ICOT Technical Report: TR-270

TR-270

Restricted Least Fixed Points and Recursine Query Processing Strategies

by
N. Miyazaki (Oki) and H. Itoh

June. 1987

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

Restricted Least Fixed Points and Recursive Query Processing Strategies

Nobuyoshi Miyazaki* and Hidenori Itoh**

*Oki Electric Industry Co. Ltd.

**Institute for New Generation Computer Technology

Abstract

It has been difficult to understand logical relationships between various strategies for recursive query processing. The difficulty results from the fact that we use both database and logic programming concepts together to realize efficient processing strategies, and from the lack of common principles for these strategies. This paper tries to give a common ground to strategies based on the concept of restricted least fixed points. This concept is an extension of Aho and Ullman's strategy and can be applied to a broad class of queries. Because restricted least fixed points are much smaller than original least fixed points, they can be computed efficiently. The concept also gives a criterion to compare strategies based on what they compute rather than how they compute the result. First, the principle of restricted least fixed points and restrictors is discussed. Next, classification and comparison of strategies are discussed based on what they compute, and strategies which appear very different are shown as variations of algorithms to compute some kinds of restricted least fixed points.

1 Introduction

Numerous strategies for recursive query processing in deductive databases have been proposed over the past few years. As a result, it is known that most queries can be processed, at least in principle, and several strategies of good performance have been found. However, it is difficult to decide which strategy is appropriate to a given query, because these strategies look very different from each other. The main reason for this confusion is that this field is on the boundary of database technology and logic programming technology, and many techniques have been applied to the same problem.

An attempt to clarify this situation has been made by Bancilhon and Ramakrishnan in their extensive survey [Bancilhon86a]. They used several characteristics to classify strategies. A common measure to compare the performance of strategies was also used. However, it is still difficult to see why the performance of an algorithm is roughly the same as another algorithm. For instance, the magic set strategy [Bancilhon86b] is bottom up, compile and iterative. In contrast, the recursive query/subquery strategy (QSQR) [Vieille86] is top down, interpretive and recursive. It is interesting to see that these two strategies show the same performance except for the non-linear case, although they are exactly opposite in the classification [Bancilhon86a]. Their explanation for this similarity is that the algorithms show the same behavior as measured by:

(1) the amount of duplication of work,

(2) the size of the set of relevant facts, and

(3) the arity of intermediate relations.

However, their explanation does not show why these strategies behave this way.

Beeri and Ramakrishnan proposed a principle of sideways information passing (sip) to discuss the second criterion above, and pointed out that all published strategies in the literature could be described by sip [Beeri87]. They also claim that the collection of sips and the control strategy are distinct, possibly independent components of a query evaluation strategy.

This paper discusses the problem from a different angle. The authors' previous paper proposed the concept of restricted least fixed points (rlfp) and their computation algorithm [Miyazaki87a]. Although this concept may be seen as another way to realize sips by side rules, the underlying principle is different. The principal idea is to restrict the size of derived relations rather than restricting the computation. Because restricted least fixed points are much smaller than original least fixed points, they can be computed efficiently. This paper discusses how this principle can be used to clarify the relationships of various strategies. Thus, the notion of rlfp and sip complement each other as foundations of recursive query processing strategies.

Chapter 2 summarizes the concept of restricted least fixed points and restrictors proposed in the authors' previous paper. Chapter 3 discusses the propagation of restriction conditions and decomposition of restrictors. Chapter 4 proposes a criterion to classify strategies and classifies strategies. Several

strategies are shown as variations of ways to compute restricted least fixed points.

2 Restricted Least Fixed Points

A query is expressed by a set of Horn clauses as shown in equation 1-a. It is assumed that there are no functions or structures in predicates to simplify the discussion. The query has its equivalent relational expressions as shown in equation 1-b [Ceri86] [Miyazaki86].

```
:-r1(a set of arguments).
rk:-hk(r1,r2,...,rn). (There may be
        several clauses for the same rk.)   (Eq.1-a)
```

$$\text{answer} = \sigma_F(r1)$$

$$rk = fk(r1,r2,\ldots,rn) \qquad k=1,2,\ldots,n \qquad \text{(Eq.1-b)}$$

where rk corresponds to derived relations. Among rks, r1 corresponds to the result relation that corresponds to the goal. σ_F is a selection operator corresponding to the goal. Base relations stored in extensional database (EDB) are not shown in equation 1. The answer of this query can be obtained by computing least fixed points of rks and then computing $\sigma_F(r1)$. Several strategies have been proposed to compute least fixed points [Bancilhon86a]. However, computing least fixed points directly is not a good way to answer the query because usually only small subsets of rks contribute the answer and these subsets may be computed without computing entire derived relations. One way to reduce the size of derived relations to answer the query

was proposed by Aho and Ullman [Aho79]. The essential point of their strategy is as follows. Consider a query with one derived relation,

answer = $\sigma_F(r)$

$r = f(r)$.

$f(r)$ may be separated into two parts; one does not involve r, and the other involves r. In other words, $f(r) = g + h(r)$ where $+$ means union. If $h(r)$ is commutative, i.e. $\sigma_F(h(r)) = h(\sigma_F(r))$, then $f(r)$ is said to be commutative, and $\sigma_F(r) = \sigma_F(f(r)) = \sigma_F(g) + h(\sigma_F(r))$. Therefore, if $r' = \sigma_F(r)$, we get

answer = r'

$r' = \sigma_F(g) + h(r')$.

Using this equation, the answer can be computed without computing the whole derived relation, r. Although this strategy does not always work, the idea can be extended to a more general strategy. First, if $h(r)$ is commutative and $r' = \sigma_F(r)$, the following equation also holds.

answer = $\sigma_F(r')$

$r' = \sigma_F(f(r'))$

This equation does not always hold in general either, but there may exist a selection operator σ^* such that

$\sigma_F(r') = \sigma_F(r)$ = answer

$r' = \sigma^*(f(r'))$.

If such σ^* is found, the answer can be computed by computing r' instead of r. When D is the domain of r, $\sigma^*(r) = (\sigma^*(D) \cap r)$. $\sigma^*(D)$ is denoted as r^* and the following equations obtained.

answer = $\sigma_F(r')$

$r' = r^* \cap f(r')$

These equations are generalized to the following equations.

Definition of restricted least fixed points

For a given query, Q, expressed in equation 1-b, consider the following set of equations.

$$\text{answer} = \sigma_F(r1') = \sigma_F(r1)$$

$$rk' = rk^* \cap fk(r1', r2', \ldots, rn')$$

$$\text{for } k=1,2,\ldots,n \qquad \text{(Eq.2)}$$

where each rk* is called a restrictor of rk, and <r1*,...,rn*> is called a restrictor of Q. Each rk' is a restricted fixed point of rk, and <r1',...,rn'> is called a restricted fixed point of Q. If a restrictor is given, the answer can be obtained by computing the restricted least fixed points. It is clear that computation can be done by any strategies that compute least fixed points. By setting rk*s as small as possible, the time for query processing can be minimized.

Two trivial restrictors are immediately shown. One is a set of domains. In this case, equation 2 reduces to equation 1-b. The other restrictor is $\sigma_F(D)$ when f(r) is commutative. It is easy to see the following properties from equation 2.

Property 1

If pk is a restrictor of rk, then $sk \supset pk$ is also a restrictor of rk.

Property 2

For any k, rk' is a subset of $(rk^* \cap rk)$.

The concept of restricted least fixed points is useless unless non-trivial restrictors are given. It is not usually possible to decide restrictors without consulting the EDB. Therefore, restrictors are given as derived relations. As a

result, restrictors and rlfps are given as least fixed points of a new set of rules. Expressions of restrictors are given by a set of Horn clauses rather than by relational algebraic equations to simplify the discussions. Note that the distinction between rk' and rk is not made in the algorithm, because the distinction is no longer necessary once the transformation from equation 1 to equation 2 is made. However, we distinguish between rk' and rk whenever necessary.

Algorithm 1: algorithm to get rk' and rk*

I: Initial condition

rl*:-rl_init*. (arguments are the same as rl on both sides)

Define rl_init* as a unit clause whose arguments are exactly the same as those of the goal.

Variables in rl_init* are considered to represent their domains in the corresponding relational expression.

II: Clauses for rk (definition for rk' with ' omitted)

Transform rk:-hk(rl,...,rn).

to rk:-rk*,hk(rl,...,rn).

by adding rk*, where arguments of rk* are equal to those of the head predicate.

Make the above transformation for every clause in Q.

III: Clauses for rk* (definition for rk*)

(1) Select an rk

(2) Select a clause modified in II which has rk in its body.

(3) The clause is in the following form if rk is moved to the farthest right.

rj:-rj*,gj(r1,...,rn),rk.

Generate the following clause from the above.

rk*:-rj*,gj(r1,...rn).

where the arguments of rk* are same as the rightmost rk in the selected clause.

(4) Repeat (3) if there are more than one rk in the body

(5) Repeat (1) to (4) for every possible combination.

Intuitively, the definition algorithm simulates an idealized theorem prover that determines every possible value before entering a new predicate call. It is idealized because it is not possible to determine every possible value, and an actual theorem prover such as Prolog has a different control strategy. Thus, the definition of rk*s does not always give a restrictor. Let us discuss the condition for restrictors.

The head predicate is regarded as depending on predicates of the body in a rule. Consider a rule

rj:-rj*,hj(r1,...,rn).

Numbers are attached to predicates to distinguish predicates with same name in the body of a rule. When a rule defining rk* is constructed,

rk*:-rj*,gj(r1,...,rn).,

rk* depends on the predicates in the body. Because rk depends on rk*, we have a dependency graph that represents the dependency of numbered rk on other predicates. Let S be a set of clauses defining rk*s generated from the same rule in step III. The dependency graph of S is a graph obtained by superimposing dependency graphs of rules in S. S is said to be consistent iff its dependency graph does not have a cycle. Intuitively,

consistency means that S represents an executable order of predicate calls for the rule. The definition of rk*s is called consistent iff every S generated in step III is consistent.

Algorithm 1 gives a new set of Horn clauses. The least fixed points of derived relations in these clauses can be computed. For these derived relations, there is the following theorem.

Theorem

Let r1*,...,rn* and r1',...,rn' be least fixed points which satisfy rules given by the above algorithm. If the definition of rk*s is consistent, then <r1*,...,rn*> and <r1',...,rn'> satisfy equation 2. In other words, the definition of rk*s gives a restrictor if it is consistent.

The proof of this theorem is given in [Miyazaki87a]. If the definition of rk*s is not consistent, a restrictor can be obtained by making it consistent.

Corollary

The definition of rk*s becomes the definition of a restrictor if it becomes consistent by removing some predicates in the body of its defining clauses.

In algorithm 1, rk*s are introduced for all derived relations. Because non-recursive derived relations may be directly evaluated, we may choose not to introduce restrictors for them. A query is called linear iff defining clauses have at most one occurrence of recursive predicates in body. The next property immediately follows.

Property 3

If rk*s are not introduced for non-recursive predicates corresponding to derived relations, the definition of rk*s given by algorithm 1 for a linear query Q is consistent and is a restrictor of Q.

A restrictor of Q is separable iff its defining clauses do not have rk's in the body. A separable restrictor can be computed before computing restricted least fixed points. From algorithm 1, the following property is obvious.

Property 4

It is assumed that restrictors for non-recursive derived relations are not introduced. Then, a restrictor of Q given by algorithm 1 is separable if Q is linear.

If the definition of rk*s is not consistent, some predicates in the body must be removed. The restrictive power of a restrictor depends on which predicates are removed. It is usually easy to find which predicates should be removed. A few examples of the use of restrictors for non-linear and mutually recursive queries are shown in [Miyazaki87a]. The next chapter discusses a way to make a restrictor from the definition of rk*s by analyzing the natural way to propagate restriction conditions.

3 Decomposition of Restrictors

It is desirable to retain predicates that propagate restrictive conditions when a restrictor is generated from rk*s given by algorithm 1. When the values of some arguments are given, the values of the remaining arguments can be determined by

referring to the EDB. Once some arguments are determined, the values of arguments with the same variable name in other predicates are also determined. Thus, the condition can be propagated through predicates defining rk*s. Therefore, the natural way is to retain such predicates as far as possible.

In the definition of rk*s given by algorithm 1, some arguments of predicates for the initial condition are variables. Relational algebra must be extended a little to allow variables in tuples, although the extension is conceptually easy because a variable unifies every constant. Moreover, some arguments of the restrictor may remain unbound during computation of a least fixed point. The implementation would be easier if there were no such arguments. Restrictors should be decomposed to eliminate such arguments.

This chapter discusses a way to determine a restrictor and then a way to eliminate free variables.

The natural way to generate a restrictor from clauses defining rk*s is first to analyze how the restriction conditions can be propagated. Let us consider that each argument of predicates in restrictor definition clauses can be assigned a bound or free prefix. The bound prefix is passed through predicates. A restrictor can be generated by the following algorithm.

Algorithm 2: Determining restrictor definition

I. Analysis of binding relationship

Assign bound or free prefixes to the initial condition according

to its binding relationship of arguments.

Starting from the initial condition, generate new clauses as follows. This procedure is terminated when no new binding relationship of rk*s is generated.

(1) Select a clause and assign a given binding relationship to the restrictor in the body.

(2) Assign bound to variables with same name that have been assigned bound.

(3) Except for restrictors, assign bound to arguments of each predicate if one of the arguments is assigned bound. This means that relations propagate conditions. Repeat (2) and (3) until there are no arguments to be assigned bound.

(4) Assign free to all remaining variables.

II. Transformation

(1) Eliminate predicates in the body that correspond to derived relations whose arguments are all free.

(2) If the result is consistent, then terminate. If not, make it consistent and reassign the bound or free prefixes. The definition of consistency can be extended to allow the same predicates with different binding relationship to depend on each other, because they represent different branches of a proof tree.

The elimination of predicates in step II (1) gives a consistent restrictor for many queries. However, the following problems remain.

(1) The way to determine which predicates to be eliminated, if the result of step II (1) is not consistent.

(2) If several bound prefixes can be propagated through different

predicates, the performance of later computation may be improved by eliminating some more predicates that have bound prefixes in the body of restrictor definition.

(3) The restrictor definition may have predicates corresponding to base relations whose arguments are free. Their presence conceptually gives more restrictive power. However, it is not usually a good way to retain these predicates, because evaluating these predicates for restrictors is time consuming if they correspond to large relations.

Detailed discussion of these problems is beyond the scope of this paper. They should be treated in the discussion of further optimization. Similar problems in literature are determination of sip [Beeri87] and the wave front problem [Han86]. However, the third problem does not arise if every base relation is large. In such a case, all predicates with free arguments should be eliminated to improve performance, and it is assumed that such predicates are always eliminated in the following discussions.

Next, the problem of free variables in restrictors is discussed. These arguments can be eliminated by decomposing restrictors. If a restrictor has n arguments, it can be expressed as the union of 2**n components in principle. For instance, a two argument restrictor, a*(X,Y), can be decomposed as

```
a*(X,Y):-a_bb*(X,Y).
a*(X,Y):-a_bf*(X).
a*(X,Y):-a_fb*(Y).
a*(X,Y):-a_ff*.
```

where a_bb means that both arguments are bound during

computation, a_bf means that first is bound but the second is not, and so on. Note that if there is a last component then all other components are subsumed by it. The decomposition algorithm uses the bound or free prefixes attached by algorithm 2.

Algorithm 3: Decomposition of restrictor

I. Decomposition

(1) Classify restrictor definitions according to the binding relationship of the head.

(2) Rename the restrictor predicates according to the classification.

(3) Omit arguments assigned free from decomposed restrictor predicates.

(4) Express the restrictor by decomposed predicates.

II. Elimination of original restrictor predicates from other clauses

(1) Eliminate original restrictor predicates by substituting original restrictors by decomposed restrictors. This last step is just a special case of general simplification procedure called Horn clause transformation [Miyazaki86 and 87b]. In this case, the elimination is straightforward, because restrictor predicates are expressed as a disjunction of components and do not include recursive expression.

Example 1: Non-linear ancestor

Query

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

Query with ancestor* is given as follows. The goal and the initial condition is omitted.

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

The definition of ancestor* is not consistent. The initial binding relationship of this query is [b,f]. The result of step I of algorithm 2 is as follows.

```
ancestor*(b:X,f:Z):-ancestor*(b:X,f:Y),
                    ancestor(f:Z,f:Y).
ancestor*(b:Z,f:Y):-ancestor*(b:X,f:Y),
                    ancestor(b:X,b:Z).
```

Only one binding relationship [b,f] is obtained, and step 2 eliminates ancestor(f:Z,f:Y) in the first clause to make the definition consistent. The first clause becomes redundant and may be discarded.

The step I of decomposition algorithm generates one clause,

```
ancestor_bf*(Z):-ancestor_bf*(X),ancestor(X,Z).
```

After eliminating ancestor*(X,Y) using [ancestor*(X,Y) :- ancestor_bf*(X).], the following result is obtained.

```
:-ancestor(constant,X).
ancestor(X,Y):-ancestor_bf*(X),parent(X,Y).
ancestor(X,Y):-ancestor_bf*(X),ancestor(X,Z),ancestor(Z,Y).
ancestor_init*(constant,X).
ancestor_bf*(X):-ancestor_init*(X,Y).
```

```
ancestor_bf*(Z):-ancestor_bf*(X),ancestor(X,Z).
```

Example 2: Unstable same generation [Bancilhon86b]

Query

```
:-sg(constant,X).
sg(X,X).
sg(X,Y):-parent(X,X1),sg(Y1,X1),parent(Y,Y1).
```

Except for the initial condition, the definition of sg* is

```
sg*(Y1,X1):-sg*(X,Y),parent(X,X1),parent(Y,Y1).
```

Because this is consistent, it gives the restrictor. The binding relationship of the initial condition is [b,f]. During binding relationship analysis, the following clause is obtained.

```
sg*(f:Y1,b,X1):-sg*(b:X,f:Y),parent(b:X,b:X1),
                parent(f:Y,f:Y1).
```

The head of this clause shows a different binding relationship, i.e. [f,b]. Therefore, we continue and get another clause

```
sg*(b:Y1,f,X1):-sg*(f:X,b:Y),parent(f:X,f:X1),
                parent(b:Y,b:Y1).
```

Predicates with free arguments are eliminated, giving

```
sg*(f:Y1,b,X1):-sg*(b:X,f:Y),parent(b:X,b:X1).
sg*(b:Y1,f,X1):-sg*(f:X,b:Y),parent(b:Y,b:Y1).
```

The result of step I of decomposition is as follows.

```
sg_fb*(X1):-sg_bf*(X),parent(X,X1).
sg_bf*(Y1):-sg_fb*(Y),parent(Y,Y1).
```

The restrictor has now two components.

```
sg*(X,Y):-sg_bf*(X).
sg*(X,Y):-sg_fb*(Y).
```

The final results are

```
:-sg(constant,X).
sg(X,X):-sg_bf*(X).
```

```
sg(X,X):-sg_fb*(X).
sg(X,Y):-sg_bf*(X),parent(X,X1),sg(Y1,X1),parent(Y,Y1).
sg(X,Y):-sg_bf*(Y),parent(X,X1),sg(Y1,X1),parent(Y,Y1).
sg_init*(constant,X).
sg_bf*(X):-sg_init*(X,Y).
sg_fb*(X1):-sg_bf*(X),parent(X,X1).
sg_bf*(Y1):-sg_fb*(Y),parent(Y,Y1).
```

Because there are two binding relationships, the definition of sg is also split into two expressions.

4 Query Processing Strategies

This chapter discusses the relationship of query processing strategies. The set of equations (or clauses) that defines restricted least fixed points can be regarded as equations that define a new set of relations. Clearly, relations that satisfy set of transformed equations differ in size from those defined by the original set of equations as shown in property 2. These relations can be computed by any algorithm that computes the least fixed points. Because restricted least fixed points are much smaller than original least fixed points, they can be computed efficiently. The differnce can be as large as several orders of magnitude if base relations in the EDB are large. This suggests a classification criterion of strategies based on what they compute. This criterion has a close relation to performance, because the larger the result of computation, the more time consuming it is. Three main classes and one supplementary class can be identified with this criterion.

(1) Class r: Strategies that compute least fixed points.

(2) Class $O^*(r)$: Strategies that

(a) compute restricted least fixed points, or

(b) transform rules to define a new set of derived relations that include those corresponding to restricted least fixed points, or

(c) compute answer based on rules given by (b).

(3) Class $O_F(r)$: Strategies that compute the answer without explicitly computing rs or $O^*(r)$s.

(4) Class transformation: Strategies that transform queries to other forms. They do not usually affect the size of derived relations, although the derived relations themselves are sometimes eliminated or added.

The relationship of strategies is discussed based on this classification. We have not tried to survey this field, and examples are used to discuss the variations. Therefore, the following examples are not intended to be complete. The strategy described in the previous chapters is called rlfp in this chapter.

4.1 Class r

Most strategies belonging to the first class are based on the principles of the least fixed points. Some examples of strategies of this class are:

Naive evaluation (frequently called by other names)

Semi-naive evaluation [Bancilhon86a]

Delta driven [Rohmer86]

Ordered naive evaluation (component by component) [Ceri86]

These strategies compute the least fixed points in bottom up fashion. The naive evaluation is the basis of this class of strategies. Other strategies have been proposed to reduce the amount of duplicated work.

4.2 Class $O^*(r)$

There are three subclasses in class $O^*(r)$: $O^*(r)$-a, -b and -c. The first consists of strategies that compute $O^*(r)$s before answering the query. Examples of this subclass are QSQI and QSQR [Vieille86]. These strategies are based on the top down method and look completely different from bottom up strategies as summarized in [Bancilhon86a]. However, we can see that restricted least fixed points are computed as sets called Ans_Rj, and the role of the restrictor is performed by sets called Inst_M_Rj where M denotes the binding relationship. The sets, Inst_M_Rj, are used as conditions of subqueries. The order of predicate calls is determined by a selection function. Although these concepts are closely mixed with top down control strategy, it is easy to see that QSQ and rlfp compute essentialiy the same derived relations under the assumption that a proper selection function is used in QSQ. There are several strategies preceding QSQ that belong to this subclass.

The second subclass of class $O^*(r)$ consists of strategies such as magic sets [Bancilhon86b], generalized magic sets [Beeri87], Alexander [Rohmer86], and rlfp. The common points of these strategies are:

(1) They transform a set of clauses to another set of clauses where predicates corresponding to derived relations are

transformed to predicates that satisfy equation 2.

(2) A set of predicates for new derived relations is introduced. They include predicates that have the role of restrictors.

The difference of these strategies may be found in

(1) the transformation algorithm,

(2) the transformed set of rules, and

(3) the power of the restrictors.

The generalized magic set is based on the concept of sideway information passing (sip). Sip is represented by a graph and essentially corresponds to an execution order of predicate calls. Once sip is given, a generalized magic set is determined by using a special type of rules called the adorned rule set. The adorned rule set consists of rules where predicates have a suffix to distinguish binding relationship. The original derived relations and transformed relations are not distinguished in this strategy, but the generalized magic set can be seen to have the same role as the restrictor of rlfp. Although the algorithm of generalized magic sets is more complex than that of rlfp, they produce an essentially equivalent set of clauses under the assumption that a proper sip is chosen in the former strategy. The way to choose sips is not described in [Beeri87]. The formalism used in the generalized magic set is more complex because it allows functions in predicates.

The relationship of the magic set and generalized magic set is discussed in [Beeri87]. The relationship is easier to see based on the concept of restricted least fixed points, because the magic set is obtained by modifying a restrictor slightly. A

magic set is obtained by eliminating all derived (or recursive) relations in the body of clauses that define restrictors. Thus, the magic set is equivalent to rlfp for linear queries. In fact, the restrictor of example 2 is the same as the magic set discussed in [Bancilhon86b].

Another example of this subclass is the Alexander strategy [Rohmer86]. It was pointed out that Alexander is essentially a generalized supplementary magic set [Beeri87]. Supplementary magic set is a variation of the magic set discussed in section 4.4. Alexander divides the problem into three kinds of predicates: Pbs, Sols and Conts. Although the transformation algorithm is different, Pbs correspond to restrictors, and Sols correspond to restricted least fixed points. Conts are special predicates that pass information from one rule to another. If Conts are eliminated from the rule set by Horn clause transformation discussed in section 4.4, a rule set similar to rlfp is obtained. Concepts corresponding to the analysis of the binding relationship or choosing sips are not discussed in [Rohmer86].

The last example is Aho and Ullman's. Because rlfp is an extension of Aho and Ullman's strategy, this strategy is a special case of rlfp where $\mathcal{O}^* = \mathcal{O}_F$. Moreover, it is easily shown that the restrictor given by algorithm 1 is in fact reduced to $\mathcal{O}_F$ if the query is commutative. The relationship of class $\mathcal{O}^*(r)$ strategies is summarized in Figure 1. The restricting power of restrictors in these strategies is

{Aho and Ullman's} =< {magic sets} =<

{QSQ, generalized magic sets, rlfp}.

The restricting power of magic sets is equivalent to the decomposed restrictor of rlfp if the latter is separable. Thus, this equation and property 4 answer the question in chapter 1, i.e. why QSQR and magic sets show the same performance for linear queries.

Because strategies in the class $\mathcal{O}^*(r)$-b give a new rule set but not the answer, the actual computation is done by other strategies such as those in class r or class $\mathcal{O}^*(r)$-c. The computation of restricted least fixed points by class r strategies frequently has redundant operations. The strategies in class $\mathcal{O}^*(r)$-c were proposed to reduce the redundancy. The magic counting and the generalized magic counting are known as efficient strategies based on magic sets and generalized magic sets respectively [Bancilhon86b] [Beeri87].

4.3 Class $\mathcal{O}_E(r)$

The third class consists of strategies that try to compute answers directly. Examples of these strategies are:

Henschen and Naqvi's [Henschen84]

Aho and Ullman's [Aho79]

Prolog

Among these strategies, Aho and Ullman's may be regarded as a special case of rlfp combined with semi-naive evaluation. The name of the class, $\mathcal{O}_E(r)$, is overestimated in a sense, because restricted least fixed points are identified in many strategies in this class. The name "class implicit $\mathcal{O}^*(r)$" may be more suitable.

For instance, let us analyze the processing of Prolog, which may be regarded as a query processing strategy for deductive databases. Prolog answers a query by refutation. The processing can be regarded as subsequent predicate calls. Consider the following example.

Example 3: Linear ancestor

Query

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

Prolog first attempts to get parent(X,constant) using the first clause. If it fails or more answers are necessary, it tries to gets parent(X,Y),ancestor(Y,constant). This time it first gets parent(X,Y) and uses a value of Y, say y1, to get ancestor(y1,constant) and so on. If all values to be used to get all answers in subsequent calls of ancestor(X,Y) are collected, they are equivalent to the following restrictor. The definition of ancestor* given by algorithm 1 for this query is

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

Because this query is linear, the above clauses define a restrictor. If the binding relationship is analyzed, it is

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

parent(f:X,f:Z) should be deleted to improve the performance. However, Prolog evaluates a clause from left to right and the above restrictor corresponds to the set of values Prolog uses to answer the query. Moreover, restricted least fixed point is found in Prolog processing if all instances of ancestor during

execution are collected together.

Restrictors and restricted least fixed points can be identified in Henschen and Naqvi's strategy by comparing its sequence of expanded relational algebraic expressions with that of restrictors and restricted least fixed points. Although strategies in this class can be more efficient than other strategies, they can be applied to only a limited type of queries.

4.4 Class transformation

The last class consists of strategies that transform a set of rules into another equivalent set of rules. These strategies differ from class $\mathcal{C}^{+}$(r)-b, because they do not usually affect the size of derived relations. However, derived relation themselves may be eliminated or added by them. Examples of these strategies are

Semantic Query Optimization
using Integrity constraints [Chakravarthy86]

Redundancy elimination [Sagiv87]

Horn clause transformation (elimination of derived relations by partial evaluation)
[Miyazaki86 and 87b]

Substition in relational algebra (elimination of derived relations) [Ceri86]

Rewriting of common expressions (adding derived relations for common expressions)

These strategies are independent of other classes of strategies and can be used as a supplement to other strategies. The following example shows the use of Horn clause transformation as preprocessing.

Example 4: Reducible mutual recursion

```
:-q(c,X).
q(X,Y):-a(X,Y).
q(X,Y):-b(X,Z),p(Z,Y).
p(X,Y):-c(X,Y).
p(X,Y):-d(X,Z),q(Z,Y).
```

where a, b, c and d are base relations. Although some strategies such as Henschen and Naqvi's can process this query efficiently, other strategies may not be efficient or may not be applied because p and q are mutually recursive. Horn clause transformation transforms it to the following set of rules by eliminating p.

```
:-q(c,X).
q(X,Y):-a(X,Y).
q(X,Y):-b(X,Z),c(Z,Y).
q(X,Y):-b(X,Z),c(Z,Z1),q(Z1,Y).
```

Because this is a simple query with one derived relation, many strategies can process this query efficiently.

Another example of this class is rewriting of common expressions to avoid redundant operations. Although this strategy is frequently used in traditional databases, its use has not been fully investigated for deductive databases. A use of this method is the generalized supplementary magic set discussed in [Beeri87]. Rewriting common expressions may be useful in

combination with magic sets or rlfp because common expressions can be frequently identified in them as seen in algorithm 1.

The relationship of strategies is summarized in Figure 2.

5 Conclusions

The concept of the restricted least fixed point was introduced to restrict the size of derived relations instead of restricting computation. The way to realize this concept and the relationship of this concept with other strategies were discussed. It was shown that what strategies compute is a good criterion of various strategies and that restricted least fixed points can be identified in many strategies. The performance of query processing may be improved several orders of magnitude by choosing good restrictors. Thus, this concept gives a good foundation of recursive query processing strategies.

[References]

[Aho79] Aho, A.V. and Ullman, J.D., Universality of Data Retrieval Languages, Proc. of 6th ACM POPL, pp.110-120, 1979

[Bancilhon86a] Bancilhon, F. and Ramakrishnan, R., An Amateur's Introduction to Recursive Query Processing, ACM SIGMOD '86 Proceedings, pp.16-52, 1986

[Bancilhon86b] 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, pp.1-15, 1986

[Beeri87] Beeri, C. and Ramakrishnan, R., On the Power of Magic, Proc. of 6th ACM PODS, pp.269-283, 1987

[Ceri86] Ceri, S., Gottlob, G. and Lavazza, L., Translation and Optimization of Logic Queries: The Algebraic Approach, Proc.

of 12th VLDB, pp.395-402, 1986

[Chakravarthy86] Chakravarthy, U.S., Minker, J. and Grant, J., Semantic Query Optimization: Additional Constraints and Control Strategies, Proc. of 1st EDS, pp.259-269, 1986

[Han86] Han, J. and Lu, H., Some Performance Results on Recursive Query Processing in Relational Databases, Proc. of 2nd Int. Conf. on Data Engineering, pp.533-541, 1986

[Henschen84] Henschen, L. and Naqvi, S.A., On Compiling Queries in Recursive First-Order Databases, JACM, Vol. 31, No. 1, pp.47-85, 1984

[Miyazaki86] Miyazaki, N., Yokota, H. and Itoh, H., Compiling Horn Clause Queries in Deductive Databases: A Horn Clause Transformation Approach, ICOT Technical Report TR-183, 1986

[Miyazaki87a] Miyazaki, N. and Itoh, H., Restricted Least Fixed Points in Deductive Databases, ICOT Technical Reprort, TR-257, 1987 (in Japanese)

[Miyazaki87b] Miyazaki, N., Haniuda, H. and Itoh, H., Horn Clause Transformation: An Application of Partial Evaluation in Deductive Databases, ICOT Technical Report TR-256, 1987 (in Japanese)

[Rohmer86] Rohmer, J., Lescoeur, R. and Kerist, J.M., The Alexander Method - A Technique for The Processing of Recursive Axioms in Deductive Databases, New Generation Computing, Vol. 4, pp.273-285, 1986

[Sagiv87] Sagiv, Y., Optimizing Datalog Programs, Proc. of 12th ACM PODS, pp.349-362, 1987

[Vieille86] Vieille, L., Recursive Axioms in Deductive Databases: The Query/subquery Approach, Proc. of 1st EDS, pp.179-193, 1986

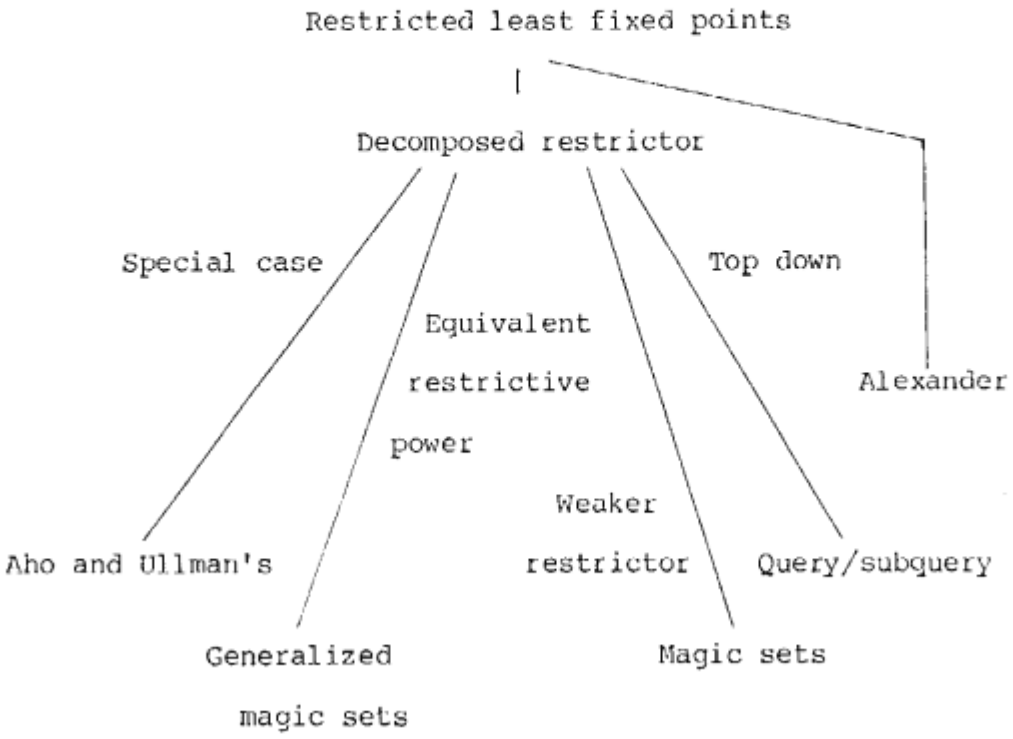

Figure 1 Relationship of class $\mathcal{O}^{\xi}(r)$ strategies

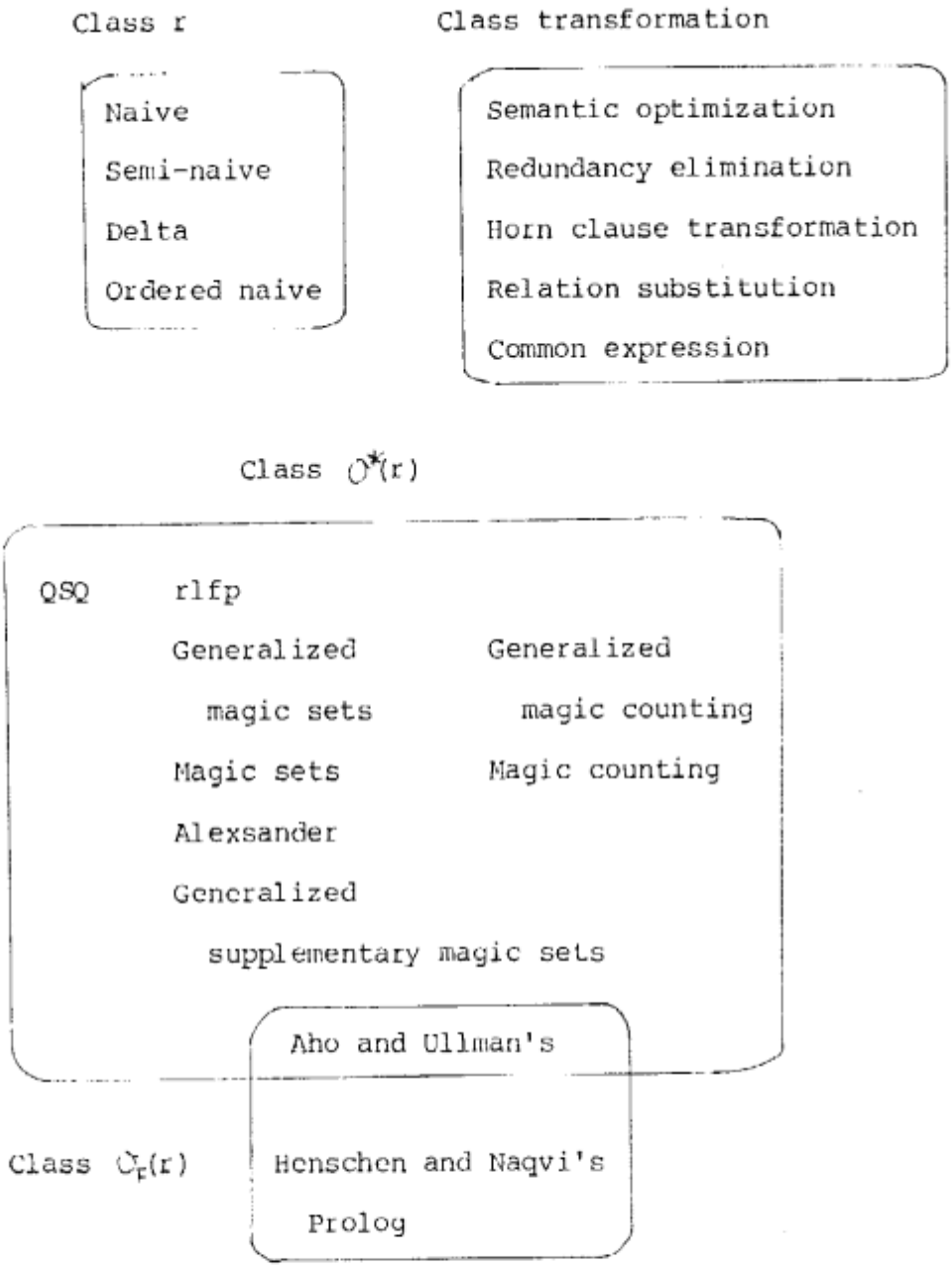

Figure 2 Relationship of strategies