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-Naqui method (Henschen-Naqui[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 Ramakrishnan 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

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

*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

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

**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:

\[
\text{Evaluator: } \quad \text{Query} \times \text{Program} \rightarrow \text{Tuples}
\]

\[
\text{Optimizer: } \quad \text{Query} \times \text{Program} \rightarrow \text{Program}
\]

---

**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.

\[
\text{CLDB} = \langle \text{IDB}, \text{EDB}, Q, \text{KF} \rangle
\]

where \( \text{IDB} \) 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 ∅.

Definition 3.1 Application Domain of an algorithm F, called AD, is Domain(F). i.e.
F: AD → CLDBs
where AD ⊆ CLDBs

Definition 3.2 Answer to a query Q based on a program P (i.e. IDB ∪ EDB), is defined as :Ans(Q)\|P =df { F | P ⊨ F and ∃θ (Qθ = Fθ)}.

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 → CLDBs is a Generalized Evaluator if
(∀C=IDB,EDB,Q,KF)∈ AD)(F(C)<IDB',EDB',Q',KF'>
⇒ Ans(Q)(IDB ∪ EDB) ⊆ KF')

F: AD → CLDBs is a Generalized Optimizer if
(∀C=IDB,EDB,Q,KF)∈ AD)(F(C)<IDB',EDB',Q',KF'>
⇒ KF' ∪ Ans(Q)(IDB ∪ EDB) = KF ∪ Ans(Q)(IDB ∪ 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 ∩ OPTIMIZERS ≠ ∅. 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 → CLDBs is a (pure) evaluator if
(∀C=IDB,EDB,Q,KF)∈ AD)(F(C)<IDB,EDB,Q,KF'>
⇒ Ans(Q)(IDB ∪ EDB) ⊆ KF')

F: AD → CLDBs is a (pure) optimizer if
(∀C=IDB,EDB,Q,KF)∈ AD)(F(C)<IDB',EDB',Q',KF'>
⇒ Ans(Q')(IDB' ∪ EDB') = Ans(Q)(IDB ∪ 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 algorithmic 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**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Mode</th>
<th>Application Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Evaluation</td>
<td>Bottom-up</td>
<td>General</td>
</tr>
<tr>
<td>Semi-naive Evaluation</td>
<td>Bottom-up</td>
<td>Linear, Acyclic</td>
</tr>
<tr>
<td>Iterative Query/Subquery</td>
<td>Top-down</td>
<td>General</td>
</tr>
<tr>
<td>Recursive Query/Subquery</td>
<td>Top-down</td>
<td>General</td>
</tr>
<tr>
<td>Henschen-Naqvi</td>
<td>Top-down</td>
<td>Linear</td>
</tr>
<tr>
<td>Prolog</td>
<td>Top-down</td>
<td>NSC</td>
</tr>
</tbody>
</table>

*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**

\[ F : AC \to CLDBs \text{ is an } \text{Optimizer based on Rules } \iff F : AC \to CLDBs \in OPTIMIZERS \land \\
(\forall C_1=\langle IDB_1, EDB_1, Q_1, KF_1 \rangle \in AC) (\forall C_2=\langle IDB_2, EDB_2, Q_2, KF_2 \rangle \in AC) \\
[IDB_1=IDB_2 \land EDB_1=EDB_2 \Rightarrow F(C_1)=F(C_2)]. \]

**Definition 4.2**

\[ F : AC \to CLDBs \text{ is an } \text{Optimizer based on Queries iff} F : AC \to CLDBs \in OPTIMIZERS \land \\
(\forall C=\langle IDB, EDB, Q, KF \rangle \in AC)(\exists C'=\langle IDB, EDB, Q', KF' \rangle \in AC) \\
[F(C) \neq F(C')]. \]
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 semi-join 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 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**

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

**Figure 5. Functional Classification Examples**

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

11
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 $F: \text{AD} \rightarrow \text{CLDBs}$, and for any $C = \langle \text{IDB}, \text{EDB}, Q, KF \rangle$, suppose that $F(C) = \langle \text{IDB}', \text{EDB}', Q', KF' \rangle$. Let $P = \text{EDB} \cup \text{IDB}$ and $P' = \text{EDB}' \cup \text{IDB}'$ then $F$ is said to be:

(i) \textit{Initial} \iff KF = \emptyset.
(ii) \textit{Intensional invariant} \iff \text{IDB} = \text{IDB}'.
(iii) \textit{Extensional invariant} \iff \text{EDB} = \text{EDB}'.
(iv) \textit{Query invariant} \iff Q = Q'.
(v) \textit{Known facts invariant} \iff KF = KF'.
(vi) \textit{Program invariant} \iff P = P'.
(vii) \textit{Explicit complete} \iff \text{Ans}(Q) \cup P \subseteq KF'.
(viii) \textit{Implicit complete} \iff \text{Ans}(Q) \cup P = \text{Ans}(Q') \cup P'.
(ix) \textit{Implicit weak complete} \iff \text{KF} \cup \text{Ans}(Q) \cup P = \text{KF'} \cup \text{Ans}(Q') \cup P'.
(x) \textit{Trivial} \iff \text{Ans}(Q) \cup P \subseteq \text{KF}.
(xi) \textit{Identical} \iff \text{IDB} = \text{IDB}' \land \text{EDB} = \text{EDB}' \land Q = Q' \land KF = KF'.

Let \textit{Generalized optimizers} be the set of all generalized optimizers, \textit{Generalized evaluators} be the set of all generalized evaluators, \textit{Initial} be the set of all Initial algorithms and \textit{Intensional invariant} be the set of all intensional invariant algorithms. The others are defined analogously. We have the following results:

\[
\text{EVALUATORS} = \text{Explicit complete} \cap \text{Program invariant} \cap \text{Query invariant}.
\]

\[
\text{OPTIMIZERS} = \text{Implicit complete} \cap \text{Known facts invariant}.
\]

\[
\text{Generalized evaluators} = \text{Explicit complete}.
\]

\[
\text{Generalized optimizers} = \text{Implicit complete}.
\]

Definition 5.2 (Composition of Algorithm Sets)

For any algorithm set $S_1$, and $S_2$, the composition of the sets $S_1$ and $S_2$, $S_1 \circ S_2$, is defined as follows:

$S_1 \circ S_2 = \{ F_1 \circ F_2 \mid \forall F_1 \in S_1 \text{ and } \forall F_2 \in S_2 \text{ and } \text{Range}(F_1) \subseteq \text{Domain}(F_2) \}$.

where $\circ$ is the composition of functions.

Composition laws for evaluators and optimizers:
Proposition 5.1
(i) \textsc{Optimizers \cdot Identical} = \textsc{Optimizers}
(ii) \textsc{Identical \cdot Optimizers} = \textsc{Optimizers}
(iii) \textsc{Evaluator\textsc{s} \cdot Identical} = \textsc{Evaluator\textsc{s}}
(iv) \textsc{Identical \cdot Evaluator\textsc{s}} = \textsc{Evaluator\textsc{s}}
(v) \textsc{Identical} \subseteq \textsc{Optimizers}
(vi) \textsc{Optimizers \cdot Optimizer} = \textsc{Optimizers}

Proposition 5.2
\langle \textsc{Optimizers}, \circ \rangle \text{ is a monoid.}

Proposition 5.3
\textsc{Optimizers \cdot Generalized_evaluators} = \textsc{Generalized_evaluators}

Proof.
For any \( F: AC \rightarrow \text{CLDBs} \in \text{Optimizers} \),
and \( G: AC' \rightarrow \text{CLDBs} \in \text{Generalized_evaluators} \),
and any \( C=\langle \text{IDB, EDB, Q, KF} \rangle \in AC \),
suppose that \( F(C)=\langle \text{IDB', EDB', Q', KF'} \rangle \)
and \( G(F(C))=\langle \text{IDB'', EDB'', Q'', KF''} \rangle \).

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

Therefore, \( \text{Optimizers \cdot Generalized_evaluators} \subseteq \text{Generalized_evaluators} \).

However, \( \text{Generalized_evaluators} = \text{Identical \cdot Generalized_evaluators} \subseteq \text{Optimizers \cdot Generalized_evaluators} \).

Therefore, \( \text{Optimizers \cdot Generalized_evaluators} = \text{Generalized_evaluators} \).

Proposition 5.4
\textsc{Optimizers \cap Evaluators} = \textsc{Identical \cap Trivial}.

Proof.
\textsc{Optimizers \cap Evaluators} \subseteq
\textsc{Program_invariant \cap Query_invariant \cap Known_facts_invariant \cap Explicit_complete} \subseteq \textsc{Identical \cap Trivial}.

\textsc{Identical \cap Trivial} \subseteq \textsc{Identical \cap Explicit_complete} \subseteq \textsc{Optimizers \cap Explicit_complete} \subseteq \textsc{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 compositability 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


The ITLI Prepublication Series

1986
86-01 The Institute of Language, Logic and Information
86-02 Peter van Emde Boas
86-03 Johan van Benthem
86-04 Reinhard Muskens
86-05 Kenneth A. Bowen, Dick de Jongh
86-06 Johan van Benthem
86-07 Renate Bartsch
87-01 Jan Willem Klop, Roel de Vrijer
87-02 Johan van Benthem
87-03 Victor Sánchez Valencia
87-04 Eelco Oversteegen
87-05 Eelco Oversteegen
87-06 Jeroen Groenendijk, Martin Stokhof
87-07 Johan van Benthem
87-08 Renate Bartsch
87-09 Henk Verbrugge

1988
LP-88-01 Michiel van Lambalgen Logic, Semantics and Philosophy of Language
LP-88-02 Yde Venema Year Report 1987
LP-88-03
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 Eutrofa
LP-88-10 Anneke Kleppe A Blasiusycolics Translation Program
ML-88-01 Jaap van Oosten Mathematical Logic and Foundations: Lifschitz' Realizability
ML-88-02 M.D.G. Swaen The Arithmetical Fragment of Martin Lof's Type Theories with weak \( \Sigma \)-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
CT-88-01 Ming Li, Paul M.B. Vitanyi Computation and Complexity Theory: 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 Maintaining Multiple Representations of Dynamic Data Structures
CT-88-04 Dick de Jongh, Lex Hendriks 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 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
X-88-01 Marc Jumelet Other prepublications: On Solovay's Completeness Theorem

1989
LP-89-01 Johan van Benthem Logic, Semantics and Philosophy of Language: The Fine-Structure of Categorial Semantics
LP-89-02 Jeroen Groenendijk, Martin Stokhof Dynamic Predicative Logic, towards a compositional, non-representational semantics of discourse
LP-89-03 Yde Venema Two-dimensional Modal Logics for Relaion 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 Lambeck Calculi: Theory and Application
LP-89-07 Heinrich Wansing The Adequacy Problem for Sequential Propositional Logic
LP-89-08 Victor Sánchez Valencia Peirce's Propositional Logic: From Algebra to Graphs
LP-89-09 Zhisheng Huang Dependency of Belief in Distributed Systems
ML-89-01 H. Wille, Albert Visser Mathematical Logic and Foundations: Explicite Fixed Points for Interpretation Logical Sequents
ML-89-02 Roel de Vrijer Extending the Lambda Calculus with Subjective Pairing is conservative
ML-89-03 Dick de Jongh, Franso 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 Rinke Verbrugge \( \Sigma \)-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 Provability Fixed points in \( \Gamma_0 + \Omega \)
CT-89-01 Michiel H.M. Smid Computation and Complexity Theory: Dynamic Deferred Data Structures
CT-89-02 Peter van Emde Boas Machine Models and Simulations
CT-89-03 Ming Li, Herman Neufglist, Leen Torenvliet, Peter van Emde Boas On Space Efficient Simulations
CT-89-04 Harry Buhrman, Leen Torenvliet, Peter van Emde Boas A Comparison of Reductions on Nondeterministic Space
CT-89-05 Pieter H. Hartel, Michiel H.M. Smid 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 Finding Isomorphisms between Finite Fields
CT-89-10 Sieger van Denneheuvel The Rule Language RL/1
CT-89-11 Zhisheng Huang, Sieger van Denneheuvel Towards Functional Classification of Recursive Query Processing

Peter van Emde Boas Other Prepublications: An Orey Sentence for Predicative Arithmetic
X-89-01 Marianne Kaisbeek New Foundations: a Survey of Quine's Set Theory
X-89-02 G. Wagmakers Index of the Heyting Nachlass
X-89-03 A.S. Troelstra Dynamic Montague Grammar, a first sketch
X-89-04 Jeroen Groenendijk, Martin Stokhof The Modal Theory of Inequality
X-89-05 Maarten de Rijke Een Relationele Semantiek voor Conceptueel Modelleren: Het RL-project