ICOT Technical Report: TR-407

TR-407

Horn Clause Transformation by Restrictor in Deductive Databases

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

June, 1988

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

Horn Clause Transformation by Restrictor in Deductive Databases

June 1988

Nobuyoshi Miyazaki*, Kazumasa Yokota **,

Hiromi Haniuda* and Hidenori Itoh**

* Oki Electric Industry Co., Ltd.

**Institute for New Generation Computer Technology

Abstract

Several methods have been proposed to improve the performance of bottom-up evaluation by query transformation (or rule rewriting) in the deductive database field. One way to realize this improvement is to introduce new predicates such as magic predicates to be used as filters. This is considered as one of the most promising ways to realize efficient query processing. Most algorithms in this category are formulated based on the procedural semantics of the top-down evaluation, and their logical meaning is difficult to recognize. This paper proposes a method called Horn clause transformation by restrictor (HCT/R) based on the declarative semantics. HCT/R is formulated as an equivalent transformation that maps the least Herbrand model to a smaller model. It is most general in this category and can be considered as a logical foundation of methods in this category. Optimization of this transformation is also discussed and an optimal algorithm is proposed.

1. Introduction

There are two major ways of processing queries in deductive databases [4]. One is the top-down method based on SLD semantics. The other is the bottom-up method based on fixpoint semantics. When some arguments of a goal are bound, the top-down method computes a certain subset of the least Herbrand model by restricting the search space in order to obtain the answer. The bottom-up method computes the whole least Herbrand model in order to obtain the same answer. Therefore, the bottom-up method is not efficient although it is more suitable to use relational database operations.

Several algorithms have been proposed to overcome the problem of the bottom-up methods by transforming queries before bottom-up evaluation. Examples of these methods are distribution of selections [2] or push of selection conditions [6], and magic set (MG) [3] [9] [10] and generalized magic set (GMG) [5]. The former two methods are simpler but can be applied to only special cases of queries, while the MG and GMG are more complex but can be used for broader class of queries. MG and GMG that use new predicates as a kind of filters are considered as one of most efficient ways to process queries by bottom-up evaluation. These methods are formulated based on the procedural semantics and explained as the simulation of top-down evaluation by bottom-up evaluation. Logical meaning of MGs is difficult to understand because the declarative semantics of them are not obvious.

This paper proposes a transformation called Horn clause transformation by restrictor (HCT/R). HCT/R is formulated based on the declarative semantics, and can be considered as a logical foundation of MGs as well as their generalization. HCT/R is formulated as an equivalent transformation that maps the least Herbrand model to smaller model while preserving the answer set. A simple optimal HCT/R algorithm is also proposed. Chapter 2 discusses the fundamental concepts of deductive databases. HCT/R is proposed in chapter 3 and its modification for practical implementation is discussed in chapter 4. The optimization of HCT/R and relationship with MGs are discussed in chapter 5 and chapter 6, respectively.

2. Deductive Databases and Equivalent Transformations

The basic concepts of deductive databases and query transformation are discussed in this chapter. It is assumed that fundamental concepts of logic programs in [7] 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 set of atoms. A clause is also denoted h:-B where h is an atom and B is a conjunct of atoms.

model: A model means the least Herbrand model.

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.

$\Rightarrow$: Let A be a set of clauses and B be a clause. A $\Rightarrow$ B means B is the logical consequence of A.

■

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. ■

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 conjunct of atoms can be mapped to a goal with an atom by adding a clause to the DDB. A goal combined with the related subset of the IDB corresponds to a query in relational databases. The following proposition is the basis of query processing [7].

Proposition 2.1 Let DDB, G and M be a deductive database, its answer for goal g, and its model. 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 $g' \in M$. ■

Because a model of DDB may be large and even infinite, the direct computation of a model by a bottom-up method is not efficient. Therefore, the concept of equivalent transformation that can reduce the size of models is defined.

Definition 2.4 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* (or simply *equivalent*) *with respect to g* iff G = G'. This is denoted $DDB \approx_g DDB'$. ■

The relation "$\approx_g$" is transitive, reflexive and symmetric, i.e., it is an equivalence relation.

Definition 2.5 Let f be a transformation from a set of all deductive databases to itself. f is an *equivalent transformation with respect to g* iff $f(DDB) \approx_g DDB$ for any DDB. ■

The term "with respect to g" may be omitted if it is obvious with respect to which goal the transformation is equivalent. The amount of EDB is considered large in a deductive database, and predicates in EDB can be handled efficiently by relational database techniques. Therefore, usually it is not necessary to transform clauses in the EDB. This fact gives the following definition.

Definition 2.6 Let $DDB = IDB \cup EDB$ be a deductive database. An equivalent transformation f is a *Horn clause transformation (HCT)* iff there exists a transformation h, and f is expressed as:

$f(DDB) = h(IDB) \cup EDB$, for any DDB.

Transformation h may also be called an HCT. ■

Definition 2.7 Let f be a transformation. Let answers of DDB and f(DDB) for g be G and G' respectively. f is *complete with respect to g* iff $G' \supset G$ for any DDB. f is *sound with respect to g* iff $G \supset G'$ for any DDB. ■

A sound and complete transformation is an equivalent transformation.

Now we have defined fundamental concepts of deductive databases and their transformations. Let us consider how an effective transformation can be obtained. First, some clauses of DDB should be discarded in order to reduce the size of model. However, the essential information must be preserved in order to assure the same answer. The simplest way to achieve both requirements is to replace clauses by their logical consequences. This consideration gives the following definition.

Definition 2.8 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. ■

Proposition 2.2 A clause replacement is sound with respect to any goal.

Proof It is clear that the model of DDB$\cup$S is same as the model of DDB. Therefore, the result of a clause replacement has a model that is a subset of the model of DDB. Hence, the resulted answer is the subset of the original answer by proposition 2.1. ■

The clause replacement is one of the most important principles of query transformations, because various transformations are obtained by applying it using fundamental principles of first order logic. For instance, Horn clause transformation by partial evaluation [8] is obtained based on the clause replacement using resolution. Another example is the distribution of selection for transitive closure [2]. This method as well as its extensions [6] can be considered as special cases of methods based on the clause replacement using ground substitution [1].

One more definition of a simple transformation is used in the following sections.

Definition 2.9 Let p be a predicate which does not appear in DDB, and S be a set of clauses such that prd(S) = p. A *clause insertion* is a transformation that maps DDB to DDB $\cup$ S. ■

The following proposition is obvious.

Proposition 2.3 A clause insertion is an equivalent transformation with respect to g if prd(g) $\neq$ prd(inserted clauses). ■

3 Horn Clause Transformation by Restrictor (HCT/R)

An HCT based on the clause replacement using subsumption is discussed in this section. Subsumption is one of the fundamental concepts in first order logic. Arguments of atoms are not shown in this section. Therefore, an atom whose predicate is r is also expressed as r.

Definition 3.1 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 $\{B\} \Rightarrow C$. ■

The basic idea of the transformation is as follows. Let <r,R> be a clause in DDB. Replace this clause by a clause, <r, {r*}∪R>, that is subsumed by <r,R>. The subset of the model that corresponds to predicate, r, may vary depending on how r* is given, but it should never be larger than the original subset.

Definition 3.2 Let DDB = IDB∪EDB, and *S* be a subset of prd(IDB) called a *restricted predicate set*. The following transformation does not change clauses in EDB. Let r* be a newly introduced predicate that corresponds to r ∈ S. The arity (i.e., the number of arguments) 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.

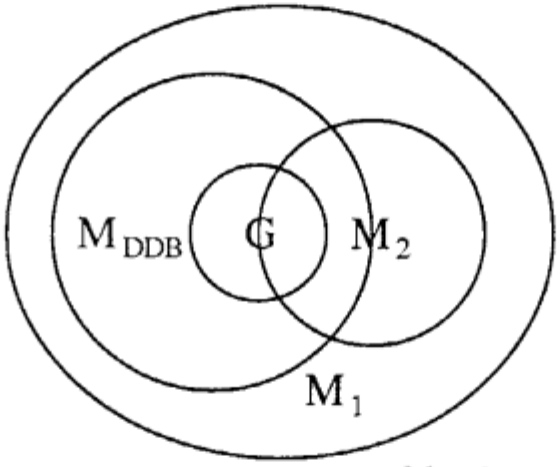
(2) Replace all clauses in IDB as follows:

Let C be a clause <r,R> in IDB, where r is an atom and R is a set of atoms. If prd(r)∈ S, replace this clause by a clause, C'= <r, {r*}∪R>, where r* is an atom and arguments of r* are the same as those of r in the head. If prd(r)∉ S, C' is the same as C.

The transformation consisting of above two steps is called *clause replacement by restrictor*. 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*.

The step 1 is an equivalent transformation by proposition 2.3. The step 2 is a clause replacement and is sound by proposition 2.2. Thus, it is clear that clause replacement by restrictor is sound. If it is complete, then it is an HCT and is called *HCT by restrictor (HCT/R).* ■

The restrictor can be considered as a kind of filters discussed in the literature [4]. The relationship between models before and after clause replacement by restrictor is shown in Figure 1. The original answer is G, and the answer after the transformation is $G \cap M_2$. If the transformation is complete, then $M_2 \supset G$. Note that there is no guarantee that M_2 is smaller than the original model, M_{DDB}, unlike other transformations based on clause replacement, because of step 1.



M_1 : Model after step 1

M_2 : Model after step 2

Figure 1. The Relationship of Models

The following proposition is useful to consider form of restrictor clauses.

Proposition 3.1 Let f be an HCT/R. Suppose DDB' = f(DDB) and <r*,P> is a restrictor clause. Let P' be a subset of P and DDB" be obtained by replacing <r*,P> by <r*,P'>. Then DDB $\approx_g$ DDB".

Proof Let G, G' and G" be the answer to g of DDB, DDB' and DDB" respectively. Because the transformation from DDB to DDB" is still a clause replacement, G⊃G". Because {<r*,P'>}⇒ <r*,P>, G"⊃G'. Therefore, G = G", because G = G'. ■

Let us discuss conditions for restrictor clauses to make this transformation equivalent by considering completeness of the transformation. Let DDB' be the transformed result. First, consider a restricted clause <g',{g*}∪R> where g=g'θ for goal g. It is necessary that the same results as DDB are obtained in DDB' by applying sequences of resolutions to g. The resolvent of g and the original clause <g',R> is Rθ. The resolvent of g and the restricted clause is (g*,R)θ = (g*θ,Rθ). If DDB⇒g*θ, we obtain Rθ from (g*θ,Rθ). Therefore, if DDB⇒g*θ, DDB' may be equivalent to DDB with respect to g. This consideration gives the following definition.

Definition 3.3 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 other clauses in DDB. Let <p,{r}∪P> and <r,R> be clauses in DDB. To simplify discussions, assume p∉ S, r∈ S and arguments of r's in both clauses are same. The corresponding restricted clauses in DDB' are <p,{r}∪P> and <r,{r*}∪R>. Consider sequences of resolutions in DDB and DDB'. In DDB, p changes as follows:

$p \rightarrow r$,P→ R,P where P and R are formulas in this notation, and italic indicates an atom to be resolved. In DDB', we have:

$p \rightarrow r$,P→ r*,R,P.

If <r*,P> is a logical consequence of DDB', then r*,R,P results in P,R,P and this formula is logically equivalent to R,P. A simple way to make <r*,P> a logical

consequence of DDB' is to include a clause <r*,P'> in DDB' where P' is a subset of P, because <r*,P'> subsumes <r*,P>. Thus, we obtain the following definition.

Definition 3.4 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. ■

Let us discuss whether restrictors obtained by initial clauses and CERCs give an equivalent transformation or not. If a CERC given by P'=P results in an equivalent transformation, any CERC can be used for equivalent transformation by proposition 3.1. A CERC given by P'=P is called a primitive CERC.

Example 3.1 ancestor

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

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

EDB includes clauses for parent.

Let the restricted predicate set be {ancestor}. The restricted clauses are:

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

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

A primitive CERC is obtained from the second restricted clause.

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

The initial clause varies depending on the goal. For instance, the initial clause obtained for the goal ancestor(taro,Y) is ancestor*(taro,Y). It is easy to see that the resulted set of clauses is equivalent with respect to any given goal for ancestor.

Next, let us consider more complex case.

Example 3.2 Mutually recursive query

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

p(X,Y):-a(X,Z1),p(Z1,Z2),q(Z2,Y)

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

q(X,Y):-p(X,Z1),c(Z1,Z2),q(Z2,Y)

Clauses for a, b and c are given in EDB.

Let restricted predicate set be {p,q}. The restricted clauses obtained are:

p(X,Y):-p*(X,Y),a(X,Y)

p(X,Y):-p*(X,Y),a(X,Z1),p(Z1,Z2),q(Z2,Y)

q(X,Y):-q*(X,Y),b(X,Y)

q(X,Y):-q*(X,Y),p(X,Z1),c(Z1,Z2),q(Z2,Y).

Primitive CERCs are obtained from the second and fourth restricted clauses. The second gives two primitive CERCs:

p*(Z1,Z2):-p*(X,Y),a(X,Z1),q(Z2,Y)

q*(Z2,Y):-p*(X,Y),a(X,Z1),p(Z1,Z2).

The fourth gives:

p*(X,Z1):-q*(X,Y),c(Z1,Z2),q(Z2,Y).

q*(Z2,Y):-q*(X,Y),p(X,Z1),c(Z1,Z2).

The resulted set of clauses combined with an initial clause is not equivalent to original set. For instance, the subset of model corresponding to p, q, and q* is always empty for the initial clause p*(c1,c2) if there is not a(c1,c2) in EDB. Thus, CERCs do not unconditionally give equivalent transformation. Let us consider the reason of this problem by revisiting the discussion to give the definition of CERCs.

Let <g,{p,r}>, <p,P> and <r,R> be clauses in DDB, and assume prd(p) and prd(r) are elements of restricted predicate set but prd(g) is not. Restricted clauses in DDB' corresponding to these clauses are:

<g,{p,r}>, <p,p*∪P> and <r,r*∪R>.

Primitive CERCs obtained from the first restricted clause are:

<p*,r> and <r*,p>.

If neither p nor q does not appear in the body of restricted clauses other than <g,{p,r}>, then there are no other primitive CERCs. We obtain following sequence of resolvents in DDB.

$g \rightarrow p,r \rightarrow P,r \rightarrow P,R$

The sequence of resolvents in DDB' is:

$g \rightarrow p,r \rightarrow p^*,P,r \rightarrow p^*,P,r^*,R$

The resolution may be continued using CERCs as follows.

$p^*,P,r^*,R \rightarrow r,P,r^*,R \rightarrow r^*,R,P,r^*,R \rightarrow p,R,P,r^*,R \rightarrow p^*,P,R,P,r^*,R.$

It is easy to see that it is not possible to get a formula equivalent to (P,R) even if the sequence is further continued. The reason of this problem is considered as follows. The head predicate of a clause is considered to be dependent to its predicates. We may consider a dependency graph based on this dependency. In the above example, there is a cyclic path,

$p \rightarrow p^* \rightarrow r \rightarrow r^* \rightarrow p,$

in the dependency graph of DDB'. Therefore, we are not able to eliminate predicates in this cycle. The problem may be solved if there are no such cycles. In the above example, the cycle may be eliminated by cutting dependency of either $p^* \rightarrow r$ or $r^* \rightarrow p$ (dependency $p \rightarrow p^*$ and $r \rightarrow r^*$ should not be cut by definition). The sequence of resolutions using CERCs without the first dependency, <p*,{}> and <r*,p>, is as follows.

$p^*,P,r^*,R \rightarrow P,r^*,R \rightarrow P,p,R \rightarrow P,p^*,P,R \rightarrow P,P,R.$

The last formula is logically equivalent to (P,R).

The above discussion gives the following definition.

Definition 3.5 Let n be the number of restricted atoms in the body of a restricted clause, C. Let N be a set of clauses that is obtained by selecting a CERC for each restricted atom in the body of C. N has n elements. A relation "$\rightarrow$" is defined for each element C* of N as follows. If p* = prd(C*) and a restricted predicate, r, is in the body of C*, then $p \rightarrow r$. Here, a predicate that appears more than once in the body of C is treated as if each instance has a different predicate by attaching identification numbers to predicates. If there are m restricted atoms in the body of C*, then there are m such relation instances corresponding to C*. Consider a directed graph that is obtained by

collecting all these relation instances of elements of N. N is *acyclic* iff this graph does not have a cycle. Let R(N) be a union of all Ns corresponding to restricted clauses. R(N) is *acyclic* iff every N is acyclic. ■

Proposition 3.2 Let DDB be a deductive database. Let DDB' be a set of clauses that consists of restricted clauses defined by definition 3.2, an initial clause and elements of acyclic R(N) defined in definitions 3.3 and 3.5. Then, DDB $\approx_g$ DDB'. ■

The proof is given in appendix 1. Note that proposition 3.2 gives a sufficient but not necessary condition for the equivalent transformation. In fact, there are examples of equivalent transformations obtained by cyclic set of CERCs.

4. Horn Clause Transformation by Partial Restrictor

A modification of the transformation discussed in chapter 3 is described in this chapter. There are two reasons for this modification.

(1) In restrictor clauses, variables of head may not appear in body. For instance, initial clause usually has such variables unless all arguments of the goal are bound. This causes two problems. First, the resulted set of clauses may not be bottom-up evaluable [4] even if the original set is. Therefore, implementation may be difficult. Second, such variables represent whole Herbrand space in the model, and result in large models.

(2) Only one restricted clause is obtained from one original clause by definition 3.2. This may be too restrictive.

The definitions in the previous chapter are modified using the concept of partial restrictor that has smaller arity than the restrictor.

Definition 4.1 Let terms in definition 3.2 be changed as follows.

(1) r* is replaced by $r^*\_^{b/f}$, where b/f is a sequence of b and f and is used as an adornment of r*. The length of the adornment is equal to the arity of r. The arity of $r^*\_^{b/f}$ is equivalent to the number of "b"s of its adornment. This predicate (or atom) is called a *partial restrictor*.

(2) A *restricted clause* becomes <r,{$r^{*\_b/f}$}∪R> instead of <r,{r*}∪R>. Here, the arguments of $r^{*\_b/f}$ in the body are same as those arguments of the head corresponding to the position of "b"s. More than one restricted clause are generated from one (original) clause, if more than one predicate $r^{*\_b/f}$ with different adornments are used for the transformation.

The resulted transformation is called *clause replacement by partial restrictor*. If it is an HCT, it is also called *HCT/R*. ■

Proposition 3.1 is also correct for clause replacement by partial restrictor. Definitions for restrictor clauses are also obtained by modifying terms of definitions in chapter 3.

Definition 4.2 The *initial clause* is obtained by changing definition 3.3 as follows. g* is replaced by $g^{*\_b/f}$. Here, the arity is equal to the number of "b"s in adornment, and arguments are the same as arguments of g corresponding to the position of "b"s. ■

Definition 4.3 The *CERC* for a clause replacement by partial restrictor is defined by replacing r* by $r^{*\_b/f}$ in definition 3.4. There may be more than one predicate, $r^{*\_b/f}$, corresponding to a predicate, r*, but only one of these alternatives is chosen. Different predicates, $r^{*\_b/f}$, corresponding to the same restricted predicate, r, can be used as the head predicate of CERCs if they are generated from different instances of the same predicate in the body of a restricted clause or from different restricted clauses. ■

Definition 4.4 A *set of CERCs* for clause replacement by partial restrictor is same as definition 3.5 except that definition 3.4 is replaced by definition 4.3. The *acyclicity* is also defined by replacing r* by $r^{*\_b/f}$. ■

Proposition 4.1 Let DDB be a deductive database. Let DDB' be a set of clauses that consists of all restricted clauses defined by definition 4.1, an initial clause and elements of acyclic R(N) defined in definitions 4.2 and 4.4. Then, DDB' $\approx_g$ DDB ■

The proof of this proposition is similar to the one for proposition 3.2. The choice of adornments is arbitrary except that every element of prd({initial clause}∪R(N)) must appear in the bodies of restricted clauses by definition 4.1.

5. Optimal HCT/R

Proposition 4.1 gives a sufficient condition for an HCT by (partial) restrictor. Let us consider the way to obtain an optimal transformation that satisfies this condition. Because the efficiency of processing depends on execution algorithms, the optimal transformation may vary depending on the execution algorithms. However, we need criteria relatively independent of the execution algorithms. We propose the following two criteria for the optimality of transformation.

(1) The size of model (e.g., the cardinality of model).

(2) The processing cost to obtain smaller models is not large.

The first is the principal criterion for query transformations, because the model is computed by bottom-up methods. The second criterion is used as a supplement. For instance, we may choose to transform even clauses in EDB. This decision usually gives smaller models, but processing cost for transforming all clauses in EDB is very large. Moreover, nothing practical is obtained by this transformation, because EDB itself is considered as part of model and it is not necessary to compute this part of model. Therefore, clauses in EDB should not be transformed by the second criterion, and the definition of clause replacement by (partial) restrictor is determined not to change the EDB. Some predicates in prd(IDB) may also be omitted from restricted predicate set, because it may reduce the size of models. For instance, if the goal does not have any bound variables, the model becomes smaller by deciding not to include prd(g) in restricted predicate set. The effect of restrictor is obtained only by constants in IDB in this case. The details of how to determine the restricted predicate set are not discussed in this paper, and the set is assumed to be same as prd(IDB).

Let us consider how to minimize models with given restricted predicate set. The DDB is considered as a mapping T_p from a set of all Herbrand interpretations of DDB to itself [7]. It is easy to see that the result of the mapping is smaller if there are more atoms in the bodies of clauses. Therefore, we can obtain smaller models by having atoms as many as possible in CERCs. However, it is sometimes time consuming to evaluate some predicates in the body of a CERC. Consider the first primitive CERC in example 3.2:

p*(Z1,Z2):-p*(X,Y),a(X,Z1),q(Z2,Y).

Suppose the initial clause is p*(c,Y). Variable X in the CERC is determined by the initial clause, and other variables may be determined by evaluating a and q in bottom-up computation. Let us assume that the sets of clauses corresponding to a and q are large (we consider that q has clauses in EDB here). The processing cost to determine values of variables other than X may or may not be large depending on the form of the body. For instance, determining values of Z1 by evaluating atom a(X,Z1) is not time consuming if the range of X is restricted. Because Z1 is the first argument of the head, we can consider the first argument of p* is always restricted. On the other hand, determining values of Z2 is time consuming because no arguments of q(Z2,Y) are restricted. Therefore, we should choose to retain a(X,Z1) and to eliminate q(Z2,Y) to obtain:

p*(Z1,Z2):-p*(X,Y),a(X,Z1).

Because the first argument of p* is restricted but the second is not, we can make model smaller by eliminating the second argument to obtain:

$p^{*\ bf}(Z1)$:-$p^{*}\_^{bf}(X)$,a(X,Z1).

The above discussion results in the following algorithm. Although there is no guarantee that this algorithm gives the smallest model for any DDB, it gives reasonably small model for any DDB. Because it is difficult to consider an algorithm that always gives the smallest model, the semi-optimal algorithm below is considered as a practical optimal algorithm. Some definitions are necessary before defining the algorithm.

Definition 5.1 Let p be a restricted atom and C be a restricted clause, <r,{p}∪B>. Let C* be a primitive CERC, <p*, B> obtained from C. A *binding propagation symbol (BPS)* of C* is defined recursively as follows.

(a) A variable that appears in arguments of the partial restrictor in B is a BPS.

(b) A constant is a BPS.

(c) If there exists an argument of an atom in B that has a BPS (and if the argument is a function, it does not have variables other than BPSs), then all variables of the atom are BPSs.

(d) A symbol is a BPS iff it satisfies one of the above conditions. ■

Next, a procedure that generates an acyclic set of CERCs based on the concept of BPSs is defined.

Definition 5.2 A CERC generation procedure *genCERC(C,S)*;

C is a restricted clause, <r, B> and S is a given restricted predicate set.

```
genCERC(C,S);
  R := ∅;
  for every atom p in B such that prd(p) ∈ S do;
    B' := B - {p};
    C* := <p*,B'>; /* p* has the same arguments as p. */
    decide BPS in C* according to definition 5.1;
    B" := B' - {atoms that contain non-BPSs};
    decide atom p*_b/f according to positions of BPSs in p* in C*;
    R := R ∪{<p*_b/f , B">};
  end;
  if R is not acyclic then R := makeAcyclic(R);
  return R;
end.
```
■

The procedure *makeAcyclic(R)* makes a set of CERCs acyclic by removing some predicates from the bodies of the CERCs. Finally, the definition of the optimal algorithm, procedure BPA (binding passing algorithm), is given.

Definition 5.3 A binding passing procedure *BPA(g,IDB)*;

This procedure regards the whole prd(IDB) as a restricted predicate set.

BPA(g,IDB);

Trestrictor := $\{g^*\_^{b/f}\}$; /* $g^*\_^{b/f}$ *is an initial clause obtained by removing arguments that include variables from atom g.* */

Trestricted := ∅;

do for every $p^*\_^{b/f} \in$ *prd(Trestrictor);*

R := {*restricted clauses having* $p^*\_^{b/f}$ *in their body generated according to definition 4.1*};

Trestricted := *Trestricted* ∪ *R;*

do for every (new) $C \in$ *Trestricted do;*

Trestrictor := *Trestrictor* ∪ *genCERC(C,prd(IDB));*

end;

end;

return Trestricted ∪ *Trestrictor;*

end. ■

The following proposition is obvious from proposition 4.1 and the definitions.

Proposition 5.1 h(IDB) = BPA(g,IDB) is an HCT. ■

BPA is an optimal HCT/R in the sense that it preserves binding by constants as far as possible and results in reasonably small models for most DDB. The acyclic set of CERCs is uniquely determined by genCERC for many queries. However, if there is more than one alternative, i.e., if *makeAcyclic* is invoked, the concept of the binding passing alone cannot decide which alternative is the best, and information on the EDB is necessary to select one alternative. Examples of the information to be used by makeAcyclic are the size of relations and the position of index on relations.

The following example illustrate how BPA works.

Example 5.1 Mutually recursive query

goal: p(c,X)

IDB: same as example 3.2. ■

First the initial clause is determined as $p^*\_^{bf}(c)$, and the restricted predicate set is {p,q}.

The first set of restricted clauses having partial restrictor $p^*\_^{bf}$ are:

$p(X,Y)$:-$p^*\_^{bf}(X)$,a(X,Y)

$p(X,Y)$:-$p^*\_^{bf}(X)$,a(X,Z1),p(Z1,Z2),q(Z2,Y)

Following two CERCs are obtained by genCERC from the second restricted clause.

$p^*\_^{bf}(Z1)$:-$p^*\_^{bf}(X)$,a(X,Z1)

$q^*\_^{bf}(Z2)$:-$p^*\_^{bf}(X)$,a(X,Z1),p(Z1,Z2).

A partial restrictor $q^*\_^{bf}$ is generated. It gives two restricted clauses:

$q(X,Y)$:-$q^*\_^{bf}(X)$,b(X,Y)

$q(X,Y)$:-$q^*\_^{bf}(X)$,p(X,Z1),c(Z1,Z2),q(Z2,Y).

From these restricted clauses genCERC generates restrictor clauses:

$p^*\_^{bf}(X)$:-$q^*\_^{bf}(X)$

$q^*\_^{bf}(Z2)$:-$q^*\_^{bf}(X)$,p(X,Z1),c(Z1,Z2).

BPA terminates because no new partial restrictors are obtained. Note that acyclic CERCs are obtained without using makeAcyclic in genCERC. However, if the goal is p(C1,C2) instead of p(c,X), makeAcyclic is necessary to determine acyclic CERCs, because the concept of BPS results in cyclic CERCs. Another example is shown in appendix 2 to illustrate how more than one type of adornment are obtained.

The acyclic set of CERCs is decided by the concept of BPSs in genCERC. We can consider slightly different algorithms to determine CERCs. Some alternatives are as follows.

(1) Atoms that do not contribute to decide variables in head may be eliminated. For instance, the atom c(X,Z) in $a^*\_^{bf}(Y)$:-$a^*\_^{bf}(X)$,b(X,Y),c(X,Z) may be eliminated although it contains BPSs.

(2) We may choose adornment with fewer b's than that decided by BPSs in order to reduce the number of partial restrictor predicates. For instance, if $a^*\_^{bf}$ is already used, $a^*\_^{bf}$ is selected as head predicate of a CERC instead of $a^*\_^{bb}$ that is obtained by BPSs.

Comparison of these alternatives is considered to be a subject of the future work.

Although BPA gives reasonably small models for most queries, there are cases where it fails to obtain smaller models. This anomaly is introduced by clause insertion of step 1 in definitions 3.2. The subset of model corresponding to original predicates is guaranteed to be smaller than or equal to the original model, but the subset corresponding to restrictors may be large for some queries (see Figure 1). This problem does not occur for Datalog, i.e., if functions are not used in DDB. An example for the problem and the suggested improvement is discussed in appendix 3.

6. Relationship with Magic Sets

HCT/R works correctly for any type of query, including mutually recursive queries. Other methods that give similar result to HCT/R have been proposed. Examples are *magic sets (MG)* [3] [9] [10] and *generalized magic sets (GMG)* [5]. Magic predicates and modified rules in these methods correspond to (partial) restrictor predicates and restricted clauses respectively. It is easy to see that these methods satisfy the condition for HCT/R, i.e., the condition of proposition 4.1, with little or no modification, although they are formulated based on different concepts. MG gives the same (sometimes slightly different) result as BPA, although it can be used only for special types of queries such as linear or nested queries. GMG can be used for a broader class of queries. Because GMG and HCT/R are formulated based on the different concepts, the details and the intuitive meanings are different. However, the transformed results are

similar to each other. The principal differences between GMG and HCT/R are as follows.

(a) GMG is formulated based on the concept of *sip (sideways information passing)*, which is an abstraction of how binding information is passed by a top-down evaluation. The transformation of GMG is performed as follows. First, clauses are adorned based on the concept of *sip* . Next, clauses having magic predicates in head are determined based on the adornments and *sip* . Finally, the modified rules (restricted clauses) are obtained. Thus, the sequences of transformation are opposite in GMG and HCT/R.

(b) HCT/R does not have a concept directly corresponding to *sip* . However, the concept of acyclicity of CERCs corresponds to the acyclicity of *sip* , because both correspond to partial order of atoms in the body of a clause. The formulation of acyclicity of CERCs is simpler because it concerns only the restricted atoms while sip concerns all atoms. However, they are essentially equivalent. The procedure genCERC also corresponds to deciding an optimal *sip* for GMG, although how to decide the optimal *sip* is not discussed in [5]. genCERC is also essentially an extension of the algorithm for MG based on the binding graph given in [10].

(c) Syntactically, adornment of both restrictor and restricted predicates is done by GMG, but only restrictor predicates are adorned by HCT/R. If adornments of restricted predicates are eliminated, the result of GMG satisfies the condition of HCT/R, i.e., the condition for proposition 4.1. Adornment of the restricted predicate is an essential step in GMG, but is not essential for the equivalent transformation, although it can be done in HCT/R by slightly changing the definitions. Adornment of restrictor predicates is also not an essential condition for the equivalency of HCT/R, but it is introduced in order to improve the transformation.

(d) The formulation of GMG is based on the behaviors of top-down and bottom-up processing, while the formulation of HCT/R is independent of execution algorithms. As a result, the formulation is simpler and the separation of the transformation from execution algorithms is more complete in HCT/R. Note that it is claimed that *sips* and

control (of execution) are independent components of a query evaluation strategy in [5]. For instance, GMG imposes well formedness condition on clauses in DDB, but HCT/R does not have such conditions. Thus, the condition for HCT/R in proposition 4.1 is the weakest one so far known.

(e) The framework of HCT/R is generalized in order to properly handle constants in DDB by introducing a restricted predicate set and by treating constants as binding passing symbols. The adornment in GMG contains only f's for an atom with no incoming arc (in *sip* graph).

(f) A simple optimal algorithm, BPA, is given for HCT/R, and it is pointed out that the concept of binding passing alone is not sufficient to obtain an optimal transformation.

Thus, HCT/R is the most general algorithm of methods that introduce new predicates to be used as a kind of filters. It is also a framework that gives logical meanings to MG and GMG that have been difficult to recognize. However, the concept of *sip* has also its advantage because it can be used as a basis of a related method, *generalized counting*, that reduces the amount of redundant computation as well as the space of computation. The performance obtained by MG is known to be good compared to other methods [4]. The performance obtained by HCT/R is better than MGs, because it gives same performance as MGs for simple queries and better performance for complex queries.

7. Conclusions

This paper proposed a method, Horn clause transformation by restrictor (HCT/R), that transforms queries by introducing new predicates called restrictors in order to improve the performance of query processing. Although it is formulated based on different concepts, it gives similar transformed set of clauses to magic sets. Major contributions of this paper are twofold.

(1) HCT/R gives a logical foundation to methods that introduce new predicates such as magic sets, because it is based on the declarative semantics rather than procedural semantics. It is also more general than magic sets and generalized magic sets.

(2) An optimal algorithm, BPA, was proposed by considering criteria for optimal transformation. Although similar algorithms have been proposed for magic sets, it is the first optimal algorithm that is effective for broad class of queries including mutual recursion. Moreover, it was pointed out that the concept of binding passing alone is not sufficient to optimize some queries.

Acknowledgement

We would like to thank our colleagues of the KBMS PHI project at ICOT and Oki Electric Industry for their valuable comments and discussions. The example in appendix 3 is shown to authors by H. Seki of ICOT.

References

[1] Abiru, Y., et.al., Constant Propagation in Deductive Databases, Proc. 36th Annual Convention IPS Japan, Mar. 1988 (in Japanese).

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

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

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

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

[6] Ceri, S., et.al., Translation and optimization of Logic Queries: An Algebraic approach, Proc. 12th VLDB, Aug. 1986.

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

[8] 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)

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

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

Appendix 1: Proof of proposition 3.2

Let DDB and DDB' be the original set of clauses and its transformed result respectively. Because the clause replacement by restrictor is sound, the completeness of the transformation is shown below.

It is shown that there exists a node in the SLD tree of DDB' corresponding to every node of SLD tree of DDB. If such node exists, there exists a success node in DDB' tree corresponding to every success node of DDB tree. Hence, the transformation is complete.

The computation rule in SLD refutation , i.e., the order of atoms to which resolution is applied, is determined as follows.

[Computation rule for DDB]

(1) Atoms in body of a clause is reordered from left to right according to a full order that is consistent with the partial order of corresponding acyclic CERCs in DDB'.

(2) The next subject of the resolution of each goal clause is the left most atom.

[Computation rule for DDB']

(1)Body of a restricted clause is reordered same way as its original clause, except that the restrictor is put at left most side. Body of a restrictor clause is reordered same way, although it has fewer atoms.

(2) Same as (2) for DDB.

Note that SLD refutation is sound and complete independent of the computation rules. Under the above preparation, the completeness is proved. Substitution of variables are omitted to simplify the discussion, because the same substitution are used in both DDB and DDB' trees.

[1] Resolution of the goal g.

(a) In DDB, :- A is obtained from :-g and clause g':-A where prd(g) = prd(g').

(b) In DDB', :-A is obtained from :-g and g':-A if prd(g) is not an element of restricted predicate set. :-g*,A is obtained from g and g':-g*,A, otherwise. :-A is also obtained from :-g*,A and the initial clause.

[2] Assume that there exists a node N* in DDB' tree that corresponds to a node N in DDB tree.

(a) Let a goal clause, :-r,A, correspond to N in DDB tree where A is a conjunct of atoms. :-A',A is obtained from it and r:-A' by resolving r.

(b) In DDB', N* also corresponds to :-r,A. If r is not restricted, then the result is same as (a) above. Otherwise, :-r*,A',A is obtained from :-r,A and r:-r*,A' by resolving r. A sequence of resolutions that refutes r* as a sub-goal always succeeds, and a goal clause :-A',A is obtained. The reason why the refutation of r* succeeds is as follows.

A clause that has r in body, p:-p*,L,r,R, is used for resolution in the path from the root to node N*. Let the result of resolution be :-p*,L,r,R,B. To reach N* , the refutation of p*,L must succeed. There is a restrictor clause that has r* in its head and the body is p*,L' where L' is a sub-expression of L, because of the computation rule. Therefore, the refutation of r* whose first resolution is performed with this clause always succeeds.

The existence of a node in DDB' tree corresponding to every node in DDB tree is proved by mathematical induction on the path length from the root to a node by [1] and [2] above. Therefore, the transformation is complete. ■

Appendix 2: An example of BPA.

The following is an example where two partial restrictors with different adornments for a restricted predicate are obtained.

Example: (non-symmetric same generation) [4]

goal: sg(c,X)

IDB: sg(X,X)

sg(X,Y):-parent(X,X1),parent(Y,Y1),sg(Y1,X1)

Clauses for parent are in EDB. ■

The initial clause is $sg^*\_^{bf}(c)$. The restricted clauses having $sg^*\_^{bf}$ are:

$sg(X,X)$:-$sg^*\_^{bf}(X)$

$sg(X,Y)$:-$sg^*\_^{bf}(X)$,parent(X,X1),parent(Y,Y1),sg(Y1,X1)

A restrictor clause is obtained by genCERC from the second restricted clause.

$sg^*\_^{fb}(X1)$:-$sg^*\_^{bf}(X)$,parent(X,X1)

Because a new partial restrictor is obtained, new restricted clauses are obtained.

$sg(X,X)$:-$sg^*\_^{fb}(X)$

$sg(X,Y)$:-$sg^*\_^{fb}(Y)$,parent(X,X1),parent(Y,Y1),sg(Y1,X1)

genCERC generates a new restrictor clause.

$sg^*\_^{bf}(Y1)$:-$sg^*\_^{fb}(Y)$,parent(Y,Y1)

The partial restrictor obtained here is not new and BPA terminates.

Appendix 3 An example where BPA fails to obtain smaller model

goal: s(f(f(c)))

IDB: s(X):-s(f(X))

EDB: ground unit clauses for s

It is clear that this DDB has a finite model, and a bottom-up evaluation can effectively computes answer. Let us apply BPA to this query.

The initial clause is $s^*\_^{b}(f(f(c)))$. A restrited clause having $s^*\_^{b}$ is:

s(X):-$s^*\_^{b}(X)$,s(f(X))

A restrictor clause is obtained from this clause.

$s^*\_^b(f(X)):\text{-}s^*\_^b(X)$

Because the resulted set of clauses has an infinite model, the bottom-up evaluation does not terminate for the transformed query. Note that a top-down evaluation also fails to terminate for the original query. This kind of anomaly never occurs if functions are not used in DDB. Moreover, there are cases opposite to this example, i.e., the original model is infinite but the transformed model is finite. We may change the algorithm as follows to cope with this problem. If the argument in head is more complex than the argument in body having same variable, then it is adorned by f instead of b. We can obtain a finite model by this modification for this example.