Institute for Language, Logic and Information

TOWARDS FUNCTIONAL CLASSIFICATION OF RECURSIVE QUERY PROCESSING

Zhisheng Huang
Sieger van Denneheuvel
Peter van Emde Boas

ITLI Prepublication Series
for Computation and Complexity Theory CT-89-11

University of Amsterdam

TOWARDS FUNCTIONAL CLASSIFICATION OF RECURSIVE QUERY PROCESSING

Zhisheng Huang
Sieger van Denneheuvel
Peter van Emde Boas
Department of Mathematics and Computer Science
University of Amsterdam

Received December 1989

Towards Functional Classification of Recursive Query Processing

Zhisheng Huang*, Sieger van Denneheuvel, Peter van Emde Boas

Department of Computer Science, University of Amsterdam

1018TV Amsterdam, The Netherlands

ABSTRACT

In recent years, many techniques concerning recursive query processing have been proposed. In search of a unified theory and strategy of recursive query processing, there have been some attempts to classify existing techniques. In this paper, we present a new classification called the *functional classification* of techniques concerning recursive query processing. The classification differs from Bancilhon and Ramakrishnan's classification based on external features and Roelants' classification based on halting conditions. Our method distinguishes the features of almost completeness, definiteness and proper finegrainedness, which are viewed as the adequate criteria for classification. A formalism of functional classification is presented. Based on the formalism, the composability of the existing techniques concerning recursive query processing is examined.

Keywords. recursive query processing, deductive databases, functional classification

* *On leave from Dept. of Computer Science, Zhenjiang Shipbuilding University, China.*

1. Introduction

Horn-clause programs without function symbols, also known as *Datalog* program, are an important area of research in the theory of deductive databases. In recent research the problem of finding efficient evaluation methods for queries expressed as *Datalog* programs has been addressed.

In recent years, many methods concerned with recursive query processing have been proposed. Some of them are:

- Naive Evaluation (van Emden and Kowalski [1976])
- Semi-naive Evaluation (Bancilhon [1986])
- Iterative Query-subquery, Recursive query-subquery (Vieille [1986])
- SLD (Kowalski and Keuhner [1971]), SLD-AL (Vieille [1987])
- SLD with relation extension (Huang [1986])
- Henschen-Naqvi method (Henschen-Naqvi[1984])
- Iterative Compilation (van Emde Boas and van Emde Boas [1986])
- Algebraic Optimization (Hansen [1987], Houtsma et al [1988])
- Semantic Optimization (Sagiv [1988] , Jarke [1986])
- Static Filtering , Dynamic Filtering (Kifer and Lozinskii [1986])
- Aho-Ullman Method (Aho and Ullman [1979])
- APEX (Lozinskii [1985])
- Existential Optimization (Ramakrishnan et al [1988])
- Magic Sets (Bancilhon et al [1986])
- Counting, Reverse Counting (Sacca and Zaniolo [1986])
- Generalized Magic Sets (Beeri et al [1987])
- Generalized Supplementary Magic Sets (Beeri and Ramakrishnan [1987])
- Generalized Counting (Sacca and Zaniolo [1986])
- Global Optimization (Vieille [1989])
- Argument Reduction by Factoring
  (Naughton, Ramakrishnan, Sagiv, and Ullman [1989])

As may be clear from the size of the above list the proposed methods concerned with recursive query processing are so many that one often wonders which strategy would be efficient in specified cases. Moreover, it remains unknown whether and how these various techniques and methods could be combined and be compared. Therefore, in recent years, some classification methods about recursive query processing techniques have been presented ([Bancilhon and Ramakrishnan 1986], [Roelants 1987]). Based on the classifications concerning recursive query processing techniques, we expect that the following goals would be gained more efficiently:

i) To make comparisons among the existing techniques.
ii) To combine one technique with other techniques.
iii) To examine the potential of recursive query processing.
iv) To develop new recursive query processing techniques.
v) To provide a unified theory and strategy concerning recursive query processing.

In this paper, we would like to present a new classification about recursive query processing techniques, called *functional classification*, which differs from Bancilhon's classification based on external features and Roelants' classification based on halting conditions. The classification distinguishes the features of almost completeness, definiteness, and proper finegrainedness, which we see as the adequate criteria for classifications. A formalism of functional classification is presented. Based on the formalism, the composability of the existing techniques about recursive query processing is examined.

The organization of this paper is as follows. The section "The Existing Classifications" provides a general overview of existing classification methods. Their corresponding advantages and disadvantages are discussed. "Distinctions Between Evaluators and Optimizers" presents our first step towards the functional classification of recursive query processing techniques. The "Functional Classification" contains our general method. The corresponding classification dimensions are proposed. In "Composability among Evaluators and Optimizers" we examine the problem of composability among the existing techniques. Finally, in "Final Remarks" we conclude the proposed classification method.

2. Existing Classification

2.1 General Classification

It seems to be necessary to discuss what the adequate criteria for classifications are, before we examine the existing classifications. In our opinion, the following criteria may be adequate for the classification about recursive query processing techniques:

i) *Almost Completeness*: It can classify almost all existing techniques. Of course, an ideal criterion is completeness. However, it may be too strong to be satisfied. The complete set of techniques seems to be unattainable, because of the lack of a unified and precise description about recursive query processing.

ii) *Definiteness*: It can provide definite classification dimensions. This means that one technique can be classified into only one class.

iii) *Proper Finegrainedness:* The classification should be properly fine so that one can make a comparison between almost every two techniques. More precisely, every two techniques which are intuitively much different should be classified into different classes. Meanwhile, we should avoid a too fine classification, since that might result in a class for each concrete category.

Generally, the existing techniques can be simply categorized by the following three classes: Top-down, bottom-up, and mixed. We call the simple classification *Naive Classification.* The so-called *Bottom-up techniques* mean that we construct proof trees from the leaves up, working towards the root. *Top-down techniques* work downwards from the root to the leaves. For those familiar with Prolog, these techniques are sometimes referred to as "SLD-resolution" or "LUSH-resolution". Least Fixpoint Computation and Naive Evaluation, are closely related to bottom-up techniques. *Mixed techniques* are those techniques that combine one top-down technique with another bottom-up technique.

Naive classification obviously suffers from the disadvantage that it has no proper finegrainedness, because it is too coarse to make a comparison between every two techniques. However, there have been some more definite methods to classify existing recursive rule processing techniques ([Bancilhon and Ramakrishnan 1986], [Roelants 87] and [Bancilhon and Ramakrishnan 1988]).

2.2 Bancilhon's Classification

In [Bancilhon and Ramakrishnana 1986], Bancilhon and Ramakrishnan present the first attempt to definitely classify the existing techniques. A more complete description of Bancilhon's classification is introduced in [Bancilhon and Ramakrishnan 1988]. In [Bancilhon and Ramakrishnan 1988], a complete description of their understanding of several main strategies, application domains and general performance based on some specified examples are given. The proposed performance evaluation criteria for these algorithms are application domain (i.e. a class of rules for which it applies), performance (i.e. Time and Space cost), and ease of implementation. Moreover, algorithms concerned with recursive query processing are classified in top down versus bottom up, compiled versus interpretive, recursive versus iterative categories.

BANCILHON'S CLASSIFICATION

Technique	Classification
Naive Evaluation	Bottom-up, Compiled, Iterative
Semi-naive Evaluation	Bottom-up, Compiled, Iterative
Iterative Query/Subquery	Top-down, Interpreted, Iterative
Recursive Query/Subquery	Top-down, Interpreted, Recursive
Henschen-Naqvi	Top-down, Compiled, Iterative
Prolog	Top-down, Interpreted, Recursive
Aho-Ullman	_
Magic Sets	_
Static Filtering	_
Counting	_
Supplementary Sets	_
Existential Optimization	_
Redundance Elimination (Sagiv)	_

Figure 1. Bancilhon's Classification Examples

Notice that on the classification table above we only list some main (i.e. well known) techniques. Other techniques can be classified into corresponding classes if necessary. The indication "-" means that the method is difficult to be categorized.

In [Roelants 1987], it is argued that performance comparisons of general algorithms are useless because they have too many degrees of freedom, the performance comparisons of specific methods are very hard if we do not have a precise description of the methods and their implementation, and it is difficult not to favour one method above another by the choice of the test examples. Also, it is argued that the main problem with this classification into compiled and interpretive approaches is that the two operations, deduction and search, cannot always be distinguished clearly. In [Bancilhon and Ramakrishnan 1988], Bancilhon and Ramakrishnan also point out that some techniques are difficult to be categorized based on the above classification dimensions alone. Therefore, the classification based on external features suffers from two disadvantages namely no-almost completeness and no-definiteness.

2.3 Roelants' Classification

In Roelants [1987], query-answering algorithms that handle recursive rules in logic databases are classified according to their halting condition. Three classes are proposed: algorithms halting on solutions, algorithms halting on subqueries and algorithms halting on both solutions and subqueries. For each class a general algorithm is described. The general algorithms are compared on three criteria: halting, completeness, and efficiency.

For halting on solutions the general algorithm is naive evaluation. The general algorithm for halting on subqueries corresponds to the technique of SLD resolution. The general algorithm for halting on both solutions and subqueries is the mixed technique which combines top-down with bottom-up techniques. Therefore, as a matter of fact, the classification based on halting conditions is similar to the naive classification. However, the former provides a definite description and analysis.

However, the main problem with this classification according to halting conditions is that some strategies concerned with specified optimizations such as algebraic optimization, semantic optimization and existential optimization, cannot be easily classified because the halting conditions have close relationship with completeness, whereas those strategies place emphasis on transformation of recursive rules instead of evaluation of recursive queries.

ROELANTS' CLASSIFICATION

Techniques	Classification
Naive Evaluation	Halting on solutions
Semi-naive Evaluation	Halting on solutions
Iterative Query/Subquery	Halting on solutions and subqueries
Recursive Query/subquery	Halting on solutions and subqueries
Aho-Ullman	Halting on solutions
Henschen-Naqvi	Halting on subqueries
Prolog	Halting on subqueries
Static Filtering	Halting on solutions and subqueries
Magic Sets	Halting on solution and subqueries
Counting	Halting on subqueries
Supplementary Sets	Halting on solution and subqueries
Existential Optimization	_
Redundance Elimination(Sagiv)	_

Figure 2. Roelants' Classification Examples

3. Distinction Between Evaluator and Optimizer

Based on the analyses of the existing classifications, we conclude that it is frequently beneficial to draw a distinction between optimization and evaluation of queries, because the optimization of queries can be expressed in an independent manner, and can be incorporated in any evaluation method one may want. It seems to be more reasonable to view the Magic sets technique and its variants as optimization techniques instead of evaluation techniques. The existing classifications confuse evaluation techniques with optimization techniques. As a matter of fact, optimization techniques should not be classified by the same dimensions which are used to classify evaluation techniques.

Intuitively, we have the following distinctions:

Query evaluation: The process of semantically transforming a query Q into a corresponding set of tuples, based on the program.

Query optimization: The process of transforming a program P into a program P', on which queries can be evaluated more efficiently.

Query processing = Query evaluation and/or Query optimization.

Therefore, there are two distinct components in a general query processing strategy namely the Evaluator and the Optimizer:

Evaluator: **Query × Program —> Tuples**

Optimizer: **Query × Program —> Program**

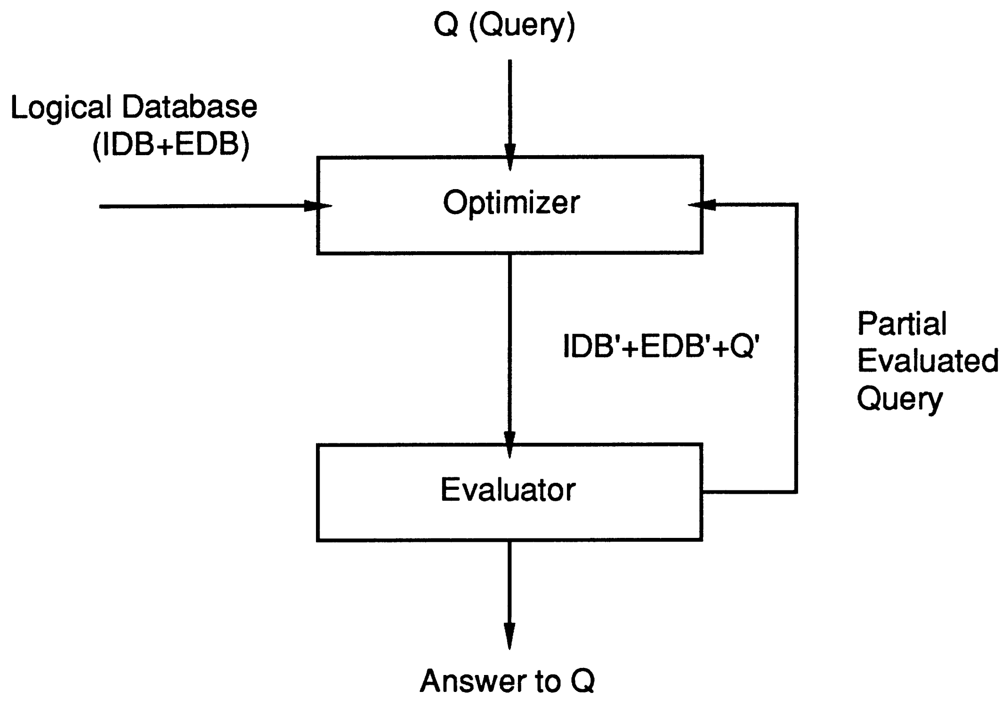

Figure 3. General Query Processing

However, in order to draw more clearly the distinction between Evaluators and Optimizers, we formally have the following definitions:

Formally, for a specified Datalog language **L**, a configuration of a logic database (CLDB), can be viewed as a four-place tuple, i.e.

CLDB=<IDB,EDB,Q,KF >
where IDB is a set of Horn-definite clauses of **L**

EDB is a set of facts of **L**,
Q is a query and
KF, called known facts, is a set of ground atomic formulas of **L**

Let CLDBs be the set of all configurations of **L**. An algorithm for query processing of logic databases can be viewed as a partial function from the set of configurations to itself. A CLDB is called an *initial logic database configuration* if its known facts are $\emptyset$.

Definition 3.1 *Application Domain* of an algorithm F, called AD, is Domain(F). i.e.
$F: AD \rightarrow CLDBs$
where $AD \subseteq CLDBs$

Definition 3.2 *Answer* to a query Q based on a program P (i.e. IDB $\cup$ EDB), is defined as :$Ans(Q)|P =_{df} \{ F \mid P \models F \text{ and } \exists\vartheta \ (Q\vartheta \equiv F\vartheta)\}$.

The distinctions between evaluators and optimizers mainly depend on whether the algorithms can obtain all answers to a query. If all answers to corresponding queries can be obtained, the algorithm would be an evaluator else it would be an optimizer as long as it can make some transformations of relevant rules or queries. Moreover, an optimizer should be able to obtain resultant rules which can be efficiently evaluated by any evaluators. The efficiency of evaluation should be one of the important criteria of optimizers. However, it is difficult to formalize the efficiency of evaluation without any precise description of algorithms and their implementations. Therefore, we have the following definitions:

Definition 3.3
$F: AD \rightarrow CLDBs$ is *a Generalized Evaluator* if
$(\forall C=<IDB,EDB,Q,KF>\in AD)(F(C)=<IDB',EDB',Q',KF'>$
$\Rightarrow Ans(Q)|\{IDB \cup EDB\} \subseteq KF')$

$F: AD \rightarrow CLDBs$ is *a Generalized Optimizer* if
$(\forall C=<IDB,EDB,Q, KF>\in AD)(F(C)=<IDB',EDB',Q',KF'>$
$\Rightarrow KF' \cup \quad Ans(Q')|\{IDB' \cup EDB'\}= KF \cup Ans(Q)|\{IDB \cup EDB\})$.

So far we still have not kept evaluators completely separate from optimizers. From the definitions above, some evaluators still can be viewed as optimizers., i.e, EVALUATORS $\cap$ OPTIMIZERS $\neq \emptyset$. Therefore, sometimes we call the evaluators and the optimizers which are defined as above the generalized ones. Of course, we can present more restricted definitions about evaluators and optimizers, by which optimizers and evaluators can be completely separated except for some trivial functions. The property will be shown in the next section (See Proposition 5.4).

Definition 3.4
$F: AD \rightarrow CLDBs$ is *a (pure) evaluator* if
$(\forall C=<IDB,EDB,Q,KF>\in AD)(F(C)=<IDB,EDB,Q,KF'>$
$\Rightarrow Ans(Q)|\{IDB \cup EDB\} \subseteq KF')$

$F: AD \rightarrow CLDBs$ is *a (pure) optimizer* if
$(\forall C=<IDB,EDB,Q, KF >\in AD)(F(C)=<IDB',EDB',Q',KF>$
$\Rightarrow Ans(Q')|\{IDB' \cup EDB'\}= Ans(Q)|\{IDB \cup EDB\})$.

Although most techniques and methods concerned with recursive query processing are lacking in the precise description of their algorithms and implementations, based on the general criterion above, we can still roughly divide most of the proposed techniques and methods concerned with recursive query processing into two main classes: Evaluators and Optimizers. Evaluators are : Naive Evaluation, Semi-naive Evaluation, Iterative Query-subquery, Recursive Query-subquery, SLD, SLD-AL, SLD with relation extensions and the Henschen-Naqvi method. Optimizers are: Iterative Compilation, Algebraic Optimization, Semantic Optimization, Static Filtering, Dynamic Filtering, Aho-Ullman Method, Existential Optimization, Magic Set, Counting, Reverse Counting, Generalized Magic Sets, Generalized Supplementary Magic Sets, Generalized Counting, Global Optimization and Argument reduction.

4. Functional Classification

4.1 Classification about Evaluators

In general, the efficient processing of recursive queries is implemented by two main approaches: firstly elimination of redundant computation by making full use of intermediate results, and secondly the limitation of necessary computation by transforming relevant rules and/or queries. (i.e. it should be able to focus on the data actually relevant for queries or rules). The former, called evaluation efficiency, is implemented in the course of evaluation of recursive queries whereas the latter, called logical efficiency, is implemented in the course of optimization of recursive queries by cutting down on the number of potentially relevant facts. It is usually difficult to implement the optimization of elimination of redundant computation, based on Least Fixpoint Evaluator in the course of optimization, since keeping the intermediate results is generally implemented in the course of evaluation and no strategies about this subject have been proposed so far.

For all evaluators of recursive queries, the following issues must be faced. First, algorithms must return all the answers to a query, which is called completeness. Secondly, it must halt after returning all answers, which is called termination. Finally, it must be evaluation efficient, i.e. it can eliminate all the redundant computations if possible.

As a matter of fact, existing classification methods concerned with algorithms of query processing, whether Bancilhon's and Ramakrishnan's classification or Roelants' classification, are methods about evaluators instead of optimizers because all of them place emphasis upon algorithmical features such as completeness, which have close relationship with evaluation. These existing classification methods have done well on the classification of evaluators.

However, anyone who attempts to classify existing strategies must face the largest difficulty, i.e., the lack of unified and precise descriptions about the existing techniques. Some techniques are even only roughly described. There are no any details about the proofs of correctness of those techniques. The difficulty seems to result in the impossibilities of any precise analysis and almost completeness. But, we argue that the functional classification is an adequate classification to solve the problem of the lack of unified and precise descriptions. Because, for any technique, the main goals on which the techniques focus should at least be described, no matter whether the details about the techniques are provided or not.

From the functional analysis point of view, the category of top-down vs bottom-up provides an adequate dimension for the classification, because it corresponds with the two main inference modes of Datalog, i.e., derivation vs generation. Meanwhile, the application domain also can be considered as an adequate dimension, because it describes a feature which generally has a close relationship with the functions of techniques. Therefore, for evaluators, the classification dimensions are inference mode and application domain.

CLASSIFICATION ABOUT EVALUATORS

Technique	Mode	Application Domain
Naive Evaluation	Bottom-up	General
Semi-naive Evaluation	Bottom-up	Linear, Acyclic
Iterative Query/Subquery	Top-down	General
Recursive Query/Subquery	Top-down	General
Henschen-Naqvi	Top-down	Linear
Prolog	Top-down	NSC

Figure 4. Classification about Evaluators

Note:
General = General evaluable rules.
Linear = Linear recursive rules.
Acyclic = Acyclic data.
NSC = No simple syntactic characterization

4.2. Classification of Optimizers

In order to cut down the number of potential relevant facts and make the evaluation of recursive queries more efficiently, optimizers should transform corresponding deduction rules of Datalog programs. According to their redundant computations relevant to queries or rules, optimization techniques can be classified into techniques based on queries and techniques based on rules.

Let OPTIMIZERS be the set of all optimizers for recursive query processing, and EVALUATORS be the set of all evaluators. Formally, we have the following definitions:

Definition 4.1

$F : AC \rightarrow CLDBs$ is an *Optimizer based on Rules* iff
$F : AC \rightarrow CLDBs \in OPTIMIZERS \wedge$
$(\forall C1=<IDB1,EDB1, Q1,KF1> \in AC) (\forall C2= <IDB2,EDB2,Q2,KF2> \in AC)$
$[IDB1=IDB2 \wedge EDB1=EDB2 \Rightarrow F(C1)=F(C2)]$.

$F : AC \rightarrow CLDBs$ is an *Optimizer based on Queries* iff
$F : AC \rightarrow CLDBs \in OPTIMIZERS \wedge$
$(\forall C=<IDB,EDB,Q,KF1> \in AC)(\exists C' =<IDB,EDB,Q', KF'> \in AC)$
$[F(C) \neq F(C')]$.

As far as the strategies of transforming rules are concerned, generally, there are three main approaches: static selection-pushing , semijoin optimization and the projection-pushing. Sometimes *semijoin optimization* is called dynamic selection-pushing. So called *selection-pushing optimizations* are those in which bound information is passed into the relevant rules in advance. *Projection-pushing optimizations* are those in which either redundant atomic formulas or redundant arguments are reduced. More concrete, we have the following examples:

- Push the selection statically: Static Filtering (Kifer and Lozinskin [1986]), Aho-Ullman method (Aho-Ullman [1979]), Algebraic Optimization (Hansen [1987],Houtsma et al [1988]).
- Push the selection dynamically: Dynamic Filtering (Kifer and Lozinskin [1986]), Magic Set, Counting (Bancilhon et al [1986]).
- Push the project : Existential optimization (Ramakrishnan et al [1988]).

Therefore, for optimizers, the adequate classification dimensions should be focus (Based on rules vs Based on query) and strategy (Selection-pushing, Semijoin, and Projection-pushing).

FUNCTIONAL CLASSIFICATION

Techniques	Classification
Naive Evaluation	(E)Bottom-up , General
Semi-naive Evaluation	(E)Bottom-up , Linear
Iterative Query/Subquery	(E)Top-down, General
Recursive Query/Subquery	(E)Top-down, General
Henschen-Naqvi	(E)Top-down, Linear
Prolog (SLD)	(E)Top-down , NSC
Aho-Ullman	(O)Based on Query,Push Selection
Static Filtering	(O)Based on Query,Push Selection
Magic Sets	(O)Based on Query, Semijoin
Counting	(O)Based on Query, Semijoin
Supplementary Sets	(O)Based on Query, Semijoin
Existential Optimization	(O)Based on Query,Push Projection
Redundant Elimination(Sagiv)	(O)Based on Rules,Push Projection

Figure 5. Functional Classification Examples

NOTE: (E)= EVALUATOR (O)= OPTIMIZER

5. Composability among Evaluators and Optimizers

In this section, we formally examine the problem of composability among evaluators and optimizers, by which we expect to achieve a better comprehension on the potential of composition among the existing techniques. Meanwhile, we expect that we would present a unified strategy by this approach.

Definition 5.1

For a query processing algorithm F: AD → CLDBs, and for any C=<IDB, EDB, Q, KF>, suppose that F(C)=<IDB', EDB', Q', KF'>. Let P= EDB∪IDB and P'= EDB'∪ IDB' then F is said to be:

(i)	*Initial*	iff KF=∅.
(ii)	*Intensional invariant*	iff IDB=IDB'.
(iii)	*Extensional invariant*	iff EDB=EDB'.
(iv)	*Query invariant*	iff Q=Q'.
(v)	*Known facts invariant*	iff KF= KF'.
(vi)	*Program invariant*	iff P = P'.
(vii)	*Explicit complete*	iff Ans(Q)| P ⊆ KF'.
(iix)	*Implicit complete*	iff Ans(Q)|P = Ans(Q')|P'.
(ix)	*Implicit weak complete*	iff KF ∪Ans(Q)|P = KF' ∪ Ans(Q')|P'.
(x)	*Trivial*	iff Ans(Q)|P ⊆ KF.
(xi)	*Identical*	iff IDB=IDB'∧ EDB=EDB' ∧ Q=Q' ∧ KF=KF'.

Let **Generalized_optimizers** be the set of all generalized optimizers, **Generalized_evaluators** be the set of all generalized evaluators, **Initial** be the set of all Initial algorithms and **Intensional_invariant** be the set of all intensional invariant algorithms. The others are defined analogously. We have the following results:

EVALUATORS = **Explicit_complete ∩ Program_invariant ∩ Query_invariant.**

OPTIMIZERS = **Implicit_complete ∩ Known_facts_invariant.**

Generalized_evaluators = Explicit_complete.

Generalized_optimizers = Implicit_complete.

Definition 5.2 (Composition of Algorithm Sets)

For any algorithm set S1, and S2, the composition of the sets S1 and S2, S1•S2, is defined as follows:

S1•S2 =$_{df}$ { F1° F2 | ∀F1 ∈ S1 and ∀F2 ∈ S2 and Range(F1) ⊆ Domain(F2)}.

where ° is the composition of functions.

Composition laws for evaluators and optimizers:

Proposition 5.1

(i) OPTIMIZERS • **Identical** = OPTIMIZERS
(ii) **Identical** • OPTIMIZERS = OPTIMIZERS
(iii) EVALUATORS• **Identical** = EVALUATORS
(iv) **Identical** • EVALUATORS = EVALUATORS
(v) **Identical** $\subset$ OPTIMIZERS
(vi) OPTIMIZERS•OPTIMIZER =OPTIMIZERS

Proposition 5.2

< OPTIMIZERS, ∘ > is a monoid.

Proposition 5.3

OPTIMIZERS • **Generalized_evaluators = Generalized_evaluators**

Proof.

For any F: AC $\rightarrow$ CLDBs $\in$ OPTIMIZERS,
and G: AC'$\rightarrow$ CLDBs $\in$ **Generalized_evaluators,**
and any C=<IDB,EDB,Q,KF> $\in$ AC,
suppose that F(C)=<IDB',EDB', Q',KF'>
and G(F(C))= <IDB'',EDB'', Q'', KF''>.

F is an optimizer and G is a Generalized evaluators $\Rightarrow$
Ans(Q)|{IDB$\cup$EDB} = Ans(Q')|{IDB' $\cup$ EDB'}
and Ans(Q'){IDB' $\cup$ EDB'} $\subseteq$ KF''$\Rightarrow$
Ans(Q)|{IDB $\cup$ EDB} $\subseteq$ FK''$\Rightarrow$
F∘G is a Generalized_evaluators.

Therefore, OPTIMIZERS • **Generalized_evaluators** $\subseteq$ **Generalized_evaluators.**

However, **Generalized_evaluators = Identical •Generalized_evaluators** $\subseteq$ OPTIMIZERS• **Generalized_evaluators.**

Therefore, OPTIMIZERS • **Generalized_evaluators = Generalized_evaluators.**

Proposition 5.4

OPTIMIZERS $\cap$ EVALUATORS = **Identical** $\cap$**Trivial.**

Proof.

OPTIMIZERS $\cap$ EVALUATORS $\subseteq$
Program_invariant $\cap$ **Query_invariant** $\cap$ **Known_facts_invariant** $\cap$
Explicit_complete $\subseteq$ **Identical** $\cap$**Trivial.**

Identical $\cap$ **Trivial** $\subseteq$ **Identical** $\cap$ **Explicit_complete** $\subseteq$
OPTIMIZERS $\cap$ **Explicit_complete** $\subseteq$
OPTIMIZERS $\cap$ EVALUATORS.

Therefore, OPTIMIZERS $\cap$ EVALUATORS = **Identical** $\cap$**Trivial.**

6. Final Remarks

We have proposed a functional classification as an attempt to categorize the existing techniques about recursive query processing. We benefit from drawing the distinctions between the evaluators and optimizers, because the composability among the existing techniques is more formally and precisely examined. We argued that the functional classification may be the only efficient approach to solve the problem since most techniques lack a precise and unified description.

However, in this paper, no performance comparisons are given. As Roelants point out, performance comparisons of general algorithms are useless because they have too many degrees of freedom. For performance comparisons of specific methods with specific data, in [Bancilhon and Ramakrishnan 1988], a satisfying description has been provided. Moreover, we do not discuss the problems of halting, completeness and efficiency of specified techniques as in [Roelants 1987]. That is because these features have a close relationship with evaluation, whereas for evaluators, in [Roelants 1987], the problems have been examined.

An important task for future research is to develop more definitely a unified strategy and theory about recursive query processing, which even can provide an implementation strategy for the efficient composition of the existing techniques.

Acknowledgements

The authors would like to thank Gerard R. Renardel de Lavalette and Karen Kwast for helpful discussions.

REFERENCES

[AU79]. Aho,A.and Ullman,J.D., Universality of Data Retrieval Languages, *Proc. 6th ACM Symposium on Principles of Programming languages*, 1979, 110-120.

[Ba86]. Bancilhon, F., Naive evaluation of recursively defined relations, in *On knowledge Base Management Systems -- Integrated Database and AI Systems* (M. Brodie and J. Mylopoulos, Eds.), Spring-Verlag, 1986, 165-178.

[BMSU86]. Bancilhon,F., Maier, D., Sagiv, Y., and Ullman, J. D. Magic Sets and Other Strange Ways to Implement Logic Programs, *Proc. 5th ACM SIGMOD-SIGACT Symposium on Principles of Database systems*, 1986, 1-15.

[BR86]. Bancilhon,F., Ramakrishnan, R., An Amateur's Introduction to Recursive Query Processing Strategies, *Proc. of ACM-Sigmod International Conference on Management of Data*, Washington, May 1986, 16-52.

[BR88]. Bancilhon, F., Ramakrishnan, R. Performance Evaluation of Logic Programs, in *Foundations of Deductive Databases and Logic Programming* (J. Minker,Ed.), Morgan Kaufmann Publishers, Los Altos, CA, 1988, 439-517.

[BR87]. Beeri, C., Ramakrishnan,R. On the Power of Magic, Proc. 6th ACM-SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 1987, 269-283.

[Ch81]. Change, C. On the Evaluation of Queries Containing Derived Relations in Relational Databases, in Advances in Data Base Theory, Vol.1 (H. Gallaire, J.Minker, and J.M. Nicolas,Eds.), Plenum Press, New York, 1981, 235-260.

[Ch86]. Chakravarthy,U.S.,et al. Semantic Query Optimization in Expert Systems and Database Systems, in Expert Database Systems, *Proc.from 1st International Conference,* (Kerschberg,L., Ed.), The Benjamin/cummings Publishing Comany, INC. 1986, 659-674.

[DE88]. van Denneheuvel, S., van Emde Boas, P., Towards Implementing RL. Preprint CT-88-10, Institute for Language, Logic and Information, University of Amsterdam, 1988.

[Em86]. van Emde Boas, P., RL, a Language for Enhanced Rules Bases Database Processing, Working Document, Rep IBM Research, RJ4869(51299), 1986.

[EE86]. van Emde Boas, H., van Emde Boas, P., Storing and Evaluating Horn-Clause Rules in a Relational Database, *IBM J. Res. Develop.* 30(1) , 1986, 80-92.

[EK76]. Emden, M., H., Kowalski, R.A. The Semantics of Predicate Logic as a Programming Language, *J. ACM* 23(4), 1976, 733-742.

[Ha87]. Hansen, M. R. Algebraic of Optimization of Recursive Database Queries, *Proc. CIPS'87,* 1987.

[HN84]. Henschen, L., Naqvi, S. On Compiling Queries in Recurtsive First Order Data Bases, *J. ACM* 31,1984, 47-85.

[HKFAK88]. Houtsma, M. A. W., van Kuijk, H. J. A., Flokstra, J., Apers, P.M.G. Kersten, M. L., A logic Query Languages and its Algebraic Optimization for a Multiprocessor Database Machine, University of Twente, 1988.

[Hu86]. Huang, Z.S. Searching Strategy of Two-level Deductive Database TLDB, *J. ZSI* 2, 1986.

[Ja86]. Jarke, M., External Semantic Query Simplification: A Graph-theoretic Approach and its Implementation in Prolog, in *Expert Database Systems, Proc. from 1st International Conf.*, (Kerschberg,L., Ed.), The Benjamin/Cummings Publishing Company,INC., 1986, 675-692.

[KL86]. Kifer, M., Lozinskii, E, Filtering Data Flow in Deductive Databases, *Proc. international Conference on Database Theory*, LNCS, No. 243, Spring-verlag, 1986, 186-202.

[KK71]. Kowalski,R. A. and D. Kuehner, Linear resolution with selection function, *Artificial Intelligence* 2, 1971, 227-260.

[Lo85]. Lozinskii, E. Evaluating Queries in Deductive Databases by Gnerating, *Proc. 11th international Joint conference on AI*, 1985, 173-177.

[Mi88]. Minker, J., Perspectives in Deductive Databases , *J. Logic Programming*, 5, 1988, 33-60.

[NRSU89]. J.F. Naughton, R. Ramkrishnan, Y. Sagiv, J.D. Ullman, Argument Reduction by Factoring, *Proc. of the 15th International Conference on Very Large Databases*, 1989, 173-182.

[RBK88]. Ramakrishnan, R., Beeri, C., Krishnamurthy, R. Optimizing Existential Datalog Queries. *Proc. 7th ACM-SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 1988, 89-102.

[Ro87]. Roelants, D. Recursive Rules in Logic Databases, Research Report, Philips Research Laboratory, March 1987.

[SZ86]. Sacca, D. and Zaniolo, C. On the Implementation of a Simple Class of Logic Queries for Databases, *Proc. 5th ACM SIGMOD-SIGART Symposium on Principle of Database Systems*, 1986, 16-23.

[Sa88]. Sagiv, Y. Optimizing Datalog Program, in *Foundations of Deductive Databases and Logic Programming* (J., Minker, Ed.) , Morgan Kaufmann Publishers Los Alots, CA, 1988, 627-658.

[Ul89] Ullman, J. D., *Principles of database and knowledge-base systems, Vol.II: The new technologies*,Computer Science Press, 1989.

[Vi86]. Vieille, L. Recursive Axioms in Deductive Databases: The Query/Subquery Approach, *Proc. First Intl. conference on Expert Database Systems*, Charleston, 1986, 179-194.

[Vi87]. Vieille, L. Database-complete Proof Procedure Based on SLD-resolution. *Proc. of the 4th Inter. Conf. on Logic Programming*, Melbourne, Australia, May, 1987, 74-103.

[Vi89]. Vieille, L., From QSQ towards QoSaQ: Global Optimization of Recursive Queries, in *Expert Database Systems , Proc. from 2nd Internationl Conference*, (Kerschberg, L., Ed.), The Benjamin/Cummings Publishing Company, INC.,1989, 743-778.

The ITLI Prepublication Series

1986		
86-01		The Institute of Language, Logic and Information
86-02	Peter van Emde Boas	A Semantical Model for Integration and Modularization of Rules
86-03	Johan van Benthem	Categorial Grammar and Lambda Calculus
86-04	Reinhard Muskens	A Relational Formulation of the Theory of Types
86-05	Kenneth A. Bowen, Dick de Jongh	Some Complete Logics for Branched Time, Part I Well-founded Time,
86-06	Johan van Benthem	Logical Syntax Forward looking Operators
1987		
87-01	Jeroen Groenendijk, Martin Stokhof	Type shifting Rules and the Semantics of Interrogatives
87-02	Renate Bartsch	Frame Representations and Discourse Representations
87-03	Jan Willem Klop, Roel de Vrijer	Unique Normal Forms for Lambda Calculus with Surjective Pairing
87-04	Johan van Benthem	Polyadic quantifiers
87-05	Víctor Sánchez Valencia	Traditional Logicians and de Morgan's Example
87-06	Eleonore Oversteegen	Temporal Adverbials in the Two Track Theory of Time
87-07	Johan van Benthem	Categorial Grammar and Type Theory
87-08	Renate Bartsch	The Construction of Properties under Perspectives
87-09	Herman Hendriks	Type Change in Semantics: The Scope of Quantification and Coordination
1988		*Logic, Semantics and Philosophy of Language:*
LP-88-01	Michiel van Lambalgen	Algorithmic Information Theory
LP-88-02	Yde Venema	Expressiveness and Completeness of an Interval Tense Logic
LP-88-03		Year Report 1987
LP-88-04	Reinhard Muskens	Going partial in Montague Grammar
LP-88-05	Johan van Benthem	Logical Constants across Varying Types
LP-88-06	Johan van Benthem	Semantic Parallels in Natural Language and Computation
LP-88-07	Renate Bartsch	Tenses, Aspects, and their Scopes in Discourse
LP-88-08	Jeroen Groenendijk, Martin Stokhof	Context and Information in Dynamic Semantics
LP-88-09	Theo M.V. Janssen	A mathematical model for the CAT framework of Eurotra
LP-88-10	Anneke Kleppe	A Blissymbolics Translation Program
		Mathematical Logic and Foundations:
ML-88-01	Jaap van Oosten	Lifschitz' Realizabiility
ML-88-02	M.D.G. Swaen	The Arithmetical Fragment of Martin Löf's Type Theories with weak Σ-elimination
ML-88-03	Dick de Jongh, Frank Veltman	Provability Logics for Relative Interpretability
ML-88-04	A.S. Troelstra	On the Early History of Intuitionistic Logic
ML-88-05	A.S. Troelstra	Remarks on Intuitionism and the Philosophy of Mathematics
		Computation and Complexity Theory:
CT-88-01	Ming Li, Paul M.B.Vitanyi	Two Decades of Applied Kolmogorov Complexity
CT-88-02	Michiel H.M. Smid	General Lower Bounds for the Partitioning of Range Trees
CT-88-03	Michiel H.M. Smid, Mark H. Overmars Leen Torenvliet, Peter van Emde Boas	Maintaining Multiple Representations of Dynamic Data Structures
CT-88-04	Dick de Jongh, Lex Hendriks Gerard R. Renardel de Lavalette	Computations in Fragments of Intuitionistic Propositional Logic
CT-88-05	Peter van Emde Boas	Machine Models and Simulations (revised version)
CT-88-06	Michiel H.M. Smid	A Data Structure for the Union-find Problem having good Single-Operation Complexity
CT-88-07	Johan van Benthem	Time, Logic and Computation
CT-88-08	Michiel H.M. Smid, Mark H. Overmars Leen Torenvliet, Peter van Emde Boas	Multiple Representations of Dynamic Data Structures
CT-88-09	Theo M.V. Janssen	Towards a Universal Parsing Algorithm for Functional Grammar
CT-88-10	Edith Spaan, Leen Torenvliet, Peter van Emde Boas	Nondeterminism, Fairness and a Fundamental Analogy
CT-88-11	Sieger van Denneheuvel, Peter van Emde Boas	Towards implementing RL
		Other prepublications:
X-88-01	Marc Jumelet	On Solovay's Completeness Theorem
1989		*Logic, Semantics and Philosophy of Language:*
LP-89-01	Johan van Benthem	The Fine-Structure of Categorial Semantics
LP-89-02	Jeroen Groenendijk, Martin Stokhof	Dynamic Predicate Logic, towards a compositional, non-representational semantics of discourse
LP-89-03	Yde Venema	Two-dimensional Modal Logics for Relation Algebras and Temporal Logic of Intervals
LP-89-04	Johan van Benthem	Language in Action
LP-89-05	Johan van Benthem	Modal Logic as a Theory of Information
LP-89-06	Andreja Prijatelj	Intensional Lambek Calculi: Theory and Application
LP-89-07	Heinrich Wansing	The Adequacy Problem for Sequential Propositional Logic
LP-89-08	Víctor Sánchez Valencia	Peirce's Propositional Logic: From Algebra to Graphs
LP-89-09	Zhisheng Huang	Dependency of Belief in Distributed Systems
		Mathematical Logic and Foundations:
ML-89-01	Dick de Jongh, Albert Visser	Explicit Fixed Points for Interpretability Logic
ML-89-02	Roel de Vrijer	Extending the Lambda Calculus with Surjective Pairing is conservative
ML-89-03	Dick de Jongh, Franco Montagna	Rosser Orderings and Free Variables
ML-89-04	Dick de Jongh, Marc Jumelet, Franco Montagna	On the Proof of Solovay's Theorem
ML-89-05	Rineke Verbrugge	Σ-completeness and Bounded Arithmetic
ML-89-06	Michiel van Lambalgen	The Axiomatization of Randomness
ML-89-07	Dirk Roorda	Elementary Inductive Definitions in HA: from Strictly Positive towards Monotone
ML-89-08	Dirk Roorda	Investigations into Classical Linear Logic
ML-89-09	Alessandra Carbone	Provable Fixed points in $I\Delta_0+\Omega_1$
		Computation and Complexity Theory:
CT-89-01	Michiel H.M. Smid	Dynamic Deferred Data Structures
CT-89-02	Peter van Emde Boas	Machine Models and Simulations
CT-89-03	Ming Li, Herman Neuféglise, Leen Torenvliet, Peter van Emde Boas	On Space Efficient Simulations
CT-89-04	Harry Buhrman, Leen Torenvliet	A Comparison of Reductions on Nondeterministic Space
CT-89-05	Pieter H. Hartel, Michiel H.M. Smid Leen Torenvliet, Willem G. Vree	A Parallel Functional Implementation of Range Queries
CT-89-06	H.W. Lenstra, Jr.	Finding Isomorphisms between Finite Fields
CT-89-07	Ming Li, Paul M.B. Vitanyi	A Theory of Learning Simple Concepts under Simple Distributions and Average Case Complexity for the Universal Distribution (Prel. Version)
CT-89-08	Harry Buhrman, Steven Homer Leen Torenvliet	Honest Reductions, Completeness and Nondeterminstic Complexity Classes
CT-89-09	Harry Buhrman, Edith Spaan, Leen Torenvliet	On Adaptive Resource Bounded Computations
CT-89-10	Sieger van Denneheuvel	The Rule Language RL/1
CT-89-11	Zhisheng Huang, Sieger van Denneheuvel Peter van Emde Boas	Towards Functional Classification of Recursive Query Processing
		Other Prepublications:
X-89-01	Marianne Kalsbeek	An Orey Sentence for Predicative Arithmetic
X-89-02	G. Wagemakers	New Foundations: a Survey of Quine's Set Theory
X-89-03	A.S. Troelstra	Index of the Heyting Nachlass
X-89-04	Jeroen Groenendijk, Martin Stokhof	Dynamic Montague Grammar, a first sketch
X-89-05	Maarten de Rijke	The Modal Theory of Inequality
X-89-06	Peter van Emde Boas	Een Relationele Semantiek voor Conceptueel Modelleren: Het RL-project