Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

364

J. Demetrovics B. Thalheim (Eds.)

MFDBS 89

2nd Symposium on
Mathematical Fundamentals of Database Systems
Visegrád, Hungary, June 26–30, 1989
Proceedings

Springer-Verlag

Berlin Heidelberg New York London Paris Tokyo Hong Kong

Table of Contents

Selective refutation of integrity constraints in deductive databases.....1
P.Asirelli, C.Billi, P.Inverardi

Approaches to updates over weak instances.....12
P.Atzeni, R.Torlone

Index selection in relational databases.....24
E.Barcucci, A.Chiuderi, R.Pinzani, M.C.Verri

Towards a schema design methodology for deductive databases.....37
J.Biskup, B.Convent

Shared abstract data types: An algebraic methodology for their specification.....53
A.Bondavalli, N.De Francesco, D.Latella, G.Vaglini

Specifying closed world assumptions for logic databases.....68
S.Brass, U.W.Lipeck

Interaction of authorities and acquaintances in the DORIS privacy model of data.....85
H.H.Brüggemann

Logical rewritings for improving the evaluation of quantified queries.....100
F.Bry

Mathematical foundations of semantic networks theory.....117
M.Burgin, V.Gladun

Functional dependencies and the semilattice of closed classes.....136
J.Demetrovics, L.O.Libkin, I.B.Muchnik

An extended view on data base conceptual design.....148
M.D.Draghici

Modeling planning problems.....172
A.E.Eiben

On the interaction between transitive closure and functional dependencies.....187
G.Gottlob, M.Schrefl, M.Stumptner

A strategy for executing complex queries.....207
E.Grazzini, F.Pippolini

Multiple task selection protocol in a distributed problem solving network.....222
T.Gyires

Equivalent schemes in semantic, nested relational, and relational database models....237
A.Heuer

Covers for functional independencies 254
J.M.Janas

Restructuring and dependencies in databases....269
E.A.Komissartschik

RTL - A relation and table language for statistical databases....285
L.Lakhal, R.Cicchetti, S.Miranda

Integration of functions in the fixpoint semantics of rule-based systems....301
E.Lambrichts, P.Nees, J.Paredaens, P.Peelman, L.Tanca

Locking policies and predeclared transactions....317
G.Lausen, E.Soisalon-Soininen

Means for management of relational fuzzy data bases - way to merging of systems of data bases and knowledge bases 337
T.A.Malyuta, V.V.Pasichnik, A.A.Stogniy

A specification language for static, dynamic and deontic integrity constraints....347
J.-J.Meyer, H.Weigand, R.Wieringa

Blocks and projections' synthesis in relational databases....367
L.A.Tenenbaum

The higher-order entity-relationship model and $(DB)^2$ 382
B.Thalheim

Goal-oriented concurrency control....398
V.Vianu, G.Vossen

Transitive closure and the LOGA$^+$-strategy for its efficient evaluation....415
W.Yan, N.Mattos

Logical Rewritings for Improving the Evaluation of Quantified Queries

François Bry

ECRC, Arabellastr. 17, D - 8000 München 81, West Germany

uucp: ...!pyramid!ecrcvax!fb

ABSTRACT *We describe a new approach for improving the evaluation of queries with quantifiers. We first introduce the concept of* constructive evaluation *as a formalization of a principle common to the methods that have been proposed for answering quantified queries. Relying on this concept, we define the class of* constructively domain independent *(cdi) formulas. We show that cdi queries can be constructively evaluated by searching only the relations they refer to. Therefore, cdi queries admit domain independent evaluations as soon as they do not explicitly refer to the database domain. Then, we define rewritings that translate general quantified queries into expressions amenable to efficient constructive evaluations. These rewritings preserve logical equivalence, hence do not compromise the semantics of queries. They are based on the generation from a query of the* ranges *and* co-ranges *of its variables. We show that the rewritings we propose permit to optimize the evaluation of cdi queries. They also reduce queries from several solvable subclasses of domain independent formulas to cdi queries, hence permitting their domain independent evaluation.*

1. Introduction

Evaluating queries that are more general than simple conjunctions is often needed for database applications. Queries with quantified variables are useful for investigating refined relationship between data, in particular for expressing integrity constraints. If queries with quantifiers are not frequent in business or management databases, they are common in the emerging areas of scientific and statistic databases. Quantified queries and constraints are also investigated in the framework of research on knowledge bases.

In this article, we first introduce the concept of *constructive evaluation* for formalizing into logic the principles of database methods that were proposed for answering quantified queries. Intuitively, an answering procedure is constructive if it process quantifications by instantiating the variables over the database domain or over a relevant part of it, defined, e.g., by types or range expressions associated with the variables. Methods that reduce the evaluation of quantified queries to the generation of extensional answers to quantifier-free expressions are constructive. We show that the methods based on relational algebra [COD 72, PAL 72, JS 82, BRY 89a] as well as the non-algebraic processing of quantifiers proposed in [DAY 83, DAY 87] are constructive. We indicate directions for research that could lead to non-constructive methods for evaluating queries with quantifiers.

Relying on the concept of constructive evaluation, we introduce the class of *constructively domain independent* formulas (short, cdi formulas). Cdi queries can be constructively evaluated against a

database conforming to the Domain Closure Axiom by searching only the relations they mention. Therefore, cdi queries admit domain independent evaluations as soon as they do not explicitly refer to the database domain.

We then describe a method for translating quantified queries of any kind into expressions amenable to constructive evaluation. This transformation preserves logical equivalence and therefore does not compromise the semantics of queries. It is applicable either for processing non-cdi queries with a constructive evaluation method, or for improving the evaluation of cdi queries. It is based on a technique for generating from a query, the candidate ranges for its quantified variables. Conventional cost estimation techniques, e.g., based on relation sizes and on physical data structures, can then be used for chosing ranges permitting efficient evaluations.

We define the quantified query transformations by means of rewriting rules. This Artificial Intelligence technique is particularly well-suited to defining transformations of programs in general, of declarative queries in particular. Systems of rewriting rules can easily be compiled into deterministic procedures that are more convenient for practical use. However, the rule formalism is more appropriate to formal investigations and to simple descriptions. Refer to [HUE 80, SCH 87] for an introduction to the concepts of rewriting systems.

To a certain extend, generating ranges can be considered similar to investigating the different ordering of relations in a conjunctive query. However, though the later is simple, the former surprisingly turns out to be rather complex. If generating ranges for quantifier-free queries is already known in the literature, as far as we know the issue has not been investigated for queries with quantifiers. The unexpected complexity of the later case in fact reflects the inherent complexity of quantified expressions. This complexity increases with the depth of nesting of quantified subexpressions.

The complexity of the problem addressed in this article is reflected by the fact that straightforward range modifications - e.g., those used for quantifier-free queries - in general compromise the semantics of quantified queries. In order to define equivalence preserving transformations, it is necessary to consider the possibilities to satisfy the quantified queries without instantiating all its variables. To this aim, we introduce the new notion of *co-range*. Roughly, a range for a variable describes a super-set of the values taken by the variable during an evaluation of the query. By contrast, a co-range is a formula the satisfaction of which induces a solution to the query that does not bind the variable under consideration. A co-range is a concept complementary to that of a range, hence the name we give to this notion.

Since databases are intended to store huge amount of data, their domains are usually large. It is therefore rather natural to consider queries that can be evaluated without searching the whole database domain. The equivalent notions 'definiteness' [KUH 67] and 'domain independence' [FAG 80] were proposed as a formalization of such queries. These two notions characterize formulas whose valuations remain unchanged under updates on relations that do not occur in the formulas. Such formulas can in principle be answered without imposing to search the whole database domain. In practice, this is possible with a constructive evaluation method only if the query is constructively domain independent.

The concepts 'definiteness' and 'domain independence' may be considered to be not precise enough, for they do not refer to any evaluation principle. In contrast, the 'constructive domain independence' is a

notion based on a class of proof procedures for quantified queries, namely, the class of constructive evaluation methods. Constructive domain independence implies definiteness and general domain independence. However, the converse is false. Constructive domain independence is more restrictive than general domain independence because it relies on *evaluation* methods of a specific kind, namely, those that are constructive. Domain independence and definiteness, as opposed, refer to the *valuations* of queries without considering the ways these valuations are obtained. Constructive domain independence is a solvable property [BRY 89b] while general domain independence is unsolvable [DIP 69].

In order to remedy to the unsolvability of the class of domain independent formulas, solvable subclasses have been proposed, among others the range-restricted formulas [ND 83, NIC 81], the allowed formulas [VGT 87], and the evaluable formulas [DEM 82]. We show that the rewritings we propose reduce the queries in these classes to cdi expressions. Thus, they give rise to (constructively) evaluate range-restricted, allowed, and evaluable queries in a domain independent manner, i.e., without explicitly searching the whole database domain.

The results presented here are part of a research on integrity constraint and query processing pursued at ECRC. Other studies were devoted to connected issues. We gave logical foundations to the introduction of quantifiers and negations in database rules and queries in [BRY 89b]. In [BRY 89a], we described an evaluation method based on relational algebra for answering efficiently quantified queries. In [BDM 88], two procedures were defined. The first one improves the evaluation of integrity constraints in updated databases. The second one - a refined version of the theorem prover SATCHMO [MB 88] - is devoted to checking the consistency of constraints and deduction rules. Other studies at ECRC investigated the evaluation of queries involving recursive deduction rules [VIE 86, VIE 88], recursive deduction rules with negations [BRY 89b], methods for coupling a database system with PROLOG [BOC 86], and the applicability of the PROLOG abstract machine by Warren to knowledge bases [BOC 87].

The article is organized as follows. Section 1 is this introduction. We recall definitions and we introduce notations in Section 2. In Section 3, we define constructive evaluation methods and constructively domain independent (cdi) queries. We describe the range generation method in Section 4. The concept of co-range is introduced in Section 5. There, we define equivalence preserving rewritings of quantified queries. In Section 6, we show that these rewritings permit to reduce range-restricted, allowed, and evaluable formulas to cdi expressions. We summarize the main points of the paper in Section 7.

2. Definitions and Notations

We consider queries that are expressed in a relational calculus with domain variables. A selection over an n-ary relation R is therefore represented in a query by means of an atom $R(t_1,...,t_n)$, where the terms t_i are constants or variables. The choice of a formal calculus intends to make the description of the transformations independent from any actual query language. Domain variables are chosen instead of tuple variables for making easier the reference to logic properties that are traditionally expressed in this manner. The logic notions we rely on in this article are basical ones. Their definitions can be found, e.g., in the tutorial [MEN 79].

The *domain* of a database is defined as the set of terms occurring as attributes in its relations. A special predicate denoted 'dom' will be used to refer to the database domain. Variables in queries are assumed to range over the database domain. This hypothesis is known under the name of Domain Closure Axiom.

Two queries F_1 and F_2 are *equivalent* if they admit the same answers on all databases. By the Domain Closure Axiom, a query $F[x_1, ..., x_n]$ with free variables $x_1, ..., x_n$ is equivalent to the query $dom(x_1) \wedge ... \wedge dom(x_n) \wedge F[x_1, ..., x_n]$. Similarly, quantified queries $\exists x\ F[x]$ and $\forall x\ F[x]$ are respectively equivalent to $\exists x\ dom(x) \wedge F[x]$ and to $\forall x\ dom(x) \Rightarrow F[x]$.

In order to precise, when necessary, the order in which conjunctive expressions are evaluated, we shall use the conjunction symbol '&'. Writing a conjunction F_1 & F_2 (where the conjuncts F_1 and F_2 may share variables) means that F_1 is intended to be evaluated before F_2. If the ordering is not relevant, we shall write indiferently $F_1 \wedge F_2$ or $F_2 \wedge F_1$.

Conventional query languages usually have typed variables. We recall that typed quantifications $\exists x$ in R: F[x] and $\forall x$ in R: F[x] correspond to $\exists x\ R[x] \wedge F[x]$ and $\forall x\ R[x] \Rightarrow F[x]$, respectively, in untyped logic. (As usual, we denote by R the typing predicate associated with a type R.) In order to treat all parts of a query uniformly, we adopt the untyped formalism. For the sake of uniformity, we shall assume that the connectives $\Rightarrow$ and $\Leftrightarrow$ are always expressed in terms of $\vee$ and $\neg$, according to the classical equivalences $(F_1 \Rightarrow F_2) \Leftrightarrow (\neg F_1 \vee F_2)$ and $(F_1 \Leftrightarrow F_2) \Leftrightarrow [(\neg F_1 \vee F_2) \wedge (F_1 \vee \neg F_2)$.

Moreover, we assume that all queries are rectified, i.e., the variables have been consistently renamed such that a same variable symbol does not occur in distinct quantifications. Thus, an expression $\forall x\ p(x) \Rightarrow [\exists x\ q(x)] \wedge r(x)$ is rewriten into $\forall x\ p(x) \Rightarrow [\exists y\ q(y)] \wedge r(x)$. This assumption is implicit in most query and programming languages. Making it explicit permits to simplify formal definitions.

Given an atom A, A itself and its negation $\neg$ A are called *literals*. A literal has *positive sign* if it is an atom, *negative sign* else. A formula $\neg$ F is a *negative* formula. A formula which is not of the form $\neg$ F is a *positive* formula. A subformula G has *positive polarity* in a formula F if G is embedded in zero or in an even number of negations in F (the left hand side of an implication being considered as an implicit negation). Similarly G has *negative polarity* in F if it is embedded in an odd number of - explicit or implicit - negations in F.

3. Constructive Evaluations of Quantified Queries

In this section, we introduce the notion of *constructive evaluation* as an abstract formalization of the principles of various approaches that have been proposed for evaluating queries with quantifiers [COD 72, PAL 72, JS 82, DAY 83, CG 85, DAY 87, BRY 89a]. Then, we define *constructively domain independent* quantified queries, i.e., queries that admit domain independent constructive evaluations.

A rather intuitive approach to the evaluation of quantified expressions is described by the procedures of Fig. 1 on the next page. An existential formula $\exists x\ F[x]$ is evaluated by searching the domain Dom of the database for a constant 'c' such that F[c] holds. Universal expressions are similarly evaluated. The lack of value 'c' in the domain Dom such that F[c] does not hold implies the truth of a universal expression $\forall x$ F[x].

```
evaluate(∃ x F[x], value):
value := false
for each c in Dom while value ≠ true
do
      evaluate(F[c], v)
      if v = true then value := true
end
```

```
evaluate(∀ x F[x], value):
value := true
for each c in Dom while value ≠ false
do
      evaluate(F[c], v)
      if v = false then value := false
end
```

Fig. 1 Two basic algorithms for evaluating quantified queries

The procedures given in Fig. 1 pipeline all operations and perform one tuple at a time. For the sake of efficiency, methods performing according to other kinds of control have been proposed. The methods based on relational algebra [COD 72, PAL 72, JS 82, CG 85, BRY 89a] as well as the methods [DAY 83, DAY 87] that are based on special procedures and data structures do not instantiate the quantified variables in a one-tuple-at-a-time manner. They instantiate the quantified variables over the database domain - or over a part of it defined by the type of the variable - in a set-oriented manner. Though not conforming to the one-tuple-at-a-time control of the procedures in Fig. 1, they retain the logic of these procedures - according to Kowalski's paradigm 'Algorithm = Logic + Control' [KOW 79].

The instantiation based approach to proving quantified expressions is very natural. It is widely applied in mathematics and in formal logic. It refers to the logical concept of *constructive proof* [TRO 77], because a solution value (a counter-example, respectively) is given in order to prove an existential query (a universal query, respectively). Constructive proofs can be recursively formalized as follows [BRY 89b, TRO 77], considering as constructively proven ground atomic formulas that are axioms.

- A constructive proof of $F_1 \wedge F_2$ consists in a constructive proof of F_1 and a constructive proof of F_2.
- A constructive proof of $F_1 \vee F_2$ consists in a constructive proof of F_1, or in a constructive proof of F_2.
- A constructive proof of $F_1 \Rightarrow F_2$ consists in specifying a procedure T which transforms any constructive proof P_1 of F_1 into a constructive proof $T(P_1)$ of F_2.
- A constructive proof of $\neg F$ consists in a constructive proof of $F \Rightarrow$ *false*.
- If the variable x ranges over the domain D, a constructive proof of $\forall x\ F[x]$ is a procedure T which, on application to any pair (t, p) of a term t and a constructive proof p that $t \in D$, yields a constructive proof T(t, p) of F[t].
- If the variable x ranges over the domain D, a constructive proof of $\exists x\ F[x]$ consists in a term t, in a constructive proof p that $t \in D$, and in a constructive proof of F[t].

Non-constructive proofs of quantified queries proceed differently, e.g., relying on the excluded middle, or by showing the impossibility of the formula or of its negation without instantiating it. Query answering methods based on non-constructive proof techniques, e.g., based on semantic processings [KIN 81], have

been investigated. Answering methods that provide intentional answers - instead of extensional ones - to quantifier-free queries, e.g., [CHO 87], could be used for defining non-constructive evaluation procedures for quantified queries. For answering methods of these kinds, the rewritings we define in this paper could very well reveal inefficient. However, they permit to improve the constructive processing of quantifiers.

Relying on the concept of constructive proof given above and on the usual notion of evaluation for quantifier-free database queries, constructive evaluations of database queries with quantifiers can be formally defined as follows:

Definition 1

A constructive evaluation of '$\forall x$ F[x]' is a procedure that, for each value 'c' returned by the evaluation of 'dom(x)' performs an evaluation of 'F[c]'.

A constructive evaluation of '$\exists x$ F[x]' is an evaluation of 'dom(x) & F[x]'.

In other words, constructive evaluation method reduce the processing of quantified queries to the conventional, quantifier-free case. We recall that, according to the Domain Closure Axiom, a constant c is proven to belong to the database domain - i.e., dom(c) is proven - if a fact $p(c_1, ..., c_n)$ is true in the database and $c = c_i$ for some $i = 1, ..., n$. In [BRY 89b], we investigate more thoroughly the relationship between database query evaluation and the constructivistic approach to logic. There, we introduce the following concepts for characterizing the queries that can be constructively evaluated without explicitly searching the database domain.

Definition 2

Let F be a formula and A be an atom in F. Let S(F, A) denote the formula obtained from F by replacing A in F by *true*.

An atom A is redundant in F if F and S(F, A) are equivalent.

A formula F is said to be *constructively domain independent* (cdi) if the domain atoms occurring in F are redundant.

For example, the domain atom 'dom(c)' is redundant in 'dom(c) $\wedge$ p(c)'. The atom 'dom(c)' is redundant in a query F each evaluation of F contains a positive fact in which 'c' occurs. Indeed, if 'c' occurs in a relation, then 'c' belongs by definition to the database domain. For example, the domain facts are redundant in constructive evaluations of 'dom(x) & p(x,y)'. More generally, constructively domain independent formulas are syntactically characterized as follows:

Proposition 1 [BRY 89b]

Positive constructively domain independent (cdi) formulas are recursively characterized as follows:

1. An atom is a cdi formula.
2. The conjunction of two cdi formulas is a cdi formula.
3. The disjunction of two cdi formulas with same free variables is a cdi formula.

4. If F_1 is a cdi formula and if F_2 is any formula whose free variables are all free in F_1, then F_1 & F_2 is a cdi formula.
5. $\exists x$ F is a closed cdi formula if F is an open cdi formula.
6. If F_1 is a cdi formula with free variable x and if F_2 is any formula with no free variable other than x, then $\forall x \ \neg[F_1 \ \& \ \neg F_2]$ is a cdi formula.

A negative formula $\neg$ F is cdi if and only if F is a closed cdi formula.

Constructively domain independent formulas are domain independent. By Proposition 1, the class of constructive domain independent formulas is solvable, as opposed to the class of domain independent formulas [DIP 69]. Other solvable subclasses of domain independent formulas have been proposed: Range-restricted formulas [ND 83, NIC 81], allowed formulas [LT 86, VGT 87], and evaluable formulas [DEM 82, VGT 87]. A formula in one of these classes is not necessarily cdi. We show in Section 6 that the rewritings we propose permit to construct equivalent cdi formulas to formulas in one of these classes.

We conclude this section by showing that the methods proposed in [COD 72, PAL 72, DAY 83, JS 82, DAY 87, BRY 89a] evaluate quantifiers in a constructive manner. To this aim, we shall distinguish between two classes of methods. The methods of the first class, namely [COD 72, PAL 72, JS 82, BRY 89a], are improvements of the processing of quantifiers in relational algebra described by Codd in [COD 72]. It is therefore sufficient to show that the processing of quantifiers of this later method is constructive. This is an immediate consequence of Proposition 2. The methods of the second class [DAY 83, DAY 87] process quantified queries differently. Proposition 3 below establishes their constructive character.

Proposition 2

Let R and S be two unary relations. Consider the quantified queries:

F_1: $\exists x \ R(x) \wedge S(x)$ $\qquad$ G_1: $\forall x \ (R(x) \Rightarrow S(x))$

F_2: $\exists x \ R(x) \wedge \neg S(x)$ $\qquad$ G_2: $\forall x \ (R(x) \Rightarrow \neg S(x))$

The evaluation of F_1 (F_2, resp.) by computing $R \underset{1=1}{\bowtie} S$ ($R - S$, resp.) and checking its non-emptiness is constructive. The evaluation of G_1 (G_2, resp.) by computing $S \div R$ ($R \underset{1=1}{\bowtie} S$, resp.) and checking its emptiness is constructive.

[*Proof:* Proposition 2 follows from Definition 1, from the equivalence $[\forall x \ R(x) \Rightarrow S(x)] \Leftrightarrow [\neg\exists x \ \neg(R(x) \wedge \neg S(x))]$, and from the definitions of the join and division operators.]

The procedures described in [DAY 83, DAY 87] process in two phases. During the first phase, special data structures are created. During the second phase, these structures are searched. This search corresponds - up to the ordering of the tuple comparisons - to the algorithms given in Fig. 1. Therefore, the following result permits to conclude that the methods [DAY 83, DAY 87] are constructive.

Proposition 3

The algorithms in Fig. 1 perform constructive evaluations of quantified queries.

[*Proof:* immediate from Definition 1.]

4. Generating Ranges

If queries are arbitrary logical formulas, they are not necessarily cdi. The database domain could in theory be used for constraining the variables without explicit ranges. If variables are typed in the considered language, types can be use instead of the domain. Although this permits to evaluate all formulas, it is often extremely inefficient. Consider for example the query

Q_1: $\exists x\ [\forall y$ lecture(y) => attends(x,y)]

asking if someone is attending all lectures. All individuals and all objects referenced in the database, in particular the lectures, belong by definition to the database domain. Considering the variable x ranging over the whole domain therefore permits to instantiate it with lectures. This could lead to asking if some lectures are attending all lectures! Typing variables does not really solve the problem, but reduces slightly the set of values to consider. Intuitively, the projection of the relation 'attends' on its first attribute - defined by the formula [$\exists z$ attends(x,z)] - is an appropriate range for x. However, Q_1 is *not* logically equivalent to the following formula:

$\exists x\ [\exists z$ attends(x,z)] $\wedge$ [$\forall y$ lecture(y) => attends(x,y)]

Elaborating on this idea, we describe a technique for recognizing variable ranges that are implicit in quantified queries. Similar techniques have been investigated in [LT 86, DEC 89]. In the next section, relying on the concept of co-range, we show how variable ranges can be used for generating rewritings of queries that preserve logical equivalence.

This technique can also be applied to cdi queries in order to obtain more constraining ranges. This can be a significant optimization. Consider for example the following query

Q_2: $\exists x$ employee(x) $\wedge$ $\forall y$ [(employee(y) $\wedge$ member(y,cs)) => subordinate(y,x)]

asking if there is an employee to whom all members of the computer science department are subordinate. If the number of employees that are superordinate to others is much smaller than the number of all employees then it is beneficial to assign to the variable x the more constraining range:

$\exists z$ subordinate(z,x)

Combined with appropriate cost estimations, the range generation technique permits interesting optimizations of cdi queries. In this article, we describe the range and co-range generation techniques. However, we do not discuss how to chose convenient ranges out of the many choices generated. Looking for more constraining ranges is in general useful for variables with universal quantifications in their scope, as the variable x in previous example queries Q_1 and Q_2.

Well-formed formulas may contain 'useless' quantifications, i.e. quantifications applying to variables that do not occur in the rest of the formula. This is for example the case of the first quantification in the formula F: $\exists x\ [\forall y$ p(y) => q(y)]. There are no chances to find ranges for variables with such quantifications! We therefore introduce two rewriting rules for avoiding these cases:

Rule 1: $\exists x\ F \rightarrow F$ if x does not occur in F

Rule 2: $\forall x\ F \rightarrow F$ if x does not occur in F

Rules 1 and 2 preserve logical equivalence.

4.1. Motivating Examples

Before formally describing the range generation technique, we illustrate its principles on a few examples.

Example 1

Consider first F_1: $\exists x_1 x_2\ [p(x_1,x_2) \wedge G]$. F_1 clearly imposes on x_1 and x_2 to range over the first and second attribute of p, respectively. If G is $q(x_1,x_2)$, the most constraining range that can be associated to x_1 and x_2 is $[p(x_1,x_2) \wedge q(x_1,x_2)]$. More generally, if $R([x_1,x_2],G)$ denotes the most constraining range induced by G on x_1 and x_2, the range for x_1 and x_2 in F_1 is $p(x_1,x_2) \wedge R([x_1,x_2],G)$.

Example 2

Consider now F_2: $\exists x\ [\neg p(x) \wedge G]$. Since negative information is not stored in databases, the subformula $\neg p(x)$ cannot contribute to the definition of a range for x. In such a case, we state $R([x],\neg p(x)) = dom(x)$. We therefore have $R([x],[\neg p(x) \wedge G]) = dom(x) \wedge R([x],G)$. Since by definition all values are in the database domain, $dom(x) \wedge R([x],G)$ is equivalent to $R([x],G)$. In the general case, the polarities of atoms instead of their signs have to be taken into account.

Example 3

Consider F_3: $\exists x\ [G_1 \vee G_2]$. When F_3 is satisfied, at least one of $R([x],G_1)$ and $R([x],G_2)$ is necessarily satisfied. Therefore, $R([x],G_1 \vee G_2) = R([x],G_1) \vee R([x],G_2)$.

Example 4

Consider F_4: $\exists x \forall y\ G$, where G is a formula in which x and y are free. If x satisfies the formula $\forall y\ G$, then it satisfies the formula $\exists y\ G$ as well. This second expression is convenient as a range, according to Definition 1. If for example G: g(x,y), it is indeed sufficient to make x ranging over the first attribute of the relation g, i.e., to assign the range $\exists y\ g(x,y)$ to x. More generally, we shall define $R([x],\forall y\ G) = \exists y\ R([x],G)$.

4.2. Rewriting Rules for Generating Ranges

While looking for expressions constraining some variable x one may encounter subformulas in which x does not occur. This is for example the case with the formula:

F_5: $\exists x \forall y\ \ \neg p(y) \vee [q(y) \wedge r(x,y)]$

Note that no equivalence preserving rewritings of F_5 permit to move p(y) out of the scope of x. For expressing that the subformulas $\neg p(y)$ and q(y) do not impose conditions on x, we define $R([x],\neg p(y))$ and R([x],q(y)) as 'empty'. This auxiliary symbol is used as a place-holder during the range generation. We assume that 'empty' is not available in the user language.

In the following definition, we consider only formulas with existential quantifications. This is not a restriction, since the universal quantifications can be rewritten in terms of (negated) existential ones according to the following equivalence preserving rule:

Rule 3: $\quad \forall x\ F \quad \rightarrow \quad \neg\ \exists x\ \neg\ F$

For the sake of simplicity, we consider in the rest of the paper that universal quantifications have been

rewritten according to Rule 3. In the following proposition, we define a rewriting system. We show that applying its rules to any formula always terminates in finite time - the rewriting system is *noetherian* - and we prove that the final result of the translation does not depend on the order of application of the rules - the rewriting system is *confluent* or has the Church-Rosser property.

Proposition 4

Let x be a variable. Let Rules 4 to 12 be defined with respect to x as follows.

Rule 4:	A	→	empty	if A is an atom not containing x
Rule 5:	A	→	dom(x)	if A is an atom with negative polarity containing x
Rule 6:	$\neg\ \exists y\ G$	→	$\exists y\ \neg G$	
Rule 7:	$dom(x) \wedge G$	→	G	if x occurs in G
Rule 8:	$G \wedge dom(x)$	→	G	if x occurs in G
Rule 9:	empty θ G	→	G	where θ stands for ∧ or ∨
Rule 10:	G θ empty	→	G	where θ stands for ∧ or ∨
Rule 11:	$\neg\ \neg\ G$	→	G	
Rule 12:	$\neg$ empty	→	empty	

The rewriting system consisting of Rules 1 to 12 is noetherian and confluent.

[*Proof:* In order to prove the noetherian character of the system, it suffices to remark that the number of times a given rule might be applied during a rewriting process is bounded by a parameter that depends only on the considered rule and on the formula to translate. Rule 7 and 8 for example, are applicable at most as many times the considered formula contains domain atoms.

Before completing the proof of Proposition 4, we recall some concepts in an informal manner. Refer to [HUE 80, SCH 87] for formal definitions.

Two subformulas SF_1 and SF_2 form a 'critical pair' if there is a formula F and two distinct rewriting rules both applicable on F through the subformulas SF_1 and SF_2, respectively. A 'normal form' of a formula F is a final translation of F. Since the system consisting of rules 1 to 12 is noetherian, as shown in [HUE 80] it suffices to prove that for all critical pairs (SF_1, SF_2) and for the corresponding normal forms NF_1, NF_2 of a formula F, we have $NF_1 = NF_2$, in order to prove that Rules 1 to 12 form a confluent rewriting system. Since this system is finite, there are finitely many critical pairs: The critical pairs can be successively checked for the required property.

Consider for example, the critical pair $(\neg\exists y\ (G \wedge dom(y)), (G \wedge dom(y)))$ where y is assumed to occur in G. By Rule 6, $\neg\exists y\ (G \wedge dom(y))$ yields $\exists y\ \neg\ (G \wedge dom(y))$. Applying Rule 9 on that formula yields $\exists y\ \neg\ G$. If rule 9 is applied first on $\neg\exists y\ (G \wedge dom(y))$, the expression $\neg\exists y\ G$ is generated. The same expression $\exists y\ \neg\ G$ is obtained with Rule 6. The other critical pairs are similarly treated.]

By Proposition 4 the following definition is correct.

Definition 3

Given a formula $\exists x\ F$, the most constraining range R for the variable x induced by F is obtained from F by applying the rewriting rules 1 to 12 with respect to x.

Intuitively, Rule 4 expresses that the variables distinct from x are "eliminated by projections" in a range for x. For example, the range for x induced by the expression $\forall y\ [\neg P(x,y) \vee Q(x,y)]$ is $\exists y\ Q(x,y)$, i.e., in algebraic terms $\pi_1(Q)$. Fig. 2 below shows how this range for x is generated from the considered query.

$\forall y\ [\neg P(x,y) \vee Q(x,y)]$	$\rightarrow$	(Rule 3)
$\neg\exists y\ \neg[\neg P(x,y) \vee Q(x,y)]$	$\rightarrow$	(Rule 6)
$\exists y\ \neg[\neg([\neg P(x,y) \vee Q(x,y)])]$	$\rightarrow$	(Rule 11)
$\exists y\ [\neg P(x,y) \vee Q(x,y)]$	$\rightarrow$	(Rule 5)
$\exists y\ [dom(x) \vee Q(x,y)]$	$\rightarrow$	(Rule 4)
$\exists y\ [empty \vee Q(x,y)]$	$\rightarrow$	(Rule 9)
$\exists y\ [Q(x,y)]$		

Fig. 2 Example of Range Generation

Rules 9 to 12 remove all occurrences of 'empty', provided there were no 'useless' quantifications in the formula, i.e., Rules 1 and 2 had been applied. Some 'dom' atoms may occur in a range generated according to Definition 3, reflecting that the considered formula does not constrain the corresponding variable. This happens in particular if the considered formula is not domain independent. Proposition 6 in the next section shows how to use the ranges generated by the rewriting system of Rules 1 to 12.

5. Co-Ranges and Equivalence Preserving Rewritings

A formula '$\exists x\ F$' is not always equivalent to '$\exists x\ R \wedge F$', where R is the most constraining range for x obtained from F with the rewriting system of Proposition 3. Consider again the request Q_1 of Section 4 (in which the universal quantifier has been rewritten with a negated existential one):

Q_1: $\exists x \neg\ [\exists y\ lecture(y) \wedge \neg\ attends(x,y)]$

By Definition 3, the range for x is $\exists z\ attends(x,z)$. The formula

Q_2: $\exists x \exists z\ attends(x,z) \wedge \neg\ \exists y\ (lecture(y) \wedge \neg\ attends(x,y))$

is not equivalent to Q_1. As opposed to Q_1, Q_2 evaluates to false when there are no lectures and no registered attendees. However, if there are some lectures, Q_1 and G have the same valuations. The problem

here is that Q_1 can be satisfied in certain cases without conditions being imposed on x.

An equivalence rewriting of a formula 'Q: ∃x F' therefore requires to consider the possibilities to satisfy Q without constraining the variable x. These possibilities can be detected in a manner similar to how conditions on the variable x were recognized, but now replacing by a 'cond' symbol the subformulas containing x. We assume that, like 'empty', the symbol 'cond' is not available in the user language. Applied to Q_1, this leads to:

C_1: ¬ [∃y lecture(y) ∧ cond]

after removal of a useless quantification. This expression can be rewritten as

¬ [∃y lecture(y)] ∨ cond

The presence of a disjunction - with positive polarity - one member of which does not contain 'cond' shows that C_1, and in turn Q_1, can be satisfied without imposing any condition on x. We call ¬ [∃y lecture(y)] the co-range for x.

Definition 4

The *co-range* C for x induced by ∃x F is obtained from F by first applying the following rewriting rules:

Rule 1:	∃x F	→	F	if x does not occur in F
Rule 3:	∀x F	→	¬ ∃x ¬ F	
Rule 13:	G	→	cond	if G is an atom containing x
Rule 14:	G ∨ cond	→	G	if G ∨ cond has positive polarity in F
Rule 15:	cond ∨ G	→	G	if cond ∨ G has positive polarity in F
Rule 16:	G ∧ cond	→	G	if G ∧ cond has negative polarity in F
Rule 17:	cond ∧ G	→	G	if cond ∧ G has negative polarity in F
Rule 18:	¬ ¬ G	→	G	
Rule 19:	¬ cond	→	cond	

and finally applying the rule:

Rule 20: G → *false* if 'cond' occurs in G

The following proposition establishes the correctness of Definition 4.

Proposition 5

The rewriting system of Definition 4 (Rules 1,3 and 13 to 19) is noetherian and confluent.

[Proof: The system is noetherain because each rule can only be applied a finite number of times on a given formula. This number depends only of the considered rule and of the given formula. The rewriting system is confluent because, for each normal pair (SF_1, SF_2) and for the associated normal forms (NF_1, NF_2) of a formula F, we have NF_1 = NF_2. Consider for example the critical pair (¬ ¬ (G ∧ cond), (G ∧ Cond)). Applying first Rule 16, then Rule 18 on ¬ ¬ (G ∧ cond) yields first ¬ ¬ G, then G. The reverse order for applying the two rules yields first (G ∧ cond), then G.]

Co-ranges express the possibilities to satisfy the query without constraining the considered variables. A

co-range equal to *false* reflects the impossibility of satisfying the query without constraining the considered variables.

Proposition 6

Consider a formula F: $\exists x\ G$. Let R and C respectively denote the most constraining range and the co-range of x induced by G, respectively. Let $T(F) = C \vee (\exists x\ R \wedge G)$.

T(F) is equivalent to F.

[*Sketch of Proof:* By Definition 4 $\exists x\ R \wedge G$ implies F. If $C \neq$ *false* then C implies F. It therefore suffices to prove that F implies T(F). Definition 4 induces that $C =$ *false* if and only if all disjuncts of the prenex disjunctive normal form of F contain some x_i's. If $C \neq$ *false* then G implies R and therefore F implies T(F). If $C =$ *false* then a model satisfying F either satisfy C or satisfy $\exists x\ R$.]

Corollary

Consider a formula F. Let T(F) denote the formula constructed from F by successively performing the rewritings defined in Proposition 3 and in Definition 4.

Constructive evaluations of T(F) are domain independent if and only if 'dom' does not occur in T(F).

[*Proof:* Immediate from Definition 1.]

According to Proposition 6, the above-mentioned query

Q_1: $\exists x\ [\forall y$ lecture(y) => attends(x,y)]

is logically equivalent to

$\neg[\exists y$ lecture(y)] $\vee\ \exists x\ (\exists z$ attends(x,z)) $\wedge\ [\forall y$ lecture(y) => attends(x,y)]

Fig. 3 on the next page shows the generation of the co-range $\neg\ \exists y$ lecture(y) of the variable x. This co-range expresses that, if there is no lecture, the query Q_1 can be satisfied without binding x. Variables in more complex queries have often co-ranges that are rather complex. This is in particular the case in presence of nested quantifications.

For optimizing quantified queries, the most constraining ranges are rarely the desirable ones. Convenient ranges can be chosen on the basis of Proposition 7.

Proposition 7

Consider a formula F: $\exists x\ G$. Let R and C be the most constraining range and co-range respectively, for x induced by G. Let $\vee_{i=1}^{i=k} D_i$ be a disjunctive form of R. For each i, let R_i be a subformula of D_i such that D_i implies R_i.

F': $C \vee [\vee_{i=1}^{i=k} \exists x\ (R_i \wedge G)]$ is equivalent to F.

[*Sketch of Proof:* F' is equivalent to T(F): $C \vee (\exists x\ R \wedge G)$ since existential quantifications distribute over disjunctions and since $\exists x\ R$ implies $\exists x \vee_{i=1}^{i=k} R_i$. By Proposition 6, T(F) is equivalent to F. It follows that F' is equivalent to F.]

$\exists x\ \forall y\ (\neg$ lecture(y) $\vee$ attends(x,y))]	$\rightarrow$	(Rule 3)
$\exists x\ \neg[\exists y\ \neg\ (\neg$ lecture(y) $\vee$ attends(x,y))]	$\rightarrow$	(Rule 13)
$\exists x\ \neg\ [\exists y\ \neg\ (\neg$ lecture(y) $\vee$ cond)]	$\rightarrow$	(Rule 1)
$\neg\ \exists y\ \neg\ (\neg$ lecture(y) $\vee$ cond)	$\rightarrow$	(Rule 14)
$\neg\ \exists y\ \neg\ \neg$ lecture(y)	$\rightarrow$	(Rule 18)
$\neg\ \exists y$ lecture(y)		

Fig. 3 Example of Co-Range Generation

F' and T(F) usually contain redundant subformulas, for instance when some variables are already restricted in F. Consider for example the formula:

F_6: $\exists x\ p(x) \wedge \neg\ (\exists y\ [\neg\ q(y) \vee \neg\ r(x,y)] \wedge \neg\ s(x))$

By Proposition 7, $T(F_6)$ is:

false $\vee\ \exists x\ [p(x) \wedge s(x)] \wedge [p(x) \wedge \neg\ (\exists y\ [\neg\ q(y) \vee \neg\ r(x,y)] \wedge \neg\ s(x))]$

The second occurrence of p(x) and the subformula $\neg$ s(x) are redundant in $T(F_6)$; an expression '*false* $\vee$ G' can be replaced by G. It is simple to modify the definitions for avoiding such redundancies.

6. Evaluable Formulas Admit Equivalent CDI Forms

In this section, we consider the class of evaluable formulas [DEM 82], a solvable subclass of domain independent formulas. We show that the rewriting technique of Proposition 7 permits to generate a cdi formula from any evaluable expression. It follows that combining the rewritings of Proposition 7 with a constructive evaluation procedure permit domain independent evaluations of evaluable formulas.

We shall rely on a reformulation of the definition of evaluable formulas in the Datalog formalism - as given in [VGT 87] - instead of the original definition by means of a procedure [DEM 82]. The definition is based on two relations we first introduce.

Definition 5

The relations 'gen' and 'con' are defined by the following rules:

gen(x,P) $\leftarrow$ atom(P), free(x,P)	con(x,P) $\leftarrow$ atom(P), free(x,P)
	con(x,A) $\leftarrow$ quantified-in(x,A)
gen(x,$\neg$A) $\leftarrow$ pushnot($\neg$A,B), gen(x,B)	con(x,$\neg$A) $\leftarrow$ pushnot($\neg$A,B), con(x,B)
gen(x,$\exists$y A) $\leftarrow$ distinct(x,y), gen(x,A)	con(x,$\exists$y A) $\leftarrow$ distinct(x,y), con(x,A)

$gen(x,A \vee B) \leftarrow gen(x,A), gen(x,B)$ $\quad$ $con(x,A \vee B) \leftarrow con(x,A), con(x,B)$

$gen(x,A \wedge B) \leftarrow gen(x,A)$ $\quad$ $con(x,A \wedge B) \leftarrow gen(x,A)$

$gen(x,A \wedge B) \leftarrow gen(x,B)$ $\quad$ $con(x,A \wedge B) \leftarrow gen(x,B)$

$con(x,A \wedge B) \leftarrow con(x,A), con(x,B)$

where	atom(P)	means that	P is an atomic formula
	distinct(x,y)		the variables x and y are not identical
	free(x,F)		the variable x is free in the formula F
	quantified-in(x,F)		x is a quantified variable in F
	pushnot($\neg$A,B)		B results from $\neg$A by pushing the negation inward

Definition 6

A formula F is *evaluable* if and only if the following three conditions are satisfied:

1. gen(x,F) holds for all variable x that is free in F
2. con(x,A) holds for all subformula $\exists x$ A of F
3. con(x,$\neg$A) holds for all subformula $\forall x$ A of F

A direct consequence of the following Proposition is that any evaluable formula reduces by the rewritings of Proposition 7 to a cdi formula in which the predicate 'dom' does not occur.

Proposition 8

Let F_1: $\exists x_1$ G_1 and F_2: $\forall x_2$ G_2 be two evaluable formulas.

The atom 'dom(x_1)' ('dom(x_2)', respectively) does not occur in the most constraining range of x_1 in F_1 (of x_2 in F_2, respectively) defined according to Definition 3.

[*Sketch of Proof:* By induction on the syntactical complexity of G_1 (of G_2, respectively).]

Since range-restricted formulas and allowed formulas are evaluable [DEM 82, VGT 87], it follows from Proposition 8 that queries in these classes reduce to cdi expressions by the rewriting considered in Proposition 7.

7. Conclusion

In this article, we have investigated equivalence preserving rewritings permitting to improve the evaluation of quantified queries. These rewritings rely on the generation, from a quantified query, of the *ranges* and *co-ranges* of its variables.

We have first introduced the concept of *constructive evaluation,* in order to characterize a proof principle common to several procedures that have been proposed for processing quantified queries [COD 72, PAL 72, DAY 83, JS 82, DAY 87, BRY 89a]. Basically, an answering procedure is constructive if it process quantifications by instantiating the variables. The rewritings that have been proposed in this article permit

to improve the constructive evaluations of quantified queries. However, they can very well reveal inefficient if a non-constructive evaluation method is considered. We have introduced the class of *constructively domain independent* formulas for characterizing queries that can be constructively evaluated without explicitly searching the database domain.

Then, we have described rewritings that translate quantified queries of any kind into expressions amenable to constructive evaluations. These rewritings are based on the generation, from a query, of the most constraining ranges for its variables. When combined with a cost estimation method, they permit to select variable ranges that achieve efficient constructive evaluations. The rewritings described in this article are quite complex. This is due to the fact that straightforward range modifications would compromise the semantics of queries. In order to define equivalence preserving rewritings, we have introduced the concept of co-range. A co-range is a formula the satisfaction of which induces a solution to the query that let unbound the variable under consideration.

Finally, we have investigated the relationship between domain independent and constructively domain independent formulas. We have shown that the rewritings described in the article yield constructively domain independent expressions when applied on formulas in solvable subclasses of the class of domain independent formulas.

8. Acknowledgement

I would like to thank an anonymous referee for helpful remarks.

9. References

[BDM 88] Bry, F., Decker, H. and Manthey, R. A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases. In *Proc. EDBT '88.* March, 1988.

[BOC 86] Bocca, J. On the Evaluation Strategy of EDUCE. In *Proc. ACM Int. Conf. on the Management of Data (SIGMOD '86).* Washington, D.C., May, 1986.

[BOC 87] Bocca, J. and Bailey, P. On Prolog-DBMS connections: A Step forward from EDUCE. In *Proc. Alvey Symp. on PROLOG and Databases.* Coventry, UK, 1987.

[BRY 89a] Bry, F. Towards an Efficient Evaluation of General Queries: Quantifier and Disjunction Processing Revisisited. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD '89).* Portland, Oregon, May 31-June 2, 1989. ECRC Report TR-KB-27, Apr. 1988.

[BRY 89b] Bry, F. Logic Programming as Constructivism: A Formalization and its Application to Databases. In *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Databases Systems (PODS '89).* Philadelphia, Pennsylvania, March 29-31, 1989. ECRC Report IR-KB-54, Sept. 1988.

[CG 85] Ceri, S. and Gottlob, G. Translating SQL in Relational Algebra: Optimization, Semantics and Equivalence of SQL Queries. *IEEE Trans.* SE-11(4), April, 1985.

[CHO 87] Cholvy, L. *Interrogation d'une Base de Règle.* Technical Report 2/3274/DERI, ONERA-CERT, Toulouse, France, Feb., 1987. in French.

[COD 72] Codd, E. *Database Systems - Courant Computer Science Symp.* Prentice Hall, Englewood Cliffs, New Jersey, 1972, Chapter Relational Completeness of Database Sublanguages.

[DAY 83] Dayal, U. Processing Queries with Quantifiers: A Horticultural Approach. In *Proc. ACM SIGMOD-SIGMACT Symp. Principles of Database SYstems (PODS '83)*, pages 125-136. Atlanta, March, 1983.

[DAY 87] Dayal, U. Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers. In *Proc. VLDB '87*, pages 197-208. August, 1987.

[DEC 89] Decker, H. The Range Form of Database Queries, or: How to Avoid Floundering. In *Proc. Österreichische Artificial Intelligence Tagung*. Igls bei Innbruck, March 28-29, 1989.

[DEM 82] Demolombe, R. *Syntactical Characterization of a Subset of Domain Independent Formulas.* Technical Report, ONERA-CERT, Toulouse, France, 1982.

[DIP 69] Di Paola, R.A. The Recursive Unsolvability of the Decision Problem for the Class of Definite Formulas. *Jour. of the ACM* 16(2), 1969.

[FAG 80] Fagin, R. Horn Clauses and Database Dependencies. In *12th Ann. ACM Symp. on Theory of Computing*, pages 123-134. 1980.

[HUE 80] Huet, G. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Jour. of the ACM* 27(4):797-821, October, 1980.

[JS 82] Jarke, M. and Schmidt, J. Query Processing Strategies in the PASCAL/R Relational Database Management System. In *Proc. ACM Int. Conf. on Management of Data (SIGMOD '82)*. June, 1982.

[KIN 81] King, J.J. *Query Optimization by Semantic Reasoning*. Technical Report STAN-CS-81-857, Stanford Univ., Dpt. of Computer Sc., May, 1981.

[KOW 79] Kowalski, R.A. Algorithm = Logic + Control. *Commun. ACM* , Aug., 1979.

[KUH 67] Kuhns, J.L. *Answering Question by Computer: A Logical Study.* Technical Report RM-5428-PR, Rand Corp., 1967.

[LT 86] Lloyd, J.W. and Topor, R.W. A Basis for Deductive Database Systems II. *Jour. of Logic Programming* 3(1):55-67, 1986.

[MB 88] Manthey, R. and Bry, F. SATCHMO: A Theorem Prover Implemented in Prolog. In *Proc. Conf. on Automated Deduction (CADE '88)*. May, 1988.

[MEN 79] Mendelson, E. *Introduction to Mathematical Logic*. Van Nostrand, New York, 1979.

[ND 83] Nicolas, J.-M. and Demolombe, R. *On the Stability of Relational Queries.* Technical Report, ONERA-CERT, Jan., 1983.

[NIC 81] Nicolas, J.-M. Logic for Improving Integrity Checking in Relational Databases. *Acta Informatica* 18(3):227-253, Dec., 1981.

[PAL 72] Palermo, F. A Data Base Search Problem. In *Proc. 4th Symp. on Computer and Information Sc.*. 1972.

[SCH 87] Schmitt, P. H. A Survey of Rewrite Systems. In *Proc. Workshop on Computer Science Logic (CSL '87)*, pages 235-262. Springer-Verlag (LNCS 329), Oct., 1987.

[TRO 77] Troelstra, A.S. *Handbook of Mathematical Logic*. North-Holland, Amsterdam and New York, 1977, pages 973-1052, Chapter Aspects of Constructive Mathematics.

[VGT 87] Van Gelder, A. and Topor, R.W. Safety and Correct Translation of Relational Calculus Formulas. In *Proc. 6th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS '87)*, pages 313-327. 1987.

[VIE 86] Vieille, L. Recursive Axioms in Deductive Databases: The Query-Subquery Approach. In *Proc. 1st Int. Conf. on Expert Database Systems*. Charleston, South Carolina, 1986.

[VIE 88] Vieille, L. From QSQ towards QoSaQ: Global optimization in Recursive Queries. In *Proc. 2nd Int. Conf. on Expert Database Systems*. Tyson Corner, Virginia, 1988.