is evaluated from right to left.

However, a Prolog-like top-down evaluation with left to right evaluation strategy is inefficient for this query.

Next, let consider a slightly different query. Suppose that parent is defined in terms of two relations, father and mother. The transformed query by the distribution of selections 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. Hence, the evaluation by bottom-up computation is not efficient, because whole parent relation must be computed. Thus, the distribution of selections may not be effective even for non-recursive derived predicates. However, a top-down method with an appropriate computation rule (and MSs) can process this query efficiently. Another way to process it efficiently is eliminate parent by another transformation discussed in [CGL86] [MHI88] [MHYI88]. The transformed query is:

```
:-ancestor(X,taro)
ancestor(X,taro):-father(X,taro)
ancestor(X,taro):-mother(X,taro)
ancestor(X,taro):-father(X,Z),ancestor(Z,taro).
ancestor(X,taro):-mother(X,Z),ancestor(Z,taro).
```

It is easy to see that this query can be efficiently processed by bottom-up computation. We can further improve the result by eliminating the second argument of ancestor, because its all instances are the same [BKBR87]. The resulting query is:

```
:-anc_taro(X)
anc_taro(X):-father(X,taro)
anc_taro(X):-mother(X,taro)
anc_taro(X):-father(X,Z),anc_taro(Z).
anc_taro(X):-mother(X,Z),anc_taro(Z).
```

The above query is not equivalent according to our definition, but it is easy to see the equivalence by considering that the second argument exists implicitly. Reducing arities of predicates improves the performance as discussed in [BR86].

4.2 The Relationship of the Distribution of Selections to Magic Sets

We show that there exists a variation of MSs that corresponds to the distribution of selections. First, we summarize basic concepts of Horn clause transformation by restrictor (HCT/R). HCT/R is a transformation that maps a clause r:- R to clause(s) of the form r:-r^*,R where r^* is called a *restrictor*. Because of the existence of restrictors, the transformed query has a smaller least model than the original query. The transformation gives an equivalent query if clauses

for restrictors are defined properly. The conditions for equivalence are found in [MYHI88]. 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 it restricts. The partial restrictor has smaller arity with proper adornment like the magic set method. Ways to obtain semi-optimal results are also discussed in [MYHI88]. The framework of the partial restrictor version is more general than other similar methods proposed in [BMSU86] [SZ87] [BeR87] [Ram88] in the sense that they can be formulated in the framework of HCT/R. A special case of the full restrictor version is used in this section.

Definition A transformation called *static HCT/R* is defined as follows:

Let Q = {:-g} ∪ D be a query. The transformed query Q' = st_hctr(Q) is obtained by following steps.

(1) Let C = r:- r_1,...,r_n be a clause in D. Replace each C by the following clause:

r:-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 unit clause, g*, having the same argument as the goal atom.

(3) For each r:-r*,r_1,...,r_n generate following clauses and add them:

For each i, generate a clause, $r*_i$:-r*, if the predicate of r_i is a derived predicate. ■

Static HCT/R is the simplest form of HCT/R. The equivalency of static HCT/R can easily be shown by the theorem for the equivalency of the transformation in [MYHI88]. The equivalency can also be directly proved by comparing SLD-trees of the original and the transformed queries. The following example illustrates how static HCT/R works. Let consider the ancestor query again.

Goal: :-ancestor(jiro,X).

Rules: ancestor(X,Y) :- parent(X,Y)

ancestor(X,Y) :-parent(X,Z),ancestor(Z,Y).

Magic set method generates the following set of clauses as the transformed result. Here, ancestor * is the magic predicate (called restrictor predicate in HCT/R).

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

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

$ancestor^{bf}$(X,Y) :- $ancestor*^{bf}$(X),parent(X,Z),$ancestor^{bf}$(Z,Y).

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

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

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

Goal: :- ancestor(taro,X)

Modified rules: ancestor(X,Y) :- ancestor*(X,Y),parent(X,Y)

ancestor(X,Y) :- ancestor*(X,Y),parent(X,Z),ancestor(Z,Y).

Seed: ancestor*(taro,X).

Magic rule: ancestor*(Z,Y) :- ancestor*(X,Y),parent(X,Z).

This result may be practically meaningless, because it violates the safety condition of queries [BR86]. Note that the seed is a nonground unit clause, and it violates the safety condition. However, it is theoretically meaningful, because the safety of queries and the equivalence of queries are independent concepts, i.e., we may consider equivalent transformation even for unsafe queries. Moreover, this result has an interesting property. The resulting set of clauses is equivalent to the original query for any types of goals, by changing only the seed. For instance, it is equivalent for the goal, :-ancestor(X,taro), if the seed is changed to ancestor*(X,taro). It is also equivalent for the goal, :-ancestor(jiro,taro), if the seed is changed to ancestor*(jiro,taro). This is an important property of the full restrictor version of HCT/R. Note that all clauses have to be changed in the usual magic set, if the type of binding in the goal is changed.

What happens if parent(X,Z) in the body of the magic rule is eliminated? The result is shown below, and this set is the result of static HCT/R defined above.

Goal: :- ancestor(taro,X)
Modified rules: ancestor(X,Y) :- ancestor*(X,Y),parent(X,Y)
ancestor(X,Y) :- ancestor*(X,Y),parent(X,Z),ancestor(Z,Y)
Seed: ancestor*(taro,X)
Magic rule: 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 restricting the computation space, because the binding propagated by parent(Z,Y) cannot be propagated. Moreover, the effect is exactly same as that of distribution of selections discussed in section 3. For instance, binding on the first argument vanishes for the 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) the binding is propagated by the magic rule. If the goal is :-ancestor(jiro,taro), only the binding on the second argument is propagated. Thus, static HCT/R has the same effect as the distribution of selections for the ancestor query. The remaining part of this section shows that this observation is valid for all queries.

We need several definitions to show the correspondence between the distribution of selections and static HCT/R. The following is the definition of a mapping used to define fixpoint semantics of logic programs [Llo87].

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

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

We need a mapping that produces nonground instances. Thus, we define another mapping.

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. A mapping T^+_D from 2^U to 2^U is defined as follows. Let I be an element of 2^U.

$T^+_D(I) = \{r' \in U \mid (r' \text{ is a unit clause in D}) \vee$

$(r:- r_1,r_2,...,r_n \in D \wedge \exists p_1,p_2,...,p_n \in I \wedge$

$\vartheta_i = mgu(r_i,p_i) \wedge \theta = \theta_1\theta_2,...,\theta_n \wedge r'=r\theta)\}$. ■

The difference is that the result of $T^+_D(I)$ may contain uninstantiated instances. The following lemma shows the properties of T^+_D.

Lemma 4.2

(a) 2^U is a complete lattice under set inclusion. The bottom ($\perp$) is $\varnothing$, and the top element 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 by the same way as proposition 6.3 in [Llo87]. (d) is a consequence of (a), (b) and (c). ■

The following lemma shows the correspondence between T^+_D and T_D.

Lemma 4.3 Let I be a subset of 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, $ground(lfp(T^+_D)) = ground(T^+_D\uparrow\omega) = ground(lub(T^+_D\uparrow\beta: \beta<\omega))$

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

The following theorem shows that the result of the distribution of selections can be obtained by transforming the result of static HCT/R. It also shows the correspondence between the least Herbrand models of the transformed results.

Theorem 4.4 Let $Q = \{:-g\} \cup D$, 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.

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

$\theta \text{ is } mgu(r^*,p^*) \wedge r' = (r :- r_1,r_2,...,r_n)\theta)\}$.

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

(b) Let DB be a database, let D1 = (lfp(F_Q) - {:-g}) ∪ DB, and let D2 = D' ∪ D* ∪ DB. Let denote model(D) instead of lfp(T_D) to simplify notation.
Then, model(D1) = model(D2) − model(D*), for any Q and DB. ■

The proof is shown in appendix 2. Theorem 4.4 shows the precise correspondence between the distribution of selections and static HCT/R. Because static HCT/R is a special case of MSs, MSs can be considered as generalizations of the distribution of selections. The following descriptions summarize how the distribution of selections and MSs are related.

(1) **AU**: The distribution of selections in relational databases is based on the commutativity of selections with other operators. This method can be generalized for transitive closure queries.

(2) **AD and KL**: For more complex queries, the distribution of selections can be generalized by introducing *disjunctions of conditions*.

(3) **Generalized distribution of selections** (in section 3): The distribution of selections can be generalized for general recursive queries. This method also uses disjunction of conditions extracted *only* from queries. However, it fails to optimize queries when extracted conditions are ineffective.

(4) **MSs**: A possible way to improve the above method is to extract conditions from *databases* as well as from *queries*. Because it is inefficient to extract conditions from databases before the fixpoint computation, it should be performed during the computation. Magic predicates are used as storage for these conditions. Note that magic sets can be regarded as disjunctions of conditions if each of its element is considered as a simple condition. Static HCT/R is a variation of MSs which extracts conditions only from queries, and thus corresponds to the generalized distribution of selections.

Thus, the difference between the distribution of selections and MSs is that the former only uses conditions extracted from queries, and the latter extracts conditions from databases as well as from the queries. Because it seems necessary to introduce new predicates in order to store conditions extracted from the database, we can conclude that MSs are natural generalizations of the selection-first principle in relational databases.

Theorem 4.4 also suggests that we can improve MSs using the above correspondence. The following is the possible but not exhaustive directions for the improvement.

(1) Elimination of magic predicates after the transformation.

(2) Application of the distribution of selections before MSs. The arities of predicates may also be reduced. The effectiveness of the distribution of selections 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.

We also notice the similarity between the distribution of selections and the rectification algorithm proposed as a transformation before MSs in [Ull89]. Investigating these possibilities is a topic of future research.

5 Conclusions

This paper discussed the generalization of the selection-first principle to recursive queries. A generalization of distribution of selections was proposed, and its relationships to other methods was discussed. We have shown that QSQs and MSs can be at least as efficient as the distribution of selections. We have also shown that there exists a variation of MSs that corresponds to the distribution of selections. As a result, the relationship between general frameworks of query processing strategies for relational databases and deductive databases becomes clearer. The revised relationship is illustrated in Figure 2. We only discussed queries expressed by a goal and definite clauses. The generalization of our result for stratified queries is possible, because both the distribution of selections and static HCT/R produce stratified results for a stratified query.

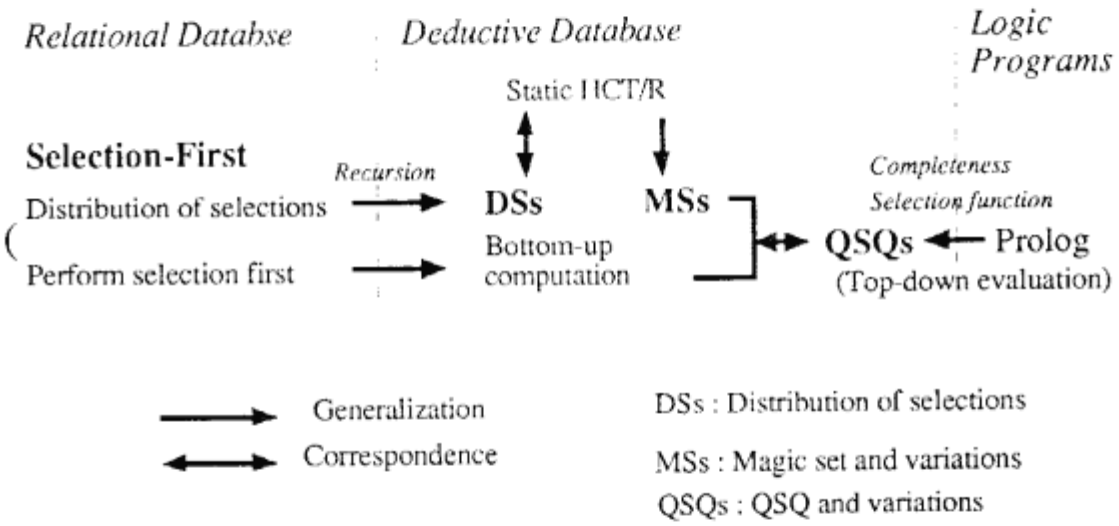

Figure 2 Revised Relationships between Methods

Acknowledgement

This work is a part of knowledge base research in the Fifth Generation Computer Systems Project. I would like to thank to members of KBMS PHI group at ICOT and Oki for the development of HCTs. Special thanks are due to Catriel Beeri and Kazumasa Yokota for valuable discussions on the general frameworks of query processing.

References

[AD88] Agrawal, R. and Devanbu, P., Moving Selections into Linear Least Fixpoint Queries, Proc. 4th Intl. Conf. on Data Engineering, 1988.

[AU79] Aho, A.V., and Ullman, J.D., Universality of Data Retrieval Languages, Proc. 6th ACM Symp. on Principles of Programming Languages, 1979.

[BaR87] Balbin, I. and Ramamohanarao, K., A Generalization of the Differential Approach to Recursive Query Evaluation, J. of Logic Programming, 1987.

[BMSU86] Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J.D., Magic Sets and Other Strange Ways to Implement Logic Programs, Proc. of 5th ACM PODS, 1986.

[BR86] Bancilhon, F. and Ramakrishnan, R., An Amateur's Introduction to Recursive Query Processing Strategies, Proc. ACM SIGMOD, 1986.

[Ban86] Bancilhon, F., Naive Evaluation of Recursively Defined Relations, in M.L. Brodie and J. Mylopoulos (eds.) On Knowledge Base Management Systems, Springer-Verlag, 1986.

[BKBR87] Beeri, C., Kanellakis, P., Bancilhon, F. and Ramakrishnan, R., Bounds on the Propagation of Selection into Logic Programs, Proc. 6th ACM PODS, 1987.

[BeR87] Beeri, C. and Ramakrishnan, R., On the Power of Magic, Proc. 6th ACM PODS, 1987.

[CGL86] Ceri, S., Gottlob, G. and Lavazza, L., Translation and Optimization of Logic Queries: An Algebraic Approach, Proc. 12th VLDB, Aug. 1986.

[KL86] Kifer, M. and Lozinskii, E.L., Filtering Data Flow in Deductive Databases, Proc. ICDT, 1986.

[Llo87] Lloyd, J.W., Foundations of Logic Programs, 2nd edition, Springer-Verlag, 1987.

[MHI88] Miyazaki, N., Haniuda, H. and Itoh, H., Horn Clause Transformation: An Application of Partial Evaluation in Deductive Databases, Trans. IPSJ, Vol. 29, No.1, 1988 (in Japanese).

[MHYI88] Miyazaki, N., Haniuda, H., Yokota, K. and Itoh, H., A Principle of Query Transformations in Deductive Databases, ICOT TR-377, 1988. (submitted for publication)

[MYHI88] Miyazaki, N., Yokota, K., Haniuda, H. and Itoh, H., Horn Clause Transformation by Restrictor in Deductive Databases, ICOT TR-407, 1988. (submitted for publication)

[Ram88] Ramakrishnan, R. Magic Templates: A Spell Binding Approach to Logic Programs, Proc. 1st Intl. Conf/Symp. on Logic Programming, 1988.

[RLK86] Rohmer, J., Lescoeur, R. and Kerisit, J.M., The Alexander Method - A Technique for the Processing of Recursive Axioms in Deductive Database, New Generation Computing, Vol. 4, pp.273-285, 1986.

[SZ87] Sacca, D. and Zaniolo, C., Implementation of Recursive Queries for a Data Language Based on Pure Horn Logic, Proc. ICLP, 1987.

[Sag87] Sagiv, Y., Optimizing Datalog Programs, Proc. 6th ACM PODS, 1987. also in Foundations of Deductive Databases and Logic Programming, edited by J. Minker, 1988.

[Sek89] Seki, H., On the Power of Alexander Templates, Proc. 8th ACM PODS, 1989.

[TS86] Tamaki, H. and Sato, T., OLD Resolution with Tabulation, Proc. of 3rd Intl. Conf. on Logic Programming, 1986.

[Ull82] Ullman, J.D., Principles of Database Systems, Computer Science Press, 1982.

[Ull85] Ullman, J.D., Implementation of Logical Query Languages for Databases, ACM TODS, Vol. 10, pp 289-321, 1985.

[Ull89] Ullman, J.D., Bottom-up Beats Top-down for Datalog, Proc. 8th ACM PODS, 1989.

[Vie86] Vieille, L., Recursive Axioms in Deductive Databases: The Query/Subquery Approach, Proc. 1st Intl. Conf. on Expert Database Systems, pp.179-193, 1986.

[Vie87] Vieille, L., Database Complete Proof Procedures Based on SLD Resolution, Proc. 4th Intl. Conf. on Logic Programming, pp.74-103, 1987.

[Vie88] Vieille, L., From QSQ towards QoSaQ: Global Optimization of Recursive Queries, Proc. of 2nd Intl. Conf. on Expert Database Systems, pp.421-436, 1988.

Appendix 1 Proof of theorem 3.7

Theorem 3.7

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

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

Proof (b) is obvious by lemma 3.6. The proof of (a) is shown in three steps.

(I) mscv(lfp(F_Q)) ⊃ lfp(F_Q*mscv). This is proved as follows. Let Q' be a fixpoint of F_Q. Then, mscv(Q') = mscv(F_Q(Q')) ⊃ mscv(F_Q(mscv(Q'))). Because F_Q(mscv(Q')) ⊃ mscv(Q'), mscv(F_Q(mscv(Q'))) ⊃ mscv(mscv(Q')) = mscv(Q'). Hence, mscv(Q') is a fixpoint of F_Q*mscv. Thus, mscv(lfp(F_Q)) is a fixpoint of F_Q*mscv, and therefore mscv(lfp(F_Q)) ⊃ lfp(F_Q*mscv).

(II) F_Q*mscv$\uparrow\alpha$ = mscv($F_Q\uparrow\alpha$) for $\alpha < \omega$. This is proved by mathematical induction.

(1) F_Q*mscv$\uparrow$0 = mscv($F_Q\uparrow$0) = ∅ for α = 0.

(2) Assume F_Q*mscv$\uparrow\alpha$ = mscv($F_Q\uparrow\alpha$) for α = i.

F_Q*mscv$\uparrow$i+1 = F_Q*mscv(F_Q*mscv$\uparrow$i) = F_Q*mscv(mscv($F_Q\uparrow$i))=

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

Thus, (b) can be proved by (1) and (2) using mathematical induction.

(III) F_Q*mscv$\uparrow\omega$ =lub{F_Q*mscv$\uparrow\alpha$: $\alpha < \omega$) = lub{mscv($F_Q\uparrow\alpha$): $\alpha < \omega$} ⊃

mscv(lub{$F_Q\uparrow\alpha$: $\alpha < \omega$}) = mscv(lfp(F_Q)) ⊃ lfp(F_Q*mscv).

Because lfp(F_Q*mscv) ⊃ F_Q*mscv$\uparrow\omega$,

F_Q*mscv$\uparrow\omega$ = lfp(F_Q*mscv) = mscv(lfp(F_Q)). ■

Appendix 2 Proof of theorem 4.4

Theorem 4.4 Let Q = {:-g} ∪ D, and let Q' = st_hctr(Q). Q' can be divided to Q' = {:-g} ∪ D' ∪ 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.

(a) Let $D^{\#}$ = {r' | ∃(r:- r*,r_1,r_2,...,r_n) ∈ D' ∧ ∃p* ∈ lfp($T^+_{D^*}$)∧

θ is mgu(r*,p*) ∧ r' = (r :- r_1,r_2,...,r_n)θ)}.

Then, lfp(F_Q) = {:-g} ∪ $D^{\#}$, for any Q.

(b) Let DB be a database, let D1 = (lfp(F_Q) - {:-g}) ∪ DB, and let D2 = D' ∪ D* ∪ DB. Let denote model(D) instead of lfp(T_D) to simplify notation.

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

Proof

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

Original clause: $r:-r_1,...,r_n$

Generated clauses: $r_1^*:-r^*$

$r_2^*:-r^*$

:

$r_n^*:-r^*$ (except for base predicates).

(I) First we prove the following property. Let body"(R) be the subset of body(R) that corresponds to predicates which appear in the head of 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) $T^+_{D^*}\uparrow 0 = \varnothing$

$F_Q\uparrow 0 = \varnothing$

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

$F_Q\uparrow 1 = \{:-g\}$

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

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

$T^+_{D^*}\uparrow j+1 = T^+_{D^*}(T^+_{D^*}\uparrow j)$ =

$$\{r' \in U \mid (r' \text{ is a unit clause in } D^*) \vee (r^*:- q^* \in D^* \wedge \exists p^* \in T^+_{D^*}\uparrow j \wedge \theta = mgu(q^*,p^*) \wedge r'=r^*\theta)\}$$

$$= \{g^*\} \cup \{r' \in U \mid r^*:- q^* \in D^* \wedge \exists p^* \in T^+_{D^*}\uparrow j \wedge \theta = mgu(q^*,p^*) \wedge r'=r^*\theta)\}$$

$$= \{g^*\} \cup \{r' \in U \mid r^*:- q^* \in D^* \wedge \exists p^* \in body"(F_Q\uparrow i) \wedge \theta = mgu(q^*,p^*) \wedge r'=r^*\theta)\}$$

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

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

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

(II) Next, we prove (a).

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

$$= \{r' \mid r:- r^*,r_1,r_2,...,r_n \in D' \wedge \exists p^* \in body"(lfp(F_Q)) \wedge \theta \text{ is } mgu(r^*,p^*) \wedge r' = (r :- r_1,r_2,...,r_n)\theta)\}.$$

$$= \{r' \mid r:- r_1,r_2,...,r_n \in D \wedge \exists p \in body"(lfp(F_Q)) \wedge \theta \text{ is } mgu(r,p) \wedge r' = (r :- r_1,r_2,...,r_n)\theta)\}.$$

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

(b) $M = model(D2) = model(D' \cup DB \cup D^*)$.

Because D* does not depend on other part, it can be separately evaluated. Thus,

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

Because $D^{\#}$ is a transformed form of D' using the information on $lfp(T^+_{D^*})$,

$M = model(D^{\#} \cup DB \cup lfp(T^+_{D^*}))$.

Because $D^{\#} \cup DB$ and $lfp(T^+_{D^*})$ are not related, they can be independently evaluated, and we obtain

$M = model(D^{\#} \cup DB) \cup model(lfp(T^+_{D^*}))$

$M = model(D^{\#} \cup DB) \cup model(lfp(T^+_{D^*}))$

$= model(D1) \cup model(lfp(T^+_{D^*}))$

$= model(D1) \cup model(D^*)$. ■