ICOT Technical Report: TR-377

TR-377

A Principle of Query Transformations in Deductive Databases

by
N. Miyazaki, H. Haniuda(Oki),
K. Yokota and H. Itoh

May, 1988

A Principle of Query Transformations in Deductive Databases

ICOT TR-377

(May 1988 Revised Sep. 1988)

Nobuyoshi Miyazaki*, Hiromi Haniuda*,
Kazumasa Yokota** and Hidenori Itoh**

* Oki Electric Industry Co., Ltd.
**Institute for New Generation Computer Technology

Abstract

Several methods that transform queries to their equivalents in order to improve the performance of bottom-up approach have been proposed for recursive query processing. This paper discusses the foundation of query transformations. First, the role of the query transformations is discussed and the concept of *goal equivalent transformation* is introduced. Goal equivalent transformation is a transformation that preserves the answer to a query but allows the change of the least models of the database. Because the least model of the database is computed by bottom-up approach, the performance can be improved by changing the size of the least model.

Next, ways to obtain smaller least models are discussed, and a conceptual procedure called clause replacement is proposed. Examples of procedures, which are applications of clause replacement using fundamental principles of first order logic such as resolution and subsumption, are shown. They include procedures called Horn clause transformation (HCT) by partial evaluation, HCT by substitution, and HCT by restrictor. Properties of these procedures are compared. The relationships of these procedures with other methods such as the distribution of selection and the magic set are discussed, and it is shown that the clause replacement is a common foundation of many transformation methods.

1. Introduction

Many methods for recursive query processing have been proposed. Summaries are found in [5] [13], and several newer methods have been proposed [6] [18] [23]. Because these methods are based on different concepts and techniques, it is difficult to recognize their logical relationship and to compare their performance. Bancilhon et al. [5] compared performance of several methods in terms of the size of the intermediate results and found three influencing factors to the performance.

(1) The amount of *duplication* of work.

(2) The size of the set of *relevant facts*.

(3) The use of *unary vs binary* intermediate relations.

There are two major approaches to recursive query processing; top-down approach based on the procedural semantics (SLD semantics) and bottom-up approach based on the fixpoint semantics [5] [10]. The typical bottom-up methods is the naive evaluation proposed in [3] and other literature. It computes the least fixpoint of the database by an iterative procedure. Although this method is well suited to apply techniques developed for relational database, its performance without improvement is not good because it computes possibly large fixpoints and the iteration involves a lot of duplication. Hence, several methods have been proposed to improve the performance of the bottom-up methods. There are mainly two ways for the improvement; improvement of the execution algorithm, and rewriting of rules before execution. Methods such as semi-naive and differential method are examples that follow the first way [2] [3]. These methods contributes to improve the first factor of the performance, i.e. the amount of duplication. Examples of rule rewriting are Aho-Ullman [1], magic sets [4] [15] [17], counting and reverse counting [4], Alexander [14], Kifer-Lozinskii [9], generalized magic sets [6], generalized counting [6] [16] magic counting [18], algebraic approach [7] and Horn clause transformation [11]. These rewriting methods contribute to improve the second influencing factor to the performance, and some of them also contributes other factors.

The object of this paper is to try to find common logical foundations of query transformations. This paper uses clause notation, and uses a term *query transformation* instead of rule rewriting to emphasize the similarity to the program transformation. Principles of query transformations are discussed and a conceptual procedure called clause replacement is proposed. Some transformation procedures are shown as examples of application of clause replacement, and properties of them are compared. Several other methods are also compared to these procedures.

2 Preliminaries and Basic Concepts

It is assumed that fundamental concepts of logic programs in [10] are known.

Definition 2.1 (Notations)

clause: A clause means a definite clause in this paper. It is denoted *h:-B* where h is an atom and B is a conjunction of atoms. A clause is also denoted *<h,B>* where h is an atom and B is a set of atoms. The latter notation is possible because the order of atoms in the body is meaningless.

model: A model means the least Herbrand model. It is also the least fixpoint.

prd(X): A predicate or a set of predicates of X. If X is an atom, prd(X) is its predicate. If X is a clause, prd(X) is its head predicate. If X is a set of clauses prd(X) is a set of their head predicates.

$C_h(P,X)$ and $C_b(P,X)$: Let P be a predicate or a set of predicates and X be a set of clauses. $C_h(P,X)$ and $C_b(P,X)$ are maximum subsets of X whose elements have predicate P or elements of set P in their heads and bodies, respectively.

$\Rightarrow$: A $\Rightarrow$ B means B is a logical consequence of A. Here, A is a set of clauses and B is a clause.

--, $\supset$: set difference, and set inclusion. ■

Definition 2.2 A *deductive database (DDB)* is a finite set of clauses. An *extensional database (EDB)* is a set of ground unit clauses, and an *intensional database (IDB)* is a set of other clauses. Thus, DDB = IDB $\cup$ EDB. ■

We assume prd(IDB) $\cap$prd(EDB) = $\emptyset$ without loss of generality, because a DDB can be easily transformed to satisfy this condition by adding predicates and clauses. The algorithms in this paper work correctly with a little modification even if this assumption is not used.

Definition 2.3 A *query* is denoted by a goal which is an atom. Let g' be a ground instance of g. Then, the *answer of the query* is the set G = {g'| DDB$\Rightarrow$ g'}. ■

Note that the definition of the goal with an atom does not practically restrict the expressive power of the query, because the goal with a conjunction of atoms can be mapped to a goal with an atom by adding a clause to the DDB. The following proposition is based on the property of the least model [10].

Proposition 2.1 Let DDB, G and M be a deductive database, its answer for goal g, and its least model, respectively. G and M may be infinite sets if functions are used in DDB. Let g' be a ground instance of g. g' $\in$ G iff g' $\in$ M. ■

The above definition is the basis of the bottom-up approach. The semantics of the deductive database is also defined by SLD resolution, and it is known that the SLD semantics is equivalent to the fixpoint semantics. The SLD semantics is the basis of the top-down approach.

Next, additional concepts used in subsequent chapters are defined.

Definition 2.4 A *dependency graph* of the DDB is a tuple [N,E]. Here, N is the set of predicates in the DDB, and E is a set of edges of the form (p,q) that means p appears in the head of a clause and q appears in its body. The transitive closure of edges, i.e., a path relation, is denoted $\rightarrow$*. A predicate, p, is *within reach of* a predicate, r, iff r$\rightarrow$*p. A clause F is *within reach of* a predicate, r, iff r$\rightarrow$*prd(F). ■

Definition 2.5 Let P be a set of predicates such that p $\in$ P and q $\in$ P iff p $\rightarrow$* q and q $\rightarrow$* p in the DDB. A *component* of the DDB is the set C_h(P,DDB), i.e. the set of clauses having elements of P in head. A predicate belongs to a component iff it appears in heads of clauses in the component. An atom belongs to a component iff its predicate belongs to the component. ■

Definition 2.6 Let N and M be components of the DDB. N is a *lower* component of M iff there exists a predicate, p, of M and a predicate, q, of N such that $p \rightarrow^* q$. The relation *lower* defines a partial order. M is an *upper* component of N iff N is a lower component of M. ■

Definition 2.7 Let B and C be clauses (or formulas). 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 the same. For instance, {a(X,Y), a(V,W)} = {a(X,Y)}. The least generalization of atoms whose predicates are the same is an atom C which is more general than these atoms, and there is no other atoms which are less general than C and more general than these atoms. ■

Definition 2.8 A clause <r,R'> is a sub-clause of <r,R> iff R' is a subset of R. Let B and C be clauses. B *subsumes* C iff there exists a sub-formula of C that is less general than B. It is obvious that if B subsumes C, then $\{B\} \Rightarrow C$. ■

3 Query Transformation

3.1 The Role of Query Transformations

The role of the query transformation is to obtain a query form that can be executed more efficiently than the original form by subsequent execution algorithms. How can we achieve this improvement? Because bottom-up methods compute the least model, if we obtain a smaller least model its computation is more efficient than the computation of the original least model.

This situation is explained as follows. When some arguments of a goal are bound, the top-down method computes a certain subset of the least model by restricting the search space in order to obtain the answer. The bottom-up method computes the whole least model in order to obtain the same answer. Query transformations can change the least model without changing the answer. Thus, the bottom-up method combined with query transformations computes a smaller least model than the original model, and this combination is almost equivalent to the top-down method in terms of what it computes. This situation is illustrated in Figure 1. The effect of the improvement is largest when the original least model is infinite and the transformed one is finite. Further discussion on the relationship of the bottom-up methods and top-down methods are found in [23].

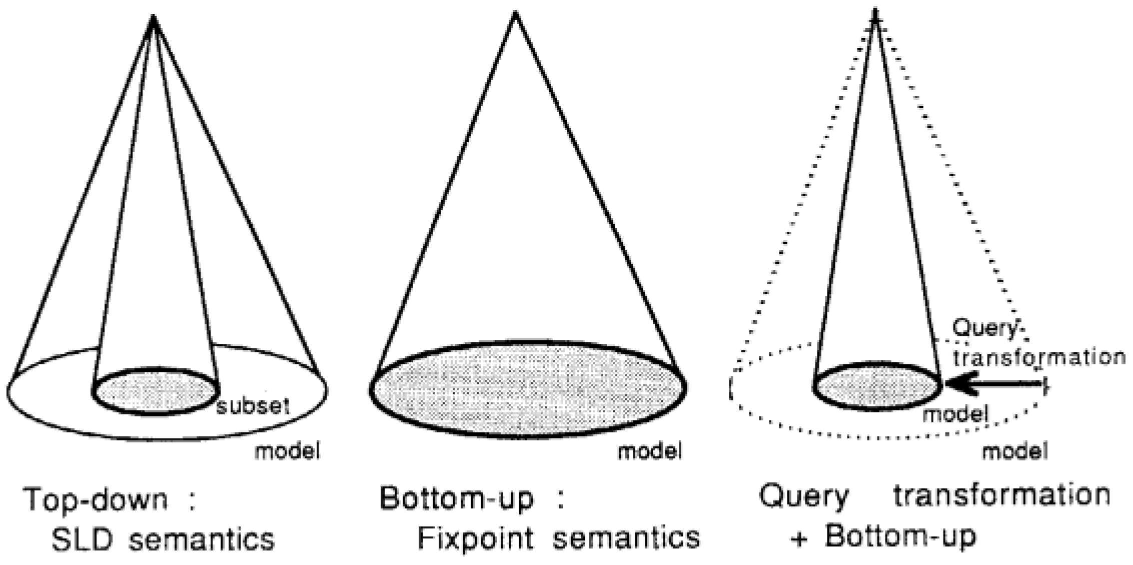

Figure 1 The role of query transformation

Let us consider effects of changing the least model to the performance factors in Chapter 1. If a transformation eliminates some clauses in the extensional database, the effect to the second factor is obvious because the set of relevant facts must be included in the transformed result. However, a transformation usually changes only the intensional database. In this case, changing the least model means changing the size of intermediate relations. Hence, if we obtain a smaller least model, there are ways to reduce the size of set of relevant facts used to compute the intermediate relations in subsequent bottom-up computation.

Once a smaller least model is obtained, the improvement of other factors can be realized by improving execution algorithms [2] [3] or further transforming the query [4] [18].

3.2. Equivalent Transformations

This section discusses principles of query transformations.

Definition 3.1 A *query transformation* is a mapping from the set of DDBs to itself.

Definition 3.2 Let DDB and DDB' be deductive databases and their answers for a goal g be G and G' respectively. DDB and DDB' is *goal equivalent with respect to g* iff G = G'. This is denoted DDB $\approx_g$ DDB'. ■

The relation "$\approx_g$" is an equivalence relation.

Definition 3.3 Let f_g be a transformation that is dependent on goal g. f_g is a *goal equivalent transformation with respect to g* iff for any DDB, f_g(DDB) $\approx_g$ DDB. ■

The concept of goal equivalence is weaker than the concept of equivalence used in program transformation of logic programs where the equivalence of least models is usually required [21]. The latter may be called model equivalence. Query transformations that are essentially model equivalent have been proposed [19]. Simple examples of model equivalent transformations are elimination of redundant clauses and tautology clauses. In this paper, *equivalent transformation* means *goal equivalent transformation* if not specified otherwise.

Definition 3.4 Let f_g be a transformation and DDB' = f_g(DDB). Let answers of DDB' and DDB for g be G' and G respectively. f_g is *complete with respect to g* iff G' $\supset$ G for any DDB. f_g is *sound with respect to g* iff G $\supset$ G' for any DDB. ■

A sound and complete transformation is an equivalent transformation.

The amount of EDB is considered large in a deductive database. Moreover, predicates in EDB can be handled efficiently by relational database techniques. Hence, clauses in the EDB are not usually transformed in query transformations. This fact gives the following definition.

Definition 3.5 An equivalent transformation f_g is called a *Horn clause transformation (HCT)* if it does not change clauses of the EDB. ■

The concept of HCT is essentially a generalization of equivalent transformation in relational databases [22], because a goal combined with an IDB corresponds to a query in the relational database. Let us consider ways to obtain query transformations that improve the performance of the successive execution. As discussed in the previous section, we can improve the performance by changing the least model to a smaller least model by a transformation. To achieve this effect, some information of the database must be discarded, i.e., some clauses of the database must be eliminated. However, simply discarding clauses usually results in inequivalent databases. Hence, some information contained in the discarded clauses should be preserved by adding other clauses. This consideration leads to the concept of clause replacement.

Definition 3.6 Let S be a set of clauses and DDB $\Rightarrow$ F for every F $\in$ S. Let E be a subset of DDB. A *clause replacement* is a transformation that maps DDB to (DDB -- E) $\cup$ S. ■

The following proposition is obvious.

Proposition 3.1 Clause replacement is sound with respect to any goal. ■

Clause replacement can be used as a guiding principle to obtain various query transformations, because it has following properties.

(1) Clause replacement is sound. Only completeness is required in order to obtain an equivalent transformation. Moreover, sometimes soundness is sufficient.

(2) Clause replacement reduces the size of the least model.

(3) Fundamental concepts such as resolution and subsumption in the first order logic are known to obtain logical consequences.

Three transformation procedures are shown in the following chapters as examples of equivalent transformations based on clause replacement. One procedure is based on resolution and the other two are based on subsumption.

4. Horn Clause Transformation by Partial Evaluation (HCT/P)

This chapter discusses a transformation based on the concept of resolution.

Definition 4.1 Let p be a predicate and C be an element of C_b(p,DDB). Let C' be a resolvent of C and F $\in C_h$(p,DDB) on p (by the most general unifier). Clearly, {C} $\cup$ C_h(p,DDB) $\Rightarrow$ C' if C' exists. Let S be a set of all such C's. A *clause replacement by* *resolution* is a transformation that maps DDB to $(\mathrm{DDB}--(\{C\}--(\{C\}\cap C_h(p,\mathrm{DDB})))) \cup S$. ■

Note that the result of the transformation is $\{\mathrm{DDB}--\{C\}\}\cup S$ if C is not an element of $C_h(p,\mathrm{DDB})$ and is $\mathrm{DDB}\cup S$ otherwise. The concept of a clause replacement by resolution is similar to the unfold transformation proposed in [21].

Proposition 4.1 A clause replacement by resolution is an equivalent transformation with respect to any goal except for the one having predicate symbol p.

Proof The transformation is sound by proposition 3.1. The completeness can be easily shown by comparing SLD trees of DDB and DDB'. ■

Because $\mathrm{prd}(\mathrm{IDB}) \cap \mathrm{prd}(\mathrm{EDB}) = \varnothing$, $C_h(p,\mathrm{DDB}) = C_h(p,\mathrm{IDB})$ if $p \in \mathrm{prd}(\mathrm{IDB})$. Hence, the following procedures can be defined based on clause replacement by resolution.

Definition 4.2 A clause replacement procedure *res(p,IDB)*:

```
res(p,IDB); /* p is a predicate. */
 R := $\varnothing$;
 for every pair $C_b \in C_b(p,IDB)$ and $C_h \in C_h(p,IDB)$ do;
    R := R $\cup$ {resolvents of $C_b$ and $C_h$ on p};
 end;
 IDB' := (IDB -- ($C_b$(p,IDB) -- $C_h$(p,IDB))) $\cup$ R;
 return IDB';
end.        ■
```

Definition 4.3 Partial evaluation procedure *p_eval(g,IDB,T)*:

T is a given set of predicates called a terminal predicate set.

```
p_eval(g,IDB,T);
  IDB' := IDB -- {clauses that are not within reach of g};
  repeat ;
     select p such that p $\in$ (prd(IDB') -- T -- {prd(g)}) ;
     IDB' := res(p,IDB');
  until no such p;
```

```
    IDB' := IDB' -- {clauses that are not within reach of g};
    return IDB';
end. ■
```

Proposition 4.2 $f_g(DDB) \equiv$ p_eval(g,IDB,T) $\cup$ EDB is an HCT if it terminates and T$\supset$prd(EDB).

Proof p_eval is the composition of clause replacement by resolution and the deletion of clauses which are not within reach of g. Because the deletion is an equivalent transformation, p_eval(g, DDB,T) is equivalent transformation by propositions 4.1. Because T $\supset$ prd(EDB), p_eval(g,DDB,T) $\approx_g$ p_eval(g,IDB,T) $\cup$ EDB ■

HCTs can be obtained by properly giving the terminal predicate set, T. An HCT given by this procedure is called an *HCT by partial evaluation (HCT/P)*. In principle, the number of elements of T should be as small as possible while assuring the termination of the procedure. The termination is guaranteed by selecting at least one predicate for T from each cycle in the dependency graph. The model after HCT/P is smaller than the original model, because some predicates are eliminated by the transformation. Queries can be simplified using HCT/P. For instance, a non-recursive query can be reduced to one that contains only predicates in the EDB except for the goal predicate. It is also possible to express a query only by self-recursive clauses (i.e., a predicate in the head appears in its body) and non-recursive clauses. The following simple example illustrates how HCT/P works.

Example 4.1

goal: p(X,Y)

IDB: p(X,Y):-p1(X,Y)

p(X,Y):-q(X,Z),p2(Z,Y)

q(X,Y):-p(X,Y)

p1 and p2 are elements of prd(EDB). ■

It is easy to see that q can be eliminated. The result of p_eval(p(X,Y),IDB,{p,p1,p2}) is:

p(X,Y):-p1(X,Y)

p(X,Y):-p(X,Z),p2(Z,Y).

Properties of HCT/P and its variations are discussed in [11]. The application of HCT/P to optimization of multiple queries is found in [20]. The resolution may be regarded as a generalization of substitution of the relational algebra expression for a (intermediate) relation symbol. A similar procedure is proposed for function-free recursive queries expressed by relational algebra based on the this algebraic substitution in [7].

5 Horn Clause Transformation by Substitution (HCT/S)

This chapter discusses a transformation procedure based on subsumption. Definition 2.8 of subsumption involves two concepts, substitution and sub-clause. This chapter deals with substitution.

Definition 5.1 Let C' be a clause obtained from C by ground substitution of variables. Clearly, $\{C\} \Rightarrow C'$. A *clause replacement by substitution* is a transformation that maps DDB to $(DDB\text{--}\{C\}) \cup \{C'\}$. ■

Because this transformation is sound by proposition 3.1, an HCT is obtained by considering the completeness of the transformation. First, a procedure that transforms a component to its equivalent is defined.

Definition 5.2 A clause replacement procedure *Comp/S(p,Comp)*;

```
  Comp is a component, and p is an atom of the component
Comp/S(p, Comp);
  I := {p}; D := { }; Comp' := { };
  repeat;
    select $q \in I$;
    C' := { };
   for every $C \in$ Comp such that prd(C) = prd(q) do;
      if (head of C)$\theta$ = q then  C' := C' $\cup$ {C$\theta$};
   end;
    Comp' := Comp' $\cup$ C';
    S := {atoms of the component which appear in bodies in C'};
    D := $D \cup \{q\}$
    I := {the least generalizations of atoms in $I \cup S \cup D$} -- D;
  until I = { };
  return Comp';
end.        ■
```

Comp/S always terminates, because there are only finite number of atoms that are more general than an atom and atoms in I become more general in each loop. Comp/S may generate redundant clauses which are less general than other clauses. These clauses can be easily eliminated.

Proposition 5.1 Let Comp' = Comp/S(p,Comp). Let E be a subset of DDB whose elements are in lower components than Comp. For any E, Comp'$\cup$E $\approx_p$ Comp$\cup$E

Proof Soundness is clear by proposition 3.1. The completeness is proved by inspecting the SLD tree of DDB, because for every subgoal in the tree which belongs to the component there exists a more general atom used for the substitution in Comp/S. ■

An HCT is recursively defined using Comp/S.

Definition 5.3 A transformation procedure *HCT/S(g,IDB)*;

```
HCT/S(g,IDB);  /* g is an atom. */
   Comp := The component of IDB to which prd(g) belongs;
   IDB' := Comp/S(g,Comp);
   for every directly lower component Lcomp of Comp in IDB do;
      for every atom p of Lcomp that appears in a body of Comp' do;
         IDB' := IDB' $\cup$ HCTS(p, Lcomp$\cup$ lower components of Lcomp in IDB);
      end;
   end;
   return IDB';
end.        ■
```

The result of HCT/S may include redundant clauses that are less general than other clauses. They can be easily eliminated during or after HCT/S. The following proposition is obtained from proposition 5.1.

Proposition 5.2 $f_g(DDB) \equiv HCT/S(g,IDB) \cup EDB$ is an HCT. ■

The following example illustrates how HCT/S works.

Example 5.1 (ancestor)

Goal: ancestor(taro,hanako)

IDB: (r1) ancestor(X,Y):-parent(X,Y)

(r2) ancestor(X,Y):-parent(X,Z),ancestor(Z,Y)

The EDB includes clauses for the parent. ■

The IDB has only one component. The first loop in Comp/S generates following clauses.

(r1)' ancestor(taro, hanako):-parent(taro, hanako)

(r2)' ancestor(taro, hanako):-parent(taro,Z),ancestor(Z,hanako)

Because ancestor(Z,hanako) is more general than ancestor(taro,hanako) it continues to generate,

(r1)" ancestor(Z, hanako):-parent(Z, hanako)

(r2)" ancestor(Z, hanako):-parent(Z,Z1),ancestor(Z1,hanako)

The procedure terminates with above 4 clauses. The first couple of clauses can be discarded as redundant, and the final result are the second couple of clauses. The result shows that equivalence is preserved by the substitution of the second argument, but not by the substitution of the first.

Example 5.2. (common ancestor)

Goal: comm_anc(X,taro,jiro)

IDB: (r3) comm_anc(X,Y,Z):-ancestor(X,Y),ancestor(X,Z)

ancestor and parent are same as the above. ■

This IDB consists of two components: [{comm_anc},{r3}] and [{ancestor},{r1,r2}].

First, HCT/S transforms r3 to:

comm_anc(X,taro,jiro):-ancestor(X,taro),ancestor(X,jiro)

by Comp/S. This is goal equivalent to the original clause. Next, HCT/S applies COMP/S to the component of ancestor with two atoms, ancestor(X,taro) and ancestor(X,jiro)

The result of HCT/S(ancestor(X,taro),{r1,r2}) is:

ancestor(X,taro):-parent(X,taro)

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

The result of HCT/S(ancestor(X,jiro),{r2,r3}) is:

ancestor(X,jiro):-parent(X,jiro)

ancestor(X,jiro):-parent(X,Z),ancestor(Z,jiro)

The result of HCT/S(comm_anc(X,taro,jiro),IDB) is the collection of these five clauses.

HCT/S is a generalization of the distribution of selections for non-recursive queries expressed by relational algebra and for transitive closure [1]. Note that the distribution of selection (by constant) is always possible for non-recursive queries, but it may not be possible for recursive queries as shown in example 5.1. Similar procedures were proposed in [7] for simple function-free recursive queries expressed in relational algebra. Similar effects can be obtained using a rule/goal graph [9]. HCT/S is the most general of these procedures because it does not restrict DDBs to Datalog, and because its output is a set of clauses and can be combined with any other procedures.

6 Horn Clause Transformation by Restrictor (HCT/R)

Another transformation based on the subsumption is discussed in this section. Arguments of atoms are usually not shown in this chapter. Hence, an atom whose predicate is r is also expressed as r. The basic idea of this transformation is as follows. Because a clause <r,R> subsumes <r, {r*}∪R>, the latter is a logical consequence of the former. Hence, a transformation that replaces <r,R> by <r, {r*}∪R> is a clause replacement.

Definition 6.1 Let *S* be a subset of prd(DDB) called a *restricted predicate set.* Let r* be a newly introduced predicate that corresponds to $r \in S$. The arity of predicate r* is the same as the arity of predicate r. Let us consider a transformation consisting of two steps.

(1) Insert clauses whose heads have such r*s. This is an equivalent transformation.

(2) Replace all clauses in DDB as follows:

Let C be a clause <r,R> in DDB, where r is an atom and R is a set of atoms. If $\mathrm{prd}(r) \in S$, replace this clause by a clause, C'= <r, $\{r^*\} \cup R$>, where r* is an atom and arguments of r* are the same as those of r in the head. If $\mathrm{prd}(r) \notin S$, C' is the same as C.

The second step is a clause replacement and is sound with respect to any goal. Predicate (or atom) r is called a *restricted predicate* (or *atom*) and predicate (or atom) r* is a *restrictor predicate* (or atom). A restrictor predicate (or atom) may be simply called a ***restrictor.*** C' is a *restricted clause* and a clause whose head is a restrictor is a *restrictor clause.* If this transformation is an HCT, it is called *HCT by restrictor (HCT/R).* ■

Let us discuss conditions for restrictor clauses to make this transformation equivalent. Let DDB' be the transformed result. First, consider a restricted clause <g',$\{g^*\} \cup R$> where $g=g'\theta$ for goal g. If $\mathrm{DDB} \Rightarrow g^*\theta$, the clause is considered to be essentially equivalent to the original clause, <g',R> with respect to g. This consideration gives the following definition.

Definition 6.2 Let g be a goal. Suppose prd(g) is an element of a restricted predicate set, S. An *initial (restrictor) clause* is defined as a unit clause that consists of g* whose arguments are the same as g. If prd(g) is not an element of S, there are no initial clauses. ■

Next, consider restricted clauses, <p,$\{r\} \cup P$> and <r,$\{r^*\} \cup R$> where $p \notin S$ and $r \in S$. Their resolvent is <p,$\{r^*\} \cup R \cup P$>. The resolvent of the original clauses is <p,$R \cup P$>. If DDB'$\Rightarrow$<r*,P>, we get <p,$P \cup R \cup P$> from <p,$\{r^*\} \cup R \cup P$> and this is logically equivalent to <p,$R \cup P$>. Thus, we obtain the following definition.

Definition 6.3 Let C' be a restricted clause, <p,{r}∪P>, where r is a restricted atom and P is a set of atoms. Let P' be a subset of P. A *candidate equivalent restrictor clause (CERC)* C* is a clause <r*,P'> where the arguments of atom r* are equivalent to those of r in C'. If there are more than one atom in C' whose predicates are the same, a CERC is defined for each atom. ■

One more concept, acyclicity, is necessary to guarantee the equivalence, because the discussion for definition 6.3 may involves tautological cycles.

Definition 6.4 A *CERC set* corresponding to a restricted clause, C', is the set of clauses obtained by selecting a CERC for every restricted atom in the body of C'. A relation "→" is defined for each element C* of the CERC set as follows. If p* = prd(C*) and a restricted predicate, r, is in the body of C*, then p→r. Here, a predicate that appears more than once in the body of C' is treated as if each instance has a different predicate. A *CERC graph* is a directed graph obtained by collecting all these relation instances of elements of the CERC set. A CERC set is *acyclic* iff its CERC graph does not have a cycle. ■

For instance, consider a restricted clause, r :-r*,p,q where p and q are restricted atoms. A possible CERC set corresponding to this clause is

p*:-r*,q

q*:-r*,p.

This set is not acyclic because its CERC graph has a cycle as shown in Figure 2.

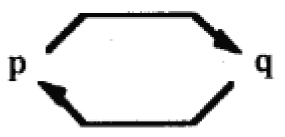

Figure 2 An Example of CERC Graph

The following CERC set corresponding to the same restricted clause is acyclic.

p*:-r*,q

q*:-r*.

The next proposition gives a sufficient condition for an HCT.

Proposition 6.1 Let DDB be a deductive database. Let DDB' be a set of clauses that consists of restricted clauses defined by definition 6.1, an initial clause defined by definition 6.2, and all elements of acyclic CERC sets each of which corresponds to a diffrent restricted clause defined in definition 6.4. Then, DDB $\approx_g$ DDB'.

Proof (sketch)

Soundness is clear by proposition 3.1 because the first step is equivalent and the second step is sound.

Completeness is proved by comparing SLD trees of DDB and DDB' as follows. The roots of these trees are g. If for every node in the DDB tree there exists a node in the DDB' tree that corresponds to that one, then every success node in DDB has its corresponding success node in the DDB' tree and the transformation is complete. This can be proved by using mathematical induction to show that every subtree of DDB' whose root has a restrictor predicate has at least one success branch. The condition of acyclicity is necessary to give a computation rule in the SLD refutation. ■

Example 6.1 The goal and the DDB are the same as example 5.1. ■

Let the restrictor set be {ancestor}. Restricted clauses are,

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

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

The initial clause is,

ancestor*(taro,hanako).

A CERC set that has maximum number of atoms in body is

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

Because this CERC set is acyclic, the set of above five clauses is equivalent to the original IDB with respect to ancestor(taro,hanako).

Example 6.2 The goal and the DDB are the same as example 5.2.

Let the restrictor set be {ancestor,comm_anc}. Restricted clauses are,

comm_anc(X,Y,Z):-comm_anc*(X,Y,Z),ancestor(X,Y),ancestor(X,Z)

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

The initial clause is

comm_anc*(X,taro,jiro).

The following is the CERC set, which have maximum number of atoms in body, corresponding to the first restricted clause.

ancestor*(X,Y):-comm_anc*(X,Y,Z),ancestor(X,Z)

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

Another CERC set corresponding to the third restricted clause is

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

The first CERC set is not acyclic. A possible acyclic CERC set is as follows, although there are other alternatives.

ancestor*(X,Y):-comm_anc*(X,Y,Z)

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

The modified set of CERCs, restricted clauses and the initial clause is equivalent to the original IDB.

Although the transformation given above is conceptually simple, it may not be very useful in practice because usually the resulted set of clauses is not bottom-up evaluable [5] even if the original set is. The result of example 6.2 shows this problem. Because the initial clause contains a variable, the first argument of comm_anc* may take any value of Herbrand universe. Moreover, the first argument of ancestor* has the same problem. Hence, the least model of this clause set may be large, and the usual bottom-up method can not handle this clause set. There are two ways to solve this problem. One is modifying the bottom-up algorithm to enable to handle this kind of variables. The other is modifying the transformation. The latter is discussed below.

The principal problem is that the restrictor have free arguments. The problem can be solved by reducing the number of arguments of the restrictor. The concept of adornment used for the magic set [4] is useful to this modification. Let us consider an adorned version of restrictor called partial restrictor. A partial restrictor is denoted $r*\_{b/f}$ where b/f is a sequence of "b" and "f" and called an adornment. Informally, "b" stands for bound and "f" stands for free. The length of the adornment is equivalent to the arity of corresponding predicate, r. The number of argument of $r*\_^{b/f}$ is the number of "b"s in its adornment, and the positions of "b"s indicate which argument of r* remain for $r*\_^{b/f}$.

Definitions from 6.1 to 6.4 can be modified using partial restrictors instead of restrictors, and proposition 6.1 also holds for the modified transformation. This modification also contributes to improve the third influencing factor to the performance in chapter 1. The details of the modifications and how to obtain an optimal transformation algorithm are discussed and an optimal algorithm is given in [12].

The result of modified HCT/R for example 6.1 is the same except for the attachment of adornment "bb". A possible result for example 6.2 is as follows.

comm_anc*_^{fbb}(taro,jiro)

ancestor*_^{fb}(Y):-comm_anc*_^{fbb}(Y,Z)

ancestor*_^{fb}(Z):-comm_anc*_^{fbb}(Y,Z)

ancestor*_^{fb}(Y):-ancestor*_^{fb}(Y)   /* may be eliminated because this is tautology. */

comm_anc(X,Y,Z):-comm_anc*_^{fbb}(Y,Z),ancestor(X,Y),ancestor(X,Z)

ancestor(X,Y):- ancestor*_^{fb}(Y),parent(X,Y)

ancestor(X,Y):-ancestor*_^{fb}(Y),parent(X,Z),ancestor(Z,Y).

HCT/R works correctly for any type of query, including mutually recursive queries. Algorithms similar to HCT/R have been proposed in the literature. Examples are *magic sets (MG)* [4] [15] [17], Alexander [14] and *generalized magic sets (GMG)* [6]. It is easy to see that these methods satisfy the condition for HCT/R, i.e., the condition of (modified) proposition 6.1, with little or no modification. MG gives the same (sometimes slightly different) result as HCT/R, although it can be used only for special types of queries such as linear or nested queries. The GMG can be used for a broader class of queries, and its result is similar to the result of HCT/R. The principal difference between GMG and HCT/R are as follows.

(a) The guiding principle of GMG is the concept of *sideways information passing (sip)* which is based on the procedural semantics. The guiding principle of HCT/R is clause replacement which is based on the declarative semantics. These two represent two complementary views of query transformations.

(b) Syntactically, adornment of both restrictor and restricted predicates is essential in GMG, while adornment is not essential but is used to improve HCT/R. Adornment of restrictor predicates is discussed in this paper. The restricted predicate can also be adorned in HCT/R by modifying the definitions. However, adornment of restricted predicates results in larger least models, and its effects on performance need more discussions.

(c) The separation of the transformation from the execution algorithm is more complete in HCT/R. Hence, the condition for HCT/R in proposition 6.1 is the weakest one so far known.

(d) A method, generalized supplementary magic set, was proposed to further improve the GMG. The same improvement can be easily applied to HCT/R. The concept of *sip* also leads to the counting method, but *clause replacement* does not.

7 A Comparison of Horn Clause Transformations

This chapter discusses some properties of HCTs discussed in previous chapters. First, the logical relationship of HCTs is shown in Figure 2.

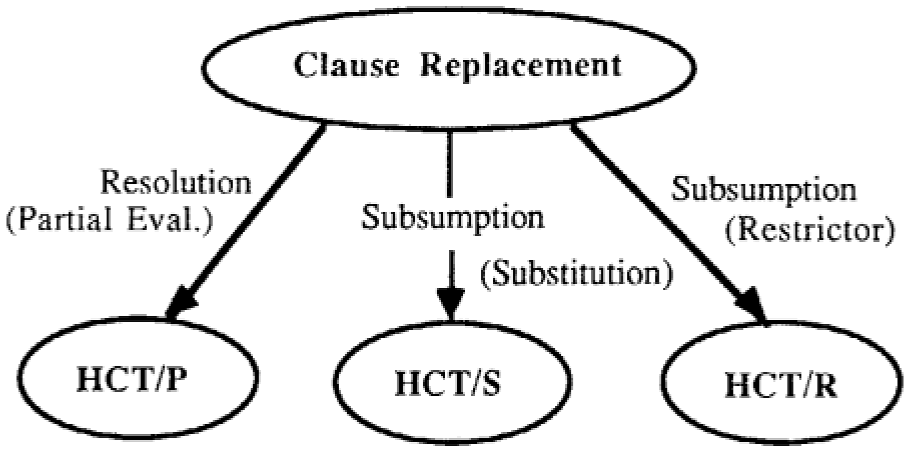

Figure 3 Relationship of Horn Clause Transformations

The performance gain by HCTs may be as large as several orders of magnitudes depending on algorithms, the contents of DDB, and the goal [5]. HCTs change least models by changing the database different ways. The changes of the database by HCTs are as follows.

(1) They eliminate clauses which are not within reach of a given goal.

(2) They eliminate clauses which are within reach of the goal but do not contribute to the answer.

(3) They eliminate predicates, and thus reduce the number of predicates.

(4) They propagate constants in a goal or clauses, and thus transform clauses to less general clauses.

(5) They introduce new predicates that can restrict the computation space using constants in a goal or clauses during the successive execution phase.

These changes always result in smaller least models except for the last one. The last one does not always guarantee smaller models because introducing new predicates results in larger models. However, the gain is usually much larger than the loss. Each HCT changes the database differently. HCT/P performs changes 1, 2 and 3. HCT/S performs changes 1, 2, and 4. HCT/R performs changes 1 and 5.

Change 1 is performed by every HCT. Change 2 is obtained by HCT/P and HCT/S. However, change 2 by HCT/P is smaller because it does not propagate constants. If there are many constants in the clause heads, change 2 by HCT/S can be very effective.

Change 3 by HCT/P results in smaller least models. However, if there are many clauses that have same head predicates, change 3 increases the number of clauses in the database. This may result in inefficient computation even if the least model is smaller.

Restricting computing space using constants can be performed either by the change 4 (HCT/S) or 5 (HCT/R). Change 4 is possible only for special cases and change 5 is always possible as illustrated in examples 5.1 and 6.1. However, if change 4 is possible the performance improvement by it is larger than by change 5, because change 4 does not introduce new predicates. An interesting example of this situation is the comparison of the results of example 5.2 and 6.2.
The (modified) result of example 6.2 is shown below excluding the tautology clause.

comm_anc*_^{fbb}(taro,jiro)

ancestor*_^{fb}(Y):-comm_anc*_^{fbb}(Y,Z)

ancestor*_^{fb}(Z):-comm_anc*_^{fbb}(Y,Z)

comm_anc(X,Y,Z):-comm_anc*_^{fbb}(Y,Z),ancestor(X,Y),ancestor(X,Z)

ancestor(X,Y):- ancestor*_^{fb}(Y),parent(X,Y)

ancestor(X,Y):-ancestor*_^{fb}(Y),parent(X,Z),ancestor(Z,Y).

The least model of this clause set is larger than the one obtained for example 5.2 by HCT/S. Hence, HCT/S is more effective than HCT/R for this example. The above result of HCT/R may be improved by applying HCT/P, because partial restrictors are not recursive. By applying HCT/P to this clause set, we obtain the following set of clauses which is same as the result of example 5.2.

comm_anc(X,taro,jiro):-ancestor(X,taro),ancestor(X,jiro)

ancestor(X,taro):-parent(X,taro)

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

ancestor(X,jiro):-parent(X,jiro)

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

The above discussions are summarized in Table 1.

Table 1 Comparison of HCTs

HCTs \ Changes	1	2	3	4&5
HCT/P	○	△	○	
HCT/S	○	○		△
HCT/R	○			○

It is concluded from Table 1 that there is no single HCT which performs all changes discussed above. Because each change has its advantage, it is not possible to obtain ideal optimized result for all queries by a single HCT. This fact indicates that a combined use of HCTs is more effective than a single HCT. It also indicates that algorithms similar to HCTs discussed in previous chapters can be improved by combining them with other algorithms. If we can neglect the transformation cost, the possible best combination is as follows.

(1) Apply HCT/P to simplify a query.

(2) Apply HCT/S.

(3) If there remain constants that cannot be distributed by HCT/S, then apply HCT/R.

(4) Apply HCT/P again to eliminate inessential restrictors.

More detailed comparisons and experiments are necessary to obtain the best method.

8. Conclusions

This paper discussed principles of query transformations, and proposed a conceptual procedure, clause replacement, as a principle. Clause replacement improves the performance of query processing by reducing the size of the least Herbrand models. Three transformations, HCT/P, HCT/S and HCT/R, were shown as examples of clause replacement. The relationships of other methods with these transformations were discussed, and it was shown that several methods such as the simplification of queries, the distribution of selection, and magic set can be reformulated and generalized using the concept of clause replacement. A comparison of HCTs indicates that combined use of HCTs is desirable to handle various type of queries. Because both input and output of HCTs are sets of definite clauses, it is easy to combine these methods to obtain optimized form of a query.

There is an important class of query transformations, i.e., counting and its variations, which is not discussed in this paper. Because these transformations are based on magic set and generalized magic set, they can be considered as further transformations of the result of HCT/R. However, definitions of equivalent transformations must be extended to deal with them, because they use supplementary arguments.

The database is restricted to a set of definite clauses in this paper. Query transformations can be also applied to stratified databases that allow negative literals in the clause body. Generalization of HCT/P and HCT/S for stratified databases is straightforward, because they can preserve the structure of queries. However, direct extension of HCT/R may result in a non-stratified database. Hence, the semantics of query transformation must be extended to generalize HCT/R.

Query transformations are used as part of query compilation in the KBMS PHI system in the FGCS project [8]. HCT/P and HCT/S were implemented although the implemented algorithms are little different from HCTs in this paper, and HCT/R is being implemented.

References

[1] Aho, A.V. and Ullman, J.D., Universality of Data Retrieval Languages, 6th ACM POPL, 1979.

[2] Balbin, I., and Ramamohanarao, K.A., A Generalization of the Differential Approach to Recursive Query Evaluation, JLP, Vol. 4, pp259-262, 1987.

[3] Bancilhon, F., Naive Evaluation of Recursively Defined Relations, in (eds.) Brodie, et al., On Knowledge Base Management Systems, Springer, 1985.

[4] Bancilhon, F., et. al., Magic Sets and Other Strange Ways to Implement Logic Programs, 5th ACM PODS, Mar. 1986.

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

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

[7] Ceri, S., et. al., Translation and Optimization of Logic Queries: The Algebraic Approach, 11th VLDB, Aug. 1986.

[8] Itoh, H., Research and Development on Knowledge Base Systems at ICOT, 12th VLDB, Aug. 1986.

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

[10] Lloyd, J.W., Foundations of Logic Programming, Springer-Verlag, 1984.

[11] 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).

[12] Miyazaki, N., Yokota, K., Haniuda,H., and Itoh, H., Horn Clause Transformation by Restrictor in Deductive Databases, ICOT TR-407, 1988.

[13] Nishio, S. and Kusumi, Y., Evaluation Methods for Recursive Queries in Deductive Databases, J. IPS Japan, Vol. 29, No. 3, 1988. (in Japanese)

[14] Rohmer, J., Lescoeur, R., and Kersit, J.M., The Alexander Method, New Generation Computing, Vol. 4, Ohmsha and Springer, 1986.

[15] Sacca, D. and Zaniolo, C., On the Implementation of a Simple Class of Logic Queries for Databases, ACM PODS, 1986.

[16] Sacca, D. and Zaniolo, C., The Generalized Counting Method for Recursive Logic Queries, ICDT, 1986.

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

[18] Sacca, D. and Zaniolo, C., Magic Counting Method, ACM SIGMOD, 1987.

[19] Sagiv, Y., Optimizing Datalog Programs, ACM PODS, 1987.

[20] Sakama, C. and Itoh, H., Partial Evaluation of Queries in Deductive Databases, WS on Partial Evaluation and Mixed Computation, 1987.

[21] Tamaki, H. and Sato, T, Unfold/Fold Transformations of Logic Programs, 2nd ICLP, 1984.

[22] Ullman, J.D., Principles of Database Systems, (2nd edition) Chapter 8, Computer Science Press, 1982.

[23] Vieille, L., From QSQ towards QoSaQ: Global Optimization of Recursive Queries, International Conf. on Expert Database Systems, 1988.