Towards an Algebraic Theory of Recursion

YANNIS E. IOANNIDIS

University of Wisconsin, Madison, Wisconsin

AND

EUGENE WONG

University of California, Berkeley, California

Abstract. An algebraic framework for the study of recursion has been developed. For immediate linear recursion, a Horn clause is represented by a relational algebra operator. It is shown that the set of all such operators forms a closed semiring. In this formalism, query answering corresponds to solving a linear equation. For the first time, the query answer is able to be expressed in an explicit algebraic form within an algebraic structure. The manipulative power thus afforded has several implications on the implementation of recursive query processing algorithms. Several possible decompositions of a given operator are presented that improve the performance of the algorithms, as well as several transformations that give the ability to take into account any selections or projections that are present in a given query. In addition, it is shown that mutual linear recursion can also be studied within a closed semiring, by using relation vectors and operator matrices. Regarding nonlinear recursion, it is first shown that Horn clauses always give rise to multilinear recursion, which can always be reduced to bilinear recursion. Bilinear recursion is then shown to form a nonassociative closed semiring. Finally, several sufficient and necessary-and-sufficient conditions for bilinear recursion to be equivalent to a linear one of a specific form are given. One of the sufficient conditions is derived by embedding bilinear recursion in an algebra.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*nonprocedural languages*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*algebraic approaches to semantics*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*logic programming*; H.2.3 [**Database Management**]: Languages—*query languages*; I.1.3 [**Algebraic Manipulation**]: Languages and Systems—*nonprocedural languages; special-purpose algebraic systems*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*logic programming*

Some results of this work are contained in the paper, "An algebraic approach to recursive inference," which appears in the *Proceedings of the 1st International Conference on Expert Database Systems* (Charleston, S.C., Apr. 1–4, 1986). Benjamin/Cummings, Redwood City, Calif., 1987, pp. 295–309, and in the paper, "Transforming nonlinear recursion into linear recursion," which appears in the *Proceedings of the 2nd International Conference on Expert Database Systems* (Tysons Corner, Va., Apr. 25–27, 1988), Benjamin/Cummings, Redwood City, Calif., 1989, pp. 401–421.

Y.E. Ioannidis was partially supported by the National Science Foundation under Grant IRI 87-03592.

Authors' addresses: Y. E. Ioannidis, Computer Sciences Department, University of Wisconsin, Madison, WI 53706; E. Wong, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

General Terms: Languages, Theory

Additional Key Words and Phrases: Closed semirings, deductive databases, query optimization, recursion

1. *Introduction*

Thus far, with few exceptions [13, 14], recursion in the database context has been studied under the formalism of relational calculus, which is a subset of first-order logic, rather than relational algebra. A possible explanation is that in conventional database systems, where no recursion is allowed, relational algebra has proved to be neither a good query language nor a useful representation for optimizing database commands (although some optimization techniques are expressed in the algebra more naturally). Recent results, however, indicate that, with recursion, there are several cases where relational algebra offers advantages over relational calculus. In particular, algebraic techniques have been used to devise new efficient algorithms for recursive query processing [27], to study several properties of recursive programs that allow transformations of such programs into more efficient ones [29, 30], and to develop a firm algebraic framework for query optimization by simulated annealing [31]. Most of the aforementioned results are hard and unnatural to obtain in a nonalgebraic setting, if possible at all. Of course, there are other results that can be obtained more naturally in a logic-based setting. The two approaches are equivalent in terms of expressive power, but each one is a better tool for the study of different aspects of logic programs. The effectiveness of the algebraic approach is based on the ability that it offers to express the query answer itself in an explicit form within an algebraic structure, which is absent in the logic-based approach. Algebraic manipulation of the query answer is thus affordable, offering useful insights into specialized properties and efficient processing strategies of recursive queries.

In this paper, we develop an algebraic theory for the study of recursion in Horn clause programs. In order to put the results of this algebraic theory in perspective, we first provide an abstraction of the query optimization process in a database system. Given a recursive Horn clause program (or any program for that matter) and a query on one of the relations defined by it, several execution plans exist that can be employed to answer the query. In principle, all the alternatives need to be considered, so that in conjunction with statistical information about the database, the one with the best performance is chosen. An abstraction of the process of generating and testing these alternatives is shown in Figure 1. This process can be seen as having three stages: *rewriting*, *ordering*, and *planning*. For each alternative that is sent into some stage, multiple alternatives are produced in that stage and sent to the next (lower) one. Each stage can be seen as operating at a different level of representation of the original program–query pair. Proceeding from the higher levels to the lower ones, the representation becomes less abstract and more detailed. In real systems, the stages do not have so clear-cut boundaries as in Figure 1, and even if they do, the process of generating all the alternatives may involve significant interaction between them. Nevertheless, for our purposes, Figure 1 is an appropriate abstraction. The three stages are analyzed below:

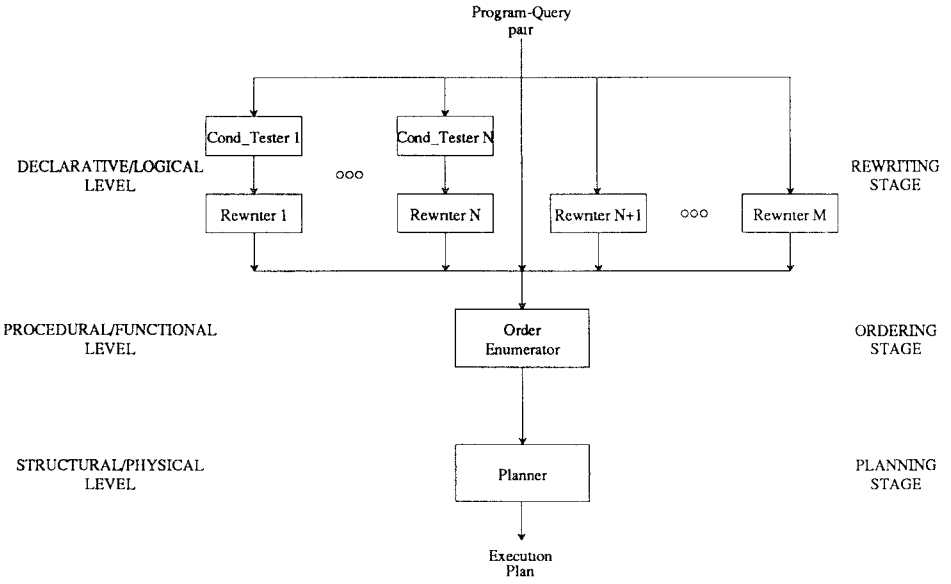

FIG. 1 Query optimizer architecture.

Rewriting. This stage produces other program–query pairs that give the same answer as the original query on the original program. Some of the transformations that produce the alternative program–query pairs are applicable only if the original one has certain properties (modules 1 to N in Figure 1), whereas others are always applicable (modules $N + 1$ to M in Figure 1). For a transformation of the former type, the original program is first passed through a decision module ("Cond_Tester" in Figure 1) that tests for the appropriate properties, and if it qualifies, it is then passed through a rewriting module. For a transformation of the latter type, the original program is simply passed through a rewriting module. In both cases, the rewritten program is sent to the next stage. If the transformation is known to always be beneficial, the original program–query pair is discarded; otherwise, it is sent to the next stage as well. Needless to say that some of the rewritten programs may qualify for further rewriting based on other properties, so this process may repeat itself multiple times. This stage works at the *declarative* level. Horn clause programs are transformed into other ones, and the transformations depend only on declarative, that is, static, characteristics of the programs. The objects manipulated at this stage are formulas of logic, so it can also be characterized as working at the *logical* level.

Ordering. This stage produces orders of execution of actions for each program–query pair produced in the previous stage. All such series of actions produce the same query answer, but their performance may very well be different. If an ordering is known to always be suboptimal, it is discarded; otherwise, it is sent to the next stage. This stage works at the *procedural* level. It takes into account procedural characteristics of the programs, and produces algorithms for answering the query. The objects manipulated at this stage are

functions accepting data as input and producing data as output, so it can also be characterized as working at the *functional* level.

Planning. This stage produces detailed execution plans for each ordered series of actions produced in the previous stage. Each execution plan specifies what indices are used, what supporting data structures are built on the fly, if/when duplicates are eliminated, and other implementation characteristics of this sort. This stage works at the *structural level.* It specifies the implementation of processing strategies at the level of data structures, and produces complete access plans. The objects manipulated at this stage are physical entities, so it can also be characterized as working at the *physical* level.

The algebraic theory developed in this paper is used to study several properties of Horn clause program–query pairs that lead to the realization of many alternative, often more efficient, query execution plans. Some of these algebraic properties are useful in identifying alternatives at the rewriting stage and some at the ordering stage (Figure 1). It should be noted that, in this paper, we put the foundations of the algebra and only present the alternative execution plans that these algebraic properties imply. We do not discuss any decision algorithms for these properties, that is, any algorithms for the "Cond_Tester" modules of Figure 1. Such algorithms for some of the discussed properties appear elsewhere [29]. Also, we do not discuss in any detail the implications of such properties on performance and whether it is always beneficial to alter a program based on them. Partial results in that direction are also found elsewhere [27]. Further investigation of these problems is part of our current and future research.

This paper is organized as follows: Section 2 gives several definitions of algebraic systems that are encountered later in the paper. In Section 3, we define the set of relational algebra operators that we consider in the paper and show that it forms a closed semiring. Section 4 formulates immediate linear recursion as an algebraic problem and shows how solving an equation provides a query answer expressed in an explicit form. In Section 5, mutual linear recursion is formulated as an algebraic problem by embedding linear systems of Horn clauses into the closed semiring of linear operator matrices. Section 6 provides several examples of cases where algebraic manipulation of the query answer at the rewriting stage gives computationally advantageous results. Section 7 does the same at the ordering stage. In Section 8, multilinear recursion is studied within the nonassociative closed semiring of relation vectors. Section 9 provides several conditions for bilinear recursion to be equivalent to a linear one of a specific form, most of which require testing for equivalence or containment of recursive programs. Section 10 embeds bilinear recursion into a nonassociative algebra and describes the derivation of one more sufficient condition for linearizability that only requires testing for equivalence of nonrecursive programs. In Section 11, we compare the algebraic approach with the logic-based approach and discuss the merits and limitations of the former. Finally, in Section 12, we summarize our results.

2. *Definitions of Algebraic Systems*

Before investigating recursion from an algebraic viewpoint, we need some definitions from algebra. In the following, any algebraic system with set S is represented by E_S. Also, for a system S on which a partial order $\leq$ is

defined, limits of sequences are defined as follows: If T is a subset of S, then b is the *least upper bound* of T (denoted as sup T) if for all $x \in T$, $x \leq b$, and for any other c satisfying $x \leq c$ for all $x \in T$, the inequality $b \leq c$ must hold. The *greatest lower bound* of T (denoted by inf T), if it exists, is defined similarly. For a sequence $\{x_k\}$, $x_k \in S$, we define its *limit superior* as $\overline{lim}\ x_k = \inf_n \sup_{n \leq k} x_k$. Similarly, we define its *limit inferior* as $\underline{lim}\ x_k = \sup_n \inf_{n \leq k} x_k$. The sequence $\{x_k\}$ *converges* if and only if $\overline{lim}\ x_k = \underline{lim}\ x_k$. In that case, the *limit* of $\{x_k\}$ is $l = \overline{lim}\ x_k = \underline{lim}\ x_k$. This definition is extended to convergence of a series. We say that a series $\{x_i\}$ converges if the sequence $\{y_k = \Sigma_{i=1}^{k} x_i\}$ converges. In this case, the limit of the series is denoted by $\Sigma_{i=1}^{\infty} x_i$.

Definition 2.1. A *group* is a system $E_S = (S, +, 0)$, where S is a nonempty set and $+$ is a binary operator on S, such that for all $a, b, c \in S$ the following hold:

(1) S is closed under $+$, that is, $a + b \in S$.
(2) The operator $+$ is associative, that is, $(a + b) + c = a + (b + c)$.
(3) 0 is an *identity element* with respect to $+$, that is, $a + 0 = 0 + a = a$.
(4) For all elements in S, there is an *inverse element* in S, that is, there exists b such that $a + b = 0$.

If (4) fails to hold (there is no inverse element), then E_S is a *monoid*. If (3) and (4) fail to hold, then $E_S = (S, +)$ is a *semigroup*. If the operator $+$ is also commutative, that is, $a + b = b + a$ for all $a, b \in S$, then the above structures are called *abelain groups, monoids*, and *semigroups* respectively.

Definition 2.2. A *field* is a system $E_S = (S, +, *, 0, 1)$, where S is a nonempty set, and $+$ and $*$ are binary operators on S, such that the following hold:

(1) $(S, +, 0)$ is an abelian group.
(2) $(S, *, 1)$ is an abelian monoid, and $(S\text{-}\{0\}, *, 1)$ is an abelian group.

In addition, for all $a, b, c \in S$ the following holds:

(3) The operation $*$ distributes over $+$, that is, $a*(b + c) = a*b + a*c$ and $(b + c)*a = b*a + c*a$.

Example 2.1. The system $E_{\mathbb{R}} = (\mathbb{R}, +, *, 0, 1)$, where $\mathbb{R}$ is the set of real numbers, and $+$ and $*$ the traditional addition and multiplication is a field.

Definition 2.3. A *vector space* over a field F, is a system $E_V = (V, +, 0, F)$, where V is a nonempty set and $+$ is a binary operator on V, such that the following holds:

(1) $(V, +, 0)$ is a abelian group.

In addition, for all $v \in V$ and $\alpha \in F$, an element αv is defined in V, such that for all $v, w \in V$ and $\alpha, \beta \in F$, the following hold:

(2) $\alpha(v + w) = \alpha v + \alpha w$.
(3) $(\alpha + \beta)v = \alpha v + \beta v$.
(4) $\alpha(\beta v) = (\alpha\beta)v$.
(5) $1v = v$.

In the last condition, 1 represents the identity element of F with respect to multiplication.

Definition 2.4. An *algebra* over a field F is a system $E_V = (V, +, *, 0, F)$, where V is a nonempty set and $+$ and $*$ are binary operators on V, such that the following hold:

(1) $(V, +, 0, F)$ is a vector space.
(2) $(V, *)$ is a semigroup.

In addition, for all $u, v, w \in V$ and $\alpha \in F$, the following hold:

(3) The operation $*$ distributes over $+$, that is, $u*(v + w) = u*v + u*w$ and $(v + w)*u = v*u + w*u$.
(4) $\alpha(v*w) = (\alpha v)*w = v*(\alpha w)$.

If associativity of $*$ fails to hold, then E_S is a *nonassociative algebra*.

Definition 2.5. A *closed semiring* is a system $E_S = (S, +, *, 0, 1)$, where S is a nonempty set on which a partial order $\leq$ is defined, and $+$ and $*$ are binary operators on S, such that for all $a, b, c \in S$, the following hold:

(1) $(S, +, 0)$ is a abelian monoid and the operation $+$ is idempotent, that is, $a + a = a$.
(2) $(S, *, 1)$ is a monoid and 0 is an *annihilator*, that is, then $a*0 = 0*a = 0$.
(3) The operation $*$ distributes over $+$, that is, $a*(b + c) = a*b + a*c$ and $(b + c)*a = b*a + c*a$.
(4) If $a_i \in S$, $i \geq 1$, is a countably infinite set, the limit $\sum_{i=1}^{\infty} a_i$ of the series $\sum_{i=1}^{k} a_i$ exists, it is unique, and it is an element of S. Moreover, associativity, commutativity, and idempotence apply to countably infinite sums.
(5) The operation $*$ distributes over countably infinite sums.

If associativity of $*$ fails to hold, then E_S is a *nonassociative closed semiring*. If 1 does not exist, then $(S, +, *, 0)$ is a *closed semiring without identity*. Finally, if an additive inverse exists for all elements of S, that is, if $(S, +, 0)$ is an abelian group, then E_S is a *closed ring*.

Example 2.2. The following system is a closed semiring: ({*false, true*}, OR, AND, *false, true*) [2]. □

Inductively, the powers of an element a of a closed semiring may be defined as:

$$a^0 = 1, \quad a^n = a^{n-1}*a = a*a^{n-1}, \qquad \text{for all} \quad n \geq 0.$$

Likewise, the *transitive closure* of a, denoted by a^*, is defined as

$$a^* = \sum_{i=0}^{\infty} a^i.$$

Note that property (4) of closed semirings guarantees the existence of a^* in S. Also note the similarity between the definition of a closed semiring and a *path algebra* [12, 42]. The only difference is that a path algebra does not necessarily satisfy properties (4) and (5).

Definitions 2.1 to 2.4 can be found in any standard text on algebra [26]. Closed semirings have been defined elsewhere as well [2, 20]. Our definition is

the one used by Aho et al. [2], but it is slightly different by being more precise in the definition of the limit of a series. Finally, some researchers have given a less general definition of closed semirings, which requires the existence of a separate transitive closure operator instead of the existence of the limit of all countable series [5, 34]. Although this less-general definition is adequate for our work, we nevertheless decided to adopt the more general one.

3. *Closed Semiring of Linear Relational Operators*

Consider a fixed, possibly infinite set C. A database D is a vector $D = (C_D \cup \{ERROR\}, \mathbf{R}_1, \ldots, \mathbf{R}_n)$[1], where $C_D \subseteq C$ is a (possibly infinite) set, $ERROR \notin C_D$, and for each $1 \leq i \leq n$, $\mathbf{R}_i \subseteq C_D^{a_i}$ is a *relation* of *arity* a_i. The implications of allowing infinite relations in D will be discussed later. Each element of $\mathbf{R}_i$ is called a *tuple*. Without loss of generality, we assume that the constants in the database are typeless, and so a *relation scheme* is defined as a relation name together with a relation arity.

Relational algebra was introduced by Codd to formally describe the operations performed on relations in a database system [17]. This paper focuses on a subset of the operators originally proposed. We are interested in the set $S = \{X, \sigma_q, \pi_p\}$ of relational operators, where each operator is defined as follows:

X: Cross product of relations.

σ_q: Selection of tuples in a relation satisfying some constraint q of the form "column1 *op* column2" or "column *op* c," $c \in C_D$, with $op \in \{=, \geq, >, \leq, <\}$.

π_p: Projection of a relation on a subset of its columns in some order specified by p.

Several other interesting relational operators can be expressed using the ones in S. *Natural join*, denoted by $\bowtie$, is equal to a cross product followed by an equality selection and elimination of the joined columns of one of the relations by projection. For relations of the same arity, *intersection* is equal to a cross product also, followed by a series of equality selections that compare corresponding columns of the two relations and cover all the columns of both relations, followed by a projection of the columns of one of them. There are only two relational operators from the original proposal that are not incorporated in this study, namely, division and set-difference. The significance of the exclusion of the latter from S will become clear shortly.

Consider a database $D = (C_D, \mathbf{R}_1, \ldots, \mathbf{R}_n)$ and the set S of primitive relational operators for D. Each element of S can be seen as a *unary* operator applied on some relation. This is obvious for selection and projection. For cross product, one of the operand relations is designated as a parameter of the operator (i.e., it is considered as part of the operator), so that the operator is applied on the other relation alone. In this sense, an operator $A \in S$ is a mapping $A: 2^{C_D^a} \rightarrow 2^{C_D^b}$. The set $2^{C_D^a}$ is the *domain* and the set $2^{C_D^b}$ is the *range* of A, that is, A takes relations of arity a as input and produces relations of arity b as output. Operators with the same domain are called *domain-compatible* and operators with the same range are called *range-compatible*.

[1]All relations appear in bold.

Likewise, if the domain of the operator A is the same as the range of the operator B, then A is called *dr-compatible* to B (dr for domain-range). Replacing every parameter relation in an operator by the corresponding relation scheme produces an *operator scheme*. Clearly, the mapping from operators to operator schemes is many-to-one. In the absence of a cross product, however, an operator coincides with the corresponding operator scheme.

Consider $S = \{X, \sigma_q, \pi_p\}$, the set of primitive relational operators for database D. The operator X in S is used to represent all possible cross products, that is, having as a parameter any possible relation in D. Likewise, the operators σ_q, π_p in S are used to represent all possible selections and projections, that is, having as subscripts all possible q's and p's, respectively. In addition, because of the existence of operators in S that are not all appropriately compatible, we introduce a new operator $\omega: \bigcup_{i=1}^{\infty} 2^{C_D^i} \rightarrow 2^{\{ERROR\}}$. The operator ω can be thought of as the error operator. Applied on any nonempty relation, it returns the relation $\{ERROR\}$. Applied on $\varnothing$, it returns $\varnothing$, that is, $\omega\varnothing = \varnothing$. Note that ω is domain compatible with all other operators, but it is range compatible with no other operator. With S as the basis set together with ω, the algebraic system $E_R = (R, +, *, 0, 1)$ is defined as follows:

R The set of elements is defined as follows:

—If $A \in S \cup \{0, 1, \omega\}$, then $A \in R$.
—If $A, B \in R$, then $(A + B) \in R$.
—If $A, B \in R$, then $(A * B) \in R$.
—R is minimal with respect to these conditions.

$+$ For A, B domain- and range-compatible operators in R, addition is defined as follows: for all $\mathbf{P}$ in the domain of A and B, $(A + B)\mathbf{P} = A\mathbf{P} \cup B\mathbf{P}$. Otherwise, $(A + B) = \omega$.

$*$ For A, B operators in R with A dr-compatible to B, multiplication is defined as follows: for all $\mathbf{P}$ in the domain of B, $(A*B)\mathbf{P} = A(B\mathbf{P})$. Otherwise, $(A * B) = \omega$.

0 The operator $0: \bigcup_{i=1}^{\infty} 2^{C_D^i} \cup \{\{ERROR\}\} \rightarrow \{\varnothing\}$ can be applied on any relation and always returns the empty relation: $0\mathbf{P} = \varnothing$.

1 The operator $1: \bigcup_{i=1}^{\infty} 2^{C_D^i} \cup \{\{ERROR\}\} \rightarrow \bigcup_{i=1}^{\infty} 2^{C_D^i} \cup \{\{ERROR\}\}$ can be applied on any relation and leaves the relation unchanged: $1\mathbf{P} = \mathbf{P}$.

Note that $\omega 1 = 1\omega = \omega$ and $\omega 0 = 0\omega = 0$. For notational convenience the multiplication symbol $*$ is omitted. Whenever $(AB)\mathbf{P}$ is used, with $A\ B \in R$ and $\mathbf{P}$ a relation, it actually represents $(A * B)\mathbf{P}$. Since $+$ and $*$ are associative, we often omit the parentheses around them. In that case, we assume right associativity for them. Equality of relational operators (even outside of R) is naturally defined through set equality as

$$A = B \Leftrightarrow \quad \text{for all } \mathbf{P}, \quad A\mathbf{P} = B\mathbf{P}.$$

Moreover, since $+$ is associative, idempotent, and commutative, system E_R may be enriched in structure by a partial order defined on R using set inclusion:

$$A \leq B \Leftrightarrow \quad \text{for all } \mathbf{P}, \quad A\mathbf{P} \subseteq B\mathbf{P}.$$

Evidently, with respect to this ordering, 0 is the greatest lower bound of R. To the contrary there is no least upper bound of R. For the purpose of having a well-defined notion of limit, we define a new operator $u: \bigcup_{i=1}^{\infty} 2^{C_D^i} \cup \{\{ERROR\}\} \rightarrow \{\bigcup_{i=1}^{\infty} C_D^i \cup \{ERROR\}\}$, which satisfies $A \leq u$, for all $A \in R$. The operator u can be thought of as the universal operator. Applied on any relation, it returns the set of all tuples of all arities on C_D, including the element $ERROR$. Note that u is not a member of R, whereas 0 is.

Before proceeding in investigating the structure of E_R, some characteristic properties of the relational operators in R are identified.

Definition 3.1. A relational operator $A \in R$ is *linear* if

(a) for all relations $\mathbf{P}$, $\mathbf{Q}$ in its domain, $A(\mathbf{P} \cup \mathbf{Q}) = A\mathbf{P} \cup A\mathbf{Q}$, and
(b) $A\varnothing = \varnothing$.

PROPOSITION 3.1. *If $A \in R$, then A is linear.*

PROOF. Consider an operator $A \in R$. The claim is proved by induction on k, which represents the number of times that addition and multiplication are applied on operators in $S \cup \{0, 1, \omega\}$ to form A.

Basis. For $k = 0$, $A \in S \cup \{0, 1, \omega\}$. It is simple to show that all these operators are linear.

Induction Step. Assume that the claim is true for all operators formed using up to $k - 1$ multiplications and additions. Let A be an operator that needs k such operations. The last operation is either addition or multiplication. Thus, A has one of the following forms:

(i) $A = B + C \Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = (B + C)(\mathbf{P} \cup \mathbf{Q})$
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = B(\mathbf{P} \cup \mathbf{Q}) \cup C(\mathbf{P} \cup \mathbf{Q})$ Definition of +
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = (B\mathbf{P} \cup B\mathbf{Q}) \cup (C\mathbf{P} \cup C\mathbf{Q})$ Induction hypothesis
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = (B\mathbf{P} \cup C\mathbf{P}) \cup (B\mathbf{Q} \cup C\mathbf{Q})$ Associativity of $\cup$
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = (B + C)\mathbf{P} \cup (B + C)\mathbf{Q}$ Definition of +
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = A\mathbf{P} \cup A\mathbf{Q}$.

$A = B + C \Rightarrow A\varnothing = (B + C)\varnothing$
$\Rightarrow A\varnothing = B\varnothing \cup C\varnothing$ Definition of +
$\Rightarrow A\varnothing = \varnothing$. Induction hypothesis

(ii) $A = BC \Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = (BC)(\mathbf{P} \cup \mathbf{Q})$
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = B(C(\mathbf{P} \cup \mathbf{Q}))$ Definition of *
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = B(C\mathbf{P} \cup C\mathbf{Q})$ Induction hypothesis
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = BC\mathbf{P} \cup BC\mathbf{Q}$ Induction hypothesis
$\Rightarrow A(\mathbf{P} \cup \mathbf{Q}) = A\mathbf{P} \cup A\mathbf{Q}$.

$A = BC \Rightarrow A\varnothing = (BC)\varnothing$
$\Rightarrow A\varnothing = B(C\varnothing)$ Definiton of *
$\Rightarrow A\varnothing = B\varnothing$ Induction hypothesis
$\Rightarrow A\varnothing = \varnothing$. Induction hypothesis

In both cases, A is proved to be linear. □

Hereafter, unless otherwise mentioned, the term "relational operator" refers to a linear relational operator.

Definition 3.2. A relational operator is *monotone* if for all relations $\mathbf{P}$, $\mathbf{Q}$ in its domain $\mathbf{P} \subseteq \mathbf{Q} \Rightarrow A\mathbf{P} \subseteq A\mathbf{Q}$.

PROPOSITION 3.2. *If a relational operator is linear, then it is monotone.*

PROOF. $\mathbf{P} \subseteq \mathbf{Q} \Rightarrow \mathbf{P} \cup \Delta\mathbf{Q} = \mathbf{Q} \Rightarrow A(\mathbf{P} \cup \Delta\mathbf{Q}) = A\mathbf{Q} \Rightarrow A\mathbf{P} \cup A\Delta\mathbf{Q} = A\mathbf{Q} \Rightarrow A\mathbf{P} \subseteq A\mathbf{Q}$. □

Proposition 3.1 and 3.2 ensure that all operators in R are both linear and monotone. To the contrary, set-difference, which has been excluded from the set of primitive relational operators S, is neither linear nor monotone.

PROPOSITION 3.3. *Let A, B, C, $D \in R$ be appropriately compatible relational operators, that is the result of any addition or multiplication is not ω. The partial order defined on R enjoys the following properties*:

(*a*) $A \leq A + B$;
(*b*) $A \leq B \Leftrightarrow A + B = B$;
(*c*) $A \leq B \Rightarrow A + C \leq B + C$;
(*d*) $A \leq B$, $C \leq D$, *and* A, B *are monotone* $\Rightarrow AC \leq BD$.

PROOF. The proofs of these properties are straightforward and are omitted. For (*d*), Propositions 3.1 and 3.2 ensure that all relational operators under consideration are monotone. □

Definition 3.3. A relational operator is *product-only* if it can be formed by applying only multiplication to elements of $S \cup \{0, 1, \omega\}$.

With the exception of 1, 0, and ω, any product-only operator $A \in R$ can be brought into the following canonical form: $A = \pi_p \sigma_{q1} \sigma_{q2} \cdots \sigma_{qk}(\mathbf{Q}X)$, that is, having no 0, 1, or ω as factors. If we denote A by $A = B_1 \cdots B_{k-1} B_k B_{k+1} \cdots B_n$, this can be achieved as follows:

—For all $1 \leq k \leq n$, if $B_k = 1$, then $A = B_1 \cdots B_{k-1} B_{k+1} \cdots B_n$.
—For all $1 \leq k \leq n$, if $B_k = 0$, then $A = 0$.
—For all $1 \leq k \leq n$, if $B_k = \omega$, and for all $1 \leq j \leq n$, $B_j \neq 0$, then $A = \omega$, otherwise $A = 0$.

Hence, with the exceptions of A being equal to 1, 0, or ω, A can be written as a product of projections, selections, and cross products. Moreover, associativity and commutativity between such operators allow the projections and selections in A to be moved to the left and cross products to be multiplied together to produce a single cross product with a larger relation, bringing A into the canonical form $A = \pi_p \sigma_{q1} \sigma_{q2} \cdots \sigma_{qk}(\mathbf{Q}X)$. (For some operators, the projection, or the selections, or the cross product may be missing.) In the sequel, we assume that all such operators (and operator schemes) are expressed in this canonical form.

Before proceeding with the theorem that algebraicly characterizes E_R, we need to prove the following two lemmas.

LEMMA 3.1. *There exists a finite set of canonical operator schemes, such that any product-only operator A is equal to an operator corresponding to a scheme in that set.*[2]

[2] These operator schemes may involve relation schemes that are not present in the original database.

PROOF. Let $A:2^{C_D^a} \to 2^{C_D^b}$ be in the canonical form $A = \pi_p \sigma_{q1} \sigma_{q2} \cdots \sigma_{qk}(\mathbf{Q}X)$. Let $\mathbf{Q} \subseteq C_D^c$. The arity c of $\mathbf{Q}$ and the number of selections k can be arbitrarily large. We show that A is equal to an operator in canonical form $A' = \pi_{p'} \sigma_{q'1} \sigma_{q'2} \cdots \sigma_{q'k'}(\mathbf{Q}'X)$, with $\mathbf{Q}' \subseteq C_D^{c'}$, such that both the arity c' of $\mathbf{Q}'$ and the number of selections k' are less than certain upper bounds (which depend on a and b only). The following series of steps in the given order construct A' from A. In every step, we show what happens to the set of selections and what happens to $\mathbf{Q}$.

(a) All selections between two columns of $\mathbf{Q}$ or a column of $\mathbf{Q}$ and a constant are applied on $\mathbf{Q}$, producing a new relation, and then are removed.

(b) For each column col of the domain of A and each operator *op*, consider the following set of selections: $\{(\text{col } op \text{ col}_t)$: col_t is a column of $\mathbf{Q}$ that does not appear in $p\} \cup \{(\text{col } op\ c_t)$: $c_t \in C_D\}$. All the selections of this set involve the same column of the domain of A, the same operator, and columns of $\mathbf{Q}$ that do not contribute to the range of A or constants. Each such set is treated differently depending on the specific *op*.

  (b1) If *op* is $=$, equality selections among the columns of $\mathbf{Q}$ $\{\text{col}_t\}$ and between these columns and the constants $\{c_t\}$ are applied on $\mathbf{Q}$, the columns of $\mathbf{Q}$ in the qualifying tuples are replaced by a single column col′ containing the value of the original columns and constants (there is only one), and the set of selections is replaced by a single selection of the form (col = col′).

  (b2) If *op* is $<$, the columns of $\mathbf{Q}$ are replaced by a single column col′ containing the minimum value of the original columns and the constants $\{c_i\}$, and the set of selections is replaced by a single selection of the form (col $<$ col′). (The same applies if *op* is $\leq$.)

  (b3) If *op* is $>$, the columns of $\mathbf{Q}$ are replaced by a single column col′ containing the maximum value of the original columns and the constants $\{c_t\}$, and the set of selections is replaced by a single selection of the form (col $>$ col′). (The same applies if *op* is $\geq$.)

(c) All columns of the original relation $\mathbf{Q}$ that do not contribute to the range of A are removed.

The rest of A remains unchanged. It is straightforward to verify that A' constructed as above is equal to A. An upper bound on the number of columns of $\mathbf{Q}'$ can be derived as follows. Because of (c), the only columns of $\mathbf{Q}$ that are kept in $\mathbf{Q}'$ are those that contribute to the range of A. In addition, columns in $\mathbf{Q}'$ are created only in (b), one for each possible column in the domain of A and each possible *op*. Thus, there are at most b columns of $\mathbf{Q}$ kept in $\mathbf{Q}'$ and there are at most $5a$ new columns created in (b) (5 operators and a columns in the domain of A). Hence, the arity c' of $\mathbf{Q}'$ is at most $5a + b$. An upper bound on the arity of $\mathbf{Q}'$ implies an upper bound on the number of selections that can be applied on the cross product of $\mathbf{Q}'$ and the operand relation, because no selection with a constant has remained in A'. With $6a + b$ total number of columns and 5 *op*s, there are at most $(6a + b)^2$ pairs of columns and at most $5(6a + b)^2$ possible selections.

We have shown that A' has a bounded arity for $\mathbf{Q}'$, a bounded number of selections, and a bounded number of projected columns (b to be exact). Hence,

we may conclude that there is a finite number of possible schemes for A'. This implies that any product-only operator can be reduced to one from a fixed, finite collection of schemes. □

In the proof of Lemma 3.1, note that if C_D is finite, there is only a finite number of relations that correspond to the scheme of $\mathbf{Q}'$, and therefore, there is only a finite number of operators that correspond to the same operator scheme. Hence, there is a finite number of operators with finite relations from a given domain to a given range.

LEMMA 3.2. *A countable sum of operators of the same scheme in canonical form is equal to a single operator of that scheme, whose parameter relation is the union of the relations of the individual operators.*

PROOF. The truth of the lemma can be seen by exchanging the roles of the domain of the operators and their parameter relations in the cross product. From $A = \Sigma_{k=0}^{\infty} A_k$, with $\mathbf{Q}_k$ the parameter relation in the cross product of A_k, this exchange produces B $(\bigcup_{k=0}^{\infty} \mathbf{Q}_k)$. The operator scheme B has the domain of the operators $\{A_k\}$ as the parameter relation scheme of its cross product. It is known that $\bigcup_{k=0}^{\infty} \mathbf{Q}_k$ is well defined, so suppose it is equal to $\mathbf{Q}$. Reversing again the roles of the domain and the parameter relation yields a single operator, which is equal to A, it has the same scheme as all the operators $\{A_k\}$, and it has $\mathbf{Q}$ as the parameter relation of its cross product. □

We now proceed to the following theorem, which characterizes the algebraic structure of the system of linear relational operators E_R.

THEOREM 3.1. *The system $E_R = (R, +, *, 0, 1)$ of linear relational operators is a closed semiring.*

PROOF. The proofs of properties 1, 2, and 3 of Definition 2.5 follow directly from the definitions of + and *. For points 4 and 5, let $\{A_i\}$ be a countably infinite set of operators in R. If some A_i, A_j are not domain-compatible or they are not range-compatible, or if some A_i is equal to ω, then $\lim_{n\to\infty} \Sigma_{i=1}^{n} A_i = \omega$. The proof is straightforward given the definition of limit. It makes use of the fact that u is the least upper bound of R and 0 is the greatest lower bound of R. For this pathological case, points 4 and 5 of Definition 2.5 clearly hold.

Assume that all the operators $\{A_i\}$ are domain- and range-compatible (without any being equal to ω) and in canonical form. By Lemma 3.1, we conclude that all those operators are equal to ones whose schemes belong to a finite set R_{os}. By Lemma 3.2, we conclude that the sum of all operators of the same scheme is equal to a single operator. Hence, $A = \lim_{n\to\infty} \Sigma_{i=1}^{n} A_i$ is equal to the sum of a finite set of operators, whose schemes belong to R_{os}. Therefore, A is well defined and it is a member of R. Since A is equal to a finite sum of operators, points 4 and 5 of Definition 2.5 are easy consequences of the definitions of + and *. □

Note that multiplication with the set-difference operator does not always distribute over addition; if d is set-difference and A, B two other operators then the equality $d(A + B) = dA + dB$ does not always hold. Including d in R would make E_R not to be a closed semiring.

Since E_R is a closed semiring, the definitions for the nth power and the transitive closure of a relational operator A that is dr-compatible to itself follow directly:

$$A^n = A*A*\cdots*A \qquad (n \text{ times}),$$

with $A^0 = 1$ and

$$A^* = \sum_{k=0}^{\infty} A^k.$$

An interesting question is whether E_R is a richer system than a closed semiring. Specificly, it would be computationally advantageous if E_R were a closed ring. Unfortunately, the answer is negative.

PROPOSITION 3.4. *The system $E_R = (R, +, *, 0, 1)$ defined above on the relational operators R is not a closed ring.*

PROOF. In order for E_R to be a closed ring, every relational operator must have an additive inverse, that is, for every $A \in R$ another operator $B \in R$ must exist such that $A + B = 0$. If suffices to find one operator in R that lacks an additive inverse. The multiplicative identity 1 serves this purpose. Assume that there exists an operator -1 such that $1 + (-1) = 0$. Then, for any nonempty relation $\mathbf{P}$,

$$(1 + (-1))\mathbf{P} = \varnothing \Rightarrow \mathbf{P} \cup (-1)\mathbf{P} = \varnothing,$$

which is a contradiction, since $\mathbf{P}$ was taken to be nonempty. □

4. *Immediate Linear Recursion*

Consider a *range-restricted linear recursive* Horn clause of the form

$$\mathbf{P}(\underline{x}^{(0)}) \wedge \mathbf{Q}_1(\underline{x}^{(1)}) \wedge \cdots \wedge \mathbf{Q}_k(\underline{x}^{(k)}) \rightarrow \mathbf{P}(\underline{x}^{(k+1)}), \tag{4.1}$$

where $\mathbf{P}$ is a *derived* relation, and for each i, $\mathbf{Q}_i$ is a relation stored in the database and $\underline{x}^{(i)}$ is a vector of variables. It is range-restricted because we require that every variable in $\underline{x}^{(k+1)}$ appears among the variables of $\underline{x}^{(i)}$, $0 \le i \le k$. It is recursive because $\mathbf{P}$ appears in both the antecedent and the consequent. It is linear because $\mathbf{P}$ appears only once in the antecedent. (The dual use of "linear" for a recursive Horn clause and for a relational operator in R will be justified in Proposition 4.1.) Note that we make no assumptions about the relations being finite. This allows a non-range-restricted Horn clause to be represented by a range-restricted one of the form in (4.1), by introducing infinite relations in the antecedent. It also allows arithmetic functions (e.g., addition) to be represented by infinite relations. (Clearly, such functions are directly evaluable and they are not explicitly stored.) In addition, constants can be represented by introducing singleton relations in the antecedent. Thus, the form of (4.1) is general, and every linear recursive Horn clause can be expressed in it.

Such a Horn clause can be expressed in relational terms as follows. Let $\mathbf{P}$, $\{\mathbf{Q}_i\}$ be relations, $\mathbf{P} \subseteq C_D^a$, and $f_{\{\mathbf{Q}_i\}}(\mathbf{P})$ be a function on $\mathbf{P}$, $f_{\{\mathbf{Q}_i\}}: 2^{C_D^a} \rightarrow 2^{C_D^a}$. (The set of relations $\{\mathbf{Q}_i\}$ is part of the function.) Then, (4.1) takes on the form

$f_{\{\mathbf{Q}_i\}}(\mathbf{P}) \subseteq \mathbf{P}$, or equivalently $\mathbf{P} \cup f_{\{\mathbf{Q}_i\}}(\mathbf{P}) = \mathbf{P}$. In addition to (4.1), consider a nonrecursive Horn clause of the form

$$\mathbf{Q}(\underline{x}) \rightarrow \mathbf{P}(\underline{x}).$$

The problem of recursive inference can be stated in relational form as follows. Given fixed relations $\mathbf{Q}, \mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and function $f_{\{\mathbf{Q}_i\}}$, find $\mathbf{P}$ such that

(i) $f_{\{\mathbf{Q}_i\}}(\mathbf{P}) \subseteq \mathbf{P}$
(ii) $\mathbf{Q} \subseteq \mathbf{P}$
(iii) $\mathbf{P}$ is minimal with respect to (i) and (ii), that is, if $\mathbf{P}'$ satisfies (a) and (b) then $\mathbf{P} \subseteq \mathbf{P}'$.

Conditions (i), (ii), and (iii) are equivalent to the following ones:

(A) $\mathbf{Q} \cup f_{\{\mathbf{Q}_i\}}(\mathbf{P}) = \mathbf{P}$
(B) $\mathbf{P}$ is minimal with respect to (A), that is, if $\mathbf{P}'$ satisfies (A) then $\mathbf{P} \subseteq \mathbf{P}'$.

Our goal is to find $\mathbf{P}$ that satisfies (A) and (B).

PROPOSITION 4.1. *Consider a function $f_{\{\mathbf{Q}_i\}}(\mathbf{P})$ on $\mathbf{P}$, where $f_{\{\mathbf{Q}_i\}}: 2^{C_D^a} \rightarrow 2^{C_D^a}$, $a \geq 1$. The function $f_{\{\mathbf{Q}_i\}}$ represents a linear recursive Horn clause of the form* (4.1) *if and only if it corresponds to a linear relational operator in R.*

PROOF. For every recursive Horn clause of the form (4.1), there is a unique underlying nonrecursive one that corresponds to it, which is a *conjunctive query* [15]. Every conjunctive query can be expressed as a composition of projections, selections, and cross products and vice versa [15]. Therefore, a function $f_{\{\mathbf{Q}_i\}}$, having $\{\mathbf{Q}_i\}$ as parameters and $\mathbf{P}$ as input, corresponds to a linear recursive Horn clause of the form (4.1) if and only if it corresponds to a linear operator in R. □

By Proposition 4.1, the established algebraic framework can be used to define the problem of recursive inference. Consider a linear recursive Horn clause that corresponds to a linear operator A, so that

$$A\mathbf{P} \subseteq \mathbf{P}.$$

Consider some constant relation $\mathbf{Q}$ that is either stored or produced by some other nonrecursive Horn clause, so that

$$\mathbf{Q} \subseteq \mathbf{P}.$$

The relation defined by the Horn clause is the minimal solution to the equation

$$\mathbf{P} = A\mathbf{P} \cup \mathbf{Q}. \tag{4.2}$$

Presumably, the solution is a function of $\mathbf{Q}$. Moreover, we are interested in solutions that can be computed by applications of selections, projections, cross products, and their sums or compositions. Thus, we restrict our attention to functions that correspond to operators in R, that is, $\mathbf{P}$ is written as $\mathbf{P} = B\mathbf{Q}$, $B \in R$. The problem becomes one of finding the operator B. Manipulation of (4.2) results in the elimination of $\mathbf{Q}$, so that the equation contains operators only. In this pure operator form, the recursion problem can be restated as

follows: Given operator A, find B satisfying:

$$\begin{aligned} &\text{(a) } 1 + AB = B \\ &\text{(b) } B \text{ is minimal with respect to (a), that is, } 1 + AC = C \Rightarrow B \leq C. \end{aligned} \tag{4.3}$$

THEOREM 4.1. *Consider eq.* (4.3*a*) *with restriction* (4.3*b*). *Its solution is* A^*.

PROOF. It has already been mentioned that $A \in R$, so it is linear and monotone. The system E_R is a closed semiring (Theorem 3.1). Thus, A^* exists and is unique for any A. First, A^* is a solution of (4.3a):

$$1 + AA^* = 1 + A(1 + A + \cdots) = 1 + A + A^2 + \cdots = A^*.$$

The second equality is due to the property that multiplication distributes over countable sums (Definition 2.5). Second, A^* is indeed the minimal solution (least fixpoint) of $1 + AB = B$. That is, for all operators B that satisfy (4.3a), $A^* \leq B$. This is shown by induction on the number of terms in $A^* = \Sigma_{k=0}^{\infty} A^k$.

Basis. For $n = 0$, $\Sigma_{k=0}^{0} A^k = 1$, and from (4.3a) and Proposition 3.3a, $1 \leq B$.

Induction Step. Assume that $\Sigma_{k=0}^{n} A^k \leq B$ for some $n \geq 0$. Then,

$$\begin{aligned} \sum_{k=0}^{n} A^k \leq B &\Rightarrow A \sum_{k=0}^{n} A^k \leq AB && \text{Proposition 3.3d} \\ &\Rightarrow 1 + A \sum_{k=0}^{n} A^k \leq 1 + AB && \text{Proposition 3.3c} \\ &\Rightarrow \sum_{k=0}^{n+1} A^k \leq 1 + AB && \text{Closed Semiring Properties} \\ &\Rightarrow \sum_{k=0}^{n+1} A^k \leq B. && \text{From (4.3a)} \end{aligned}$$

So, for all $n \geq 0$, $\Sigma_{k=0}^{n} A^k \leq B$. Since the sequence (of the partial sums) is upwards bounded by B and is monotone, its limit A^* is also bounded by B. Hence, for any B satisfying $B = 1 + \mathrm{AB}$, $\mathrm{A}^* \leq B$, and A^* is the least fixpoint of (4.3a). □

Theorem 4.1, originally due to Tarski [51], has been used in the study of the semantics of logic programs extensively [4, 54]. In the database context, it was first examined by Aho and Ullman [3]. It is the first time though that the solution of (4.3) is expressed in an explicit algebraic form within an algebraic structure like the closed semiring E_R. One can now algebraicly manipulate the query answer, which is represented by A^* possibly multiplied with other operators also, for example, selections and projections, and study its behavior. Some of the implications of the manipulative power thus afforded are discussed in Sections 6 and 7.

5. *Mutual Linear Recursion*

Until this point, we have concentrated on immediate recursion. However, the above algebraic framework can be extended so that it can be applied to the cases where mutual recursion exists as well. Without loss of generality, we assume

that all relations that are not derived recursively are stored in the database, that is, they are not produced by some nonrecursive Horn clause.

Definition 5.1. Consider a set of Horn clauses, and let $\{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n\}$ be the relations in the consequents of its elements. The set of Horn clauses is called *linear* if each Horn clause has at most one of $\{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n\}$ in its antecedent.

Example 5.1. The following system of mutually recursive Horn clauses is linear:

$$\begin{aligned} \mathbf{Q}(x, z) \wedge \mathbf{T}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{P}(y, x) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{P}(z, x) \wedge \mathbf{S}(z, z, y) &\rightarrow \mathbf{Q}(x, y), \\ \mathbf{R}(x, y) &\rightarrow \mathbf{Q}(x, y). \end{aligned}$$

To the contrary, the next one is not, because of the presence of both $\mathbf{P}$ and $\mathbf{Q}$ in the antecedent of the first Horn clause.

$$\begin{aligned} \mathbf{P}(w, z) \wedge \mathbf{Q}(x, z) \wedge \mathbf{T}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{P}(y, x) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{P}(z, x) \wedge \mathbf{S}(z, z, y) &\rightarrow \mathbf{Q}(x, y), \\ \mathbf{R}(x, y) &\rightarrow \mathbf{Q}(x, y). \end{aligned}$$

□

Note that this definition of linear is different (more restrictive) from the one given by Bancilhon and Ramakrishnan [8]. That definition includes systems that are not linear. In particular, it includes systems that can be decomposed into component linear systems. These can be solved in such an order that the relations produced by one component become parameters to the next one. We believe that a more precise term for such a system is *piecewise linear* [18] and we use the term *linear* according to Definition 5.1.

Consider a linear system of mutually recursive Horn clauses defining relations $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n$. Each Horn clause is represented algebraicly using a linear operator in R. Thus, a system of n equations is generated with n unknown variables $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n$ and is solved as an ordinary linear system. The interaction between the Horn clauses can be arbitrarily complex as long as the resulting system is linear. The possibility of immediate recursion is not excluded either. The system produced is sufficiently general to give the solution for the relations concerned.

Example 5.2. Consider the most general case for two relations P_1 and P_2, that are defined by both immediately recursive and mutually recursive Horn clauses in a linear way. With A, B, C, and D being the appropriate linear operators in R, the situation is represented by the following linear system:

$$\begin{aligned} \mathbf{P}_1 &= A\mathbf{P}_1 \cup B\mathbf{P}_2 \cup \mathbf{Q}_1, \\ \mathbf{P}_2 &= C\mathbf{P}_1 \cup D\mathbf{P}_2 \cup \mathbf{Q}_2. \end{aligned}$$

□

Define $M_n(R)$ as the set of $n \times n$ matrices, $n \geq 1$, whose entries belong to the set of linear relational operators R. Note that for any such matrix, all the operators in a column are domain-compatible and all the operators in a row are

range-compatible. With $M_n(R)$ as its set of elements, the system $E_{M_n(R)} = (M_n(R), +, *, \underline{0}, \underline{1})$ is defined as follows:

$+$ If $\underline{A} = [A_{ij}]$, $\underline{B} = [B_{ij}]$ are two matrices in $M_n(R)$, then addition is defined by $\underline{A} + \underline{B} = [A_{ij} + B_{ij}]$.

$*$ If $\underline{A} = [A_{ij}]$, $\underline{B} = [B_{ij}]$ are two matrices in $M_n(R)$, then multiplication is defined by $\underline{A}*\underline{B} = [\Sigma_{k=1}^{n} A_{ik}B_{kj}]$.

$\underline{0}$ The matrix $\underline{0}$ has all its elements equal to 0.

$\underline{1}$ The matrix $\underline{1}$ has all its elements equal to 0, except the ones on the principle diagonal, which are equal to 1.

Similarly to the situation for simple operators, the multiplication symbol $*$ is omitted.

THEOREM 5.1. *The system* $E_{M_n(R)} = (M_n(R), +, *, 0, 1)$ *is a closed semiring.*

PROOF. It is known that matrices over a closed semiring form a closed semiring [2]. Since E_R is a closed semiring, one can conclude that $E_{M_n(R)}$ is also a closed semiring. □

Powers of matrices are defined as

$$\underline{A}^m = \underline{A}*\underline{A}*\cdots*\underline{A} \qquad (m \text{ times}),$$

with $\underline{A}^0 = \underline{1}$, and the transitive closure of a matrix is defined as

$$\underline{A}^* = \sum_{k=0}^{\infty} \underline{A}^k.$$

Consider a linear system of equations like the one of Example 5.2. Using elements of $M_n(R)$, we can write it in a matrix form as

$$\underline{\mathbf{P}} = \underline{A}\underline{\mathbf{P}} \oplus \underline{\mathbf{Q}}, \tag{5.1}$$

where $\underline{\mathbf{P}}$ is the vector of unknown relations, $\underline{\mathbf{Q}}$ is the vector of stored relations, and addition $\oplus$ for relation vectors is defined as

$$\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}' = (\mathbf{P}_1 \cup \mathbf{P}_1', \mathbf{P}_2 \cup \mathbf{P}_2', \ldots, \mathbf{P}_n \cup \mathbf{P}_n').$$

Since both E_R and $E_{M_n(R)}$ are closed semirings, the minimal solution to (5.1) can be found in exactly the same way as that of (4.2) and is $\underline{\mathbf{P}} = \underline{A}^*\underline{\mathbf{Q}}$.

Example 5.3. Consider the linear system of Example 5.2. Written in matrix form, it is equal to

$$\begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix} \oplus \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{bmatrix}.$$

Solving the system yields

$$\begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^* \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \end{bmatrix}.$$

The individual solutions for $\mathbf{P}_1$ and $\mathbf{P}_2$ are:

$$\mathbf{P}_1 = (A + BD^*C)^*(BD^*\mathbf{Q}_2 \cup \mathbf{Q}_1),$$
$$\mathbf{P}_2 = (D + CA^*B)^*(CA^*\mathbf{Q}_1 \cup \mathbf{Q}_2). \quad \square$$

The importance of the algebraic formulation of the problem should be emphasized at this point. Until now, few people have dealt with mutual recursion in its full generality. The methods proposed for processing queries on relations defined by a complex recursive system of Horn clauses tend to be complicated and in some cases incomplete [25]. The power of the algebraic tools lies with the fact that the solution for arbitrarily complex linear systems can be expressed in a concise way. Algebraic manipulations of this solution generate multiple equivalent expressions, some of which may be more efficient than the straightforward implementation of the original solution. Such optimization is virtually impossible in the absence of an explicit representation of the solution, due to the complexity of the corresponding processing algorithms. The level of complexity that one faces should become more clear with the following example.

Example 5.4. Consider the case of three mutually recursive relations, with no immediate recursion for any of them. The linear system representing the situation is

$$\begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} = \begin{bmatrix} 0 & A_{12} & A_{13} \\ A_{21} & 0 & A_{23} \\ A_{31} & A_{32} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \oplus \begin{bmatrix} \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \mathbf{Q}_3 \end{bmatrix}.$$

Solving for $\mathbf{P}_1$, for example, yields the solution

$$\mathbf{P}_1 = [(A_{12} + A_{13}A_{32})(A_{23}A_{32})^*A_{21} + (A_{13} + A_{12}A_{23})(A_{32}A_{23})^*A_{31}]^*\mathbf{Q},$$

where the relation $\mathbf{Q}$ is equal to

$$\mathbf{Q} = \mathbf{Q}_1 \cup (A_{12} + A_{13}A_{32})(A_{23}A_{32})^*\mathbf{Q}_2 \cup (A_{13} + A_{12}A_{23})(A_{32}A_{23})^*\mathbf{Q}_3.$$

The expression is long. Nevertheless, it represents a complete solution of the linear system for $\mathbf{P}_1$. This was hard to express before, if at all possible. $\square$

6. *Algebraic Transformations of Linear Recursion at the Rewriting Stage*

In this section, we give several examples of cases where algebraic manipulation of an explicit representation of the query answer yields computationally advantageous results. The results presented here affect the rewriting stage of query optimization (Figure 1). The first subsection presents decompositions that are applicable to A^* for a linear operator $A \in R$ that has the form $A = B + C$. The second subsection presents decompositions that are applicable to A^* when A has the form $A = BC$. The third subsection presents replacements of A^* with transitive closures of other operators. The final subsection presents transformations of the product of A^* with other operators, primarily selection and projection.

All the optimization results presented in this section depend solely on the algebraic properties of closed semirings. Hence, the results can be generalized

to mutual linear recursion as well, by simply using linear operator matrices in place of linear operators. Moreover, unless explicitly restricted to operator schemes, all results in this section hold for both operators and operator schemes. With one exception, however, all examples involve operator schemes.

6.1. Decompositions of $(B + C)^*$

Theorem 6.1. *Let $A = B + C$. If there exist k and l such that*

$$CB \leq B^k C^l, \tag{6.1}$$

and either $k \in \{0, 1\}$ or $l \in \{0, 1\}$[3]*, then $A^* = (B + C)^* = B^*C^*$.*

Proof. Clearly, $A^* = (B + C)^* = \Sigma_{i=0}^{\infty} (B + C)^i$. This means that

$$A^* = \sum_{i_1, j_1, \ldots, i_m, j_m = 0}^{\infty} B^{i_1} C^{j_1} B^{i_2} C^{j_2} \cdots B^{i_m} C^{j_m}. \tag{6.2}$$

Consider an arbitrary term $D = B^{i_1} C^{j_1} B^{i_2} C^{j_2} \ldots B^{i_m} C^{j_m}$. Assume that $CB \leq B^k C^l$, with $k \in \{0, 1\}$ (the case of $l \in \{0, 1\}$ is handled similarly). We prove by induction on $n = i_1 + \cdots + i_m$ that $D \leq B^I C^J$, $J \geq 0$, where $I = i_1$ if $k = 0$, or $I = n = i_1 + \cdots + i_m$ if $k = 1$.

Basis. For $n = 0$, $D = C^{j_1} C^{j_2} \cdots C^{j_m} = C^{j_1 + j_2 + \cdots + j_m}$, which is already in the desired form, with $I = 0$.

Induction Step. Assume that the claim is true for some $n \geq 0$. We prove it for $n + 1$. We distinguish two cases, $k = 0$ and $k = 1$.

$k = 0$ If $j_1 + j_2 + \cdots + j_{m-1} = 0$, then D is already in the desired form. Otherwise, D can be written as $D = (B^{i_1} C^{j_1} \cdots B^{i_{m-1}}) BC^{j_m}$. By the induction hypothesis, we have that $D \leq B^{i_1} C^J BC^{j_m}$, with $J \neq 0$. Applying (6.1) with $k = 0$ yields $D \leq B^{i_1} C^{J-1} C^l C^{j_m}$, which proves our claim.

$k = 1$ Again, if $j_1 + j_2 + \cdots + j_{m-1} = 0$, then D is already in the desired form. Otherwise, D can again be written as $D = (B^{i_1} C^{j_1} \cdots B^{i_{m-1}}) BC^{j_m}$. By the induction hypothesis, we have $D \leq B^n C^J BC^{j_m}$, with $J \neq 0$. The result of applying (6.1) on the above formula J times is $D \leq B^{n+1} C^{l*J} C^{j_m}$, which again proves our claim.

Hence, every term of the sum in (6.2) is $\leq$ to a term of the form $B^I C^J$. For all I and J, the term $B^I C^J$ exists in (6.2) already. Thus, (6.2) can be modified into $A^* = \Sigma_{I=0, J=0}^{\infty} B^I C^J = B^*C^*$. □

Corollary 6.1. *If B and C commute, that is, $BC = CB$, then $(B + C)^* = B^*C^* = C^*B^*$.*

Note that Corollary 6.1 states that if B and C commute, then the separable algorithm is applicable [36]. A similar observation has been made independently by Ramakrishnan et al. [41]. Further elaboration on the relationship between commutativity and separability appears elsewhere [29].

[3] The condition of Theorem 6.1 can be easily tightened to $CB \leq (1 + B)C^*$ or $CB \leq B^*(1 + C)$. For operator schemes, the two conditions are equivalent [46], whereas for operators, the tighter condition is strictly more general. Similar comments apply to Theorems 6.2 and 6.3. We have chosen to describe the less general condition for ease of presentation.

Example 6.1. As an application of Theorem 6.1 (and Corollary 6.1), consider the following program:

$$\begin{aligned} \mathbf{P}(x, z) \wedge \mathbf{Q}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{R}(x, z) \wedge \mathbf{P}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{S}(x, y) &\rightarrow \mathbf{P}(x, y). \end{aligned}$$

The two recursive Horn clauses correspond to the backward and forward computation of the transitive closure of the binary relations **Q** and **R**, respectively. It is easy to verify that the underlying conjunctive queries commute, since composing the two in both possible ways yields one that underlies the following Horn clause:

$$\mathbf{R}(x, z) \wedge \mathbf{P}(z, z') \wedge \mathbf{Q}(z', y) \rightarrow \mathbf{P}(x, y).$$

Theorem 6.1 guarantees that the fixpoint of the first Horn clause can be computed independently of the fixpoint of the second one, as long as in the end the two are combined by an additional nonrecursive operation between them. This is captured by rewriting the original program as follows:

$$\begin{aligned} \mathbf{P}_2(x, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{R}(x, z) \wedge \mathbf{P}_2(z, y) &\rightarrow \mathbf{P}_2(x, y), \\ \mathbf{P}_1(x, y) &\rightarrow \mathbf{P}_2(x, y), \\ \mathbf{P}_1(x, z) \wedge \mathbf{Q}(z, y) &\rightarrow \mathbf{P}_1(x, y), \\ \mathbf{S}(x, y) &\rightarrow \mathbf{P}_1(x, y). \end{aligned}$$

□

THEOREM 6.2. *Let $A = B + C$. If there exist k and l such that*

$$CB \leq B^k C^l, \tag{6.3}$$

and there exists p such that either $B^p = 0$ or $C^p = 0$, then $A^* = (B + C)^* = B^* C^*$.

PROOF. If $k < 2$ or $l < 2$, Theorem 6.1 ensures the truth of the statement of this theorem as well. Assume that $k \geq 2$ and $l \geq 2$. In (6.2), consider an arbitrary term $D = B^{i_1} C^{j_1} B^{i_2} C^{j_2} \cdots B^{i_m} C^{j_m}$. Assume that there exists p such that $C^p = 0$ (the case of $B^p = 0$ is handled similarly). If $m = 1$, then D is in the desired form. If not, this means that both $j_1 \geq 1$ and $i_2 \geq 1$. If $j_1 = 1$, $i_2 = 1$, and $m = 2$, that is, if D is of the form $D = B^{i_1} CBC^{j_2}$, then applying (6.3) yields $D \leq B^{i_1} B^k C^l C^{j_2}$, which is in the desired form. Otherwise, either $j_1 \geq 2$ or $i_2 \geq 2$ or $m > 2$. In the first case, D is of the form $D = D_1 C^2 B D_2$, where D_1 and D_2 are some product-only operators. In the second case, D is of the form $D = D_1 C B^2 D_2$. In the third case, applying (6.3) yields either $D \leq D_1 C^2 B D_2$ or $D \leq D_1 C B^2 D_2$. We show that in all cases $D = 0$. Equation (6.3) yields

$$D_1 C^2 B D_2 \leq D_1 C B^k C^l D_2 \qquad \text{and}$$
$$D_1 C B^2 D_2 \leq D_1 B^k C^l B D_2 \leq D_1 B^k C^{l-1} B^k C^l D_2.$$

Replacing $D_1 B^k C^{l-2}$ by D_1 in the latter expression yields $D_1 C B^k C^l D_2$. Hence, if $j_1 \geq 2$ or $i_2 \geq 2$ or $m > 2$, we have shown that $D \leq D_1 C B^k C^l D_2$.

Our goal is to prove that $D_1 CB^k C^l D_2 = 0$. We achieve this by showing that

$$CB^k C^l \leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g CB^k C^{(g+1)l} \quad \text{for all } g. \quad (6.4)$$

The formula to the right of $\leq$ is well defined, since both $k \geq 2$ and $l \geq 2$. We prove (6.4) by induction on g.

Basis. For $g = 0$, (6.4) yields $CB^k C^l \leq CB^k C^l$, which clearly holds.

Induction Step. Assume that the claim is true for some $g \geq 0$. We prove it for $g + 1$ by repeated applications of (6.3).

$$CB^k C^l \leq \left(\left(B^k C^{l-l} \right)^{k-2} B^k C^{l-2} \right)^g CB^k C^{(g+1)l}$$

Induction hypothesis

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g B^k C^l B^{k-1} C^{(g+1)l}$$

Applying (6.3) once

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g B^k C^{l-1} B^k C^l B^{k-2} C^{(g+1)l}$$

Applying (6.3) twice

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g B^k C^{l-1} B^k C^{l-1} B^k C^l B^{k-3} C^{(g+1)l}$$

Applying (6.3) three times

$$\vdots$$

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g \left(B^k C^{l-1} \right)^{h-1} B^k C^l B^{k-h} C^{(g+1)l}$$

Applying (6.3) h times

$$\vdots$$

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g \left(B^k C^{l-1} \right)^{k-1} B^k C^l C^{(g+1)l}$$

Applying (6.3) k times

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^g \left(B^k C^{l-1} \right)^{k-2} B^k C^{l-1} B^k C^{(g+2)l}$$

$$\leq \left(\left(B^k C^{l-1} \right)^{k-2} B^k C^{l-2} \right)^{g+1} CB^k C^{(g+2)l}.$$

By taking an arbitrarily large g, we can construct arbitrarily high powers of C in $C^{(g+2)l}$. For a value of g that satisfies $(g + 2)l \geq p$, since $C^p = 0$, (6.4) yields that $CB^k C^l = 0$, which in turn implies that $D = 0$. Thus, considering all cases, either $D \leq B^I C^J$, for some I, J, or $D = 0$. Hence, we conclude that $A^* = (B + C)^* = B^* C^*$. □

Theorem 6.2 is vacuously true for operator schemes, since for an operator scheme B, $B^p = 0$ can only be true when $B = 0$. Thus, this theorem is only interesting for operators, in which case the meaning of $B^p = 0$ ($C^p = 0$) is that the data in the parameter relations of B (C) is acyclic. The intuition behind the proof is that, in order for the conditions of the theorem to hold, the data in the parameter relations of B and C must not interleave in any nontrivial ways, that is, $CB^k C^l$ with $k \geq 2$ and $l \geq 2$ is equal to 0.

Theorems 6.1 and 6.2 give sufficient conditions for $(B + C)^* = B^*C^*$. The next theorem gives sufficient conditions for $(B + C)^* = B^* + C^*$.

THEOREM 6.3. *Let $A = B + C$. If there exist k and l such that*

$$CB \leq B^k \quad \text{or} \quad CB \leq C^k \tag{6.5a}$$

and

$$BC \leq B^l \quad \text{or} \quad BC \leq C^l, \tag{6.5b}$$

then $A^* = (B + C)^* = B^* + C^*$.

PROOF. Condition (6.5a) implies that the conditions of Theorem 6.1 hold. Thus, $A^* = (B + C)^* = B^*C^* = B^* + C^* + B^*BCC^*$. Assume that in (6.5b) it is $BC \leq B^l$ that holds (the other case is treated similarly). Consider an arbitrary term of B^*BCC^*, that is, $D = B^iC^j$, $i, j \geq 1$. We distinguish two cases:

$l \geq 1$: Clearly, (6.5b) implies that
$B^iC^j \leq B^{i-1+l}C^{j-1} \leq B^{i-2+2l}C^{j-2} \leq \cdots \leq B^{i+(l-1)j}$.

$l = 0$: Similarly to the previous case, if $i \geq j$, then $B^iC^j \leq B^{i-j}$, otherwise, if $i \leq j$, then $B^iC^j \leq C^{j-i}$.

In both cases, B^iC^j is $\leq$ to a term of B^* or C^*. Thus, we may conclude that $A^* = (B + C)^* = B^* + C^*$. □

Example 6.2. This example of the applicability of Theorem 6.3 involves operators to demonstrate that the results in this section are useful not only with operator schemes. Let **S** be a binary relation corresponding to a directed graph. The following logic program computes the transitive closure of **S**:

$$\mathbf{P}(x, z) \wedge \mathbf{S}(z, y) \rightarrow \mathbf{P}(x, y),$$
$$\mathbf{S}(x, y) \rightarrow \mathbf{P}(x, y).$$

Assume that the underlying undirected graph of **S** consists of two connected components, which correspond to the relations **Q** and **R**, that is, $\mathbf{Q} \cup \mathbf{R} = \mathbf{S}$. Clearly, the above program is equivalent to the following one:

$$\mathbf{P}(x, z) \wedge \mathbf{Q}(z, y) \rightarrow \mathbf{P}(x, y),$$
$$\mathbf{P}(x, z) \wedge \mathbf{R}(z, y) \rightarrow \mathbf{P}(x, y),$$
$$\mathbf{Q}(x, y) \rightarrow \mathbf{P}(x, y),$$
$$\mathbf{R}(x, y) \rightarrow \mathbf{P}(x, y).$$

Let B and C be the operators corresponding to the underlying nonrecursive Horn clauses of the two recursive ones in the above program, for the specific instances of **Q** and **R**. It is easy to verify that $BC = CB = 0$. Thus, by Theorem 6.3, independent computation of the transitive closure of each connected component and a union of the two results is enough to compute the transitive closure of the whole graph. This can be captured by rewriting the

original program into the following one:

$$\begin{aligned}
\mathbf{P}_Q(x, y) &\rightarrow \mathbf{P}(x, y),\\
\mathbf{P}_R(x, y) &\rightarrow \mathbf{P}(x, y),\\
\mathbf{P}_Q(x, z) \wedge \mathbf{Q}(z, y) &\rightarrow \mathbf{P}_Q(x, y),\\
\mathbf{Q}(x, y) &\rightarrow \mathbf{P}_Q(x, y)\\
\mathbf{P}_R(x, z) \wedge \mathbf{R}(z, y) &\rightarrow \mathbf{P}_R(x, y),\\
\mathbf{R}(x, y) &\rightarrow \mathbf{P}_R(x, y).
\end{aligned}$$

□

The results of Theorems 6.1, 6.2, and 6.3 can be extended to sums of an arbitrary number of terms in straightforward ways. Their importance lies with the fact that computing B^* and C^* separately, and then either multiplying them or adding them, has the potential of being significantly cheaper than computing $(B + C)^*$. The main reason for this is that the latter computation produces at least as many duplicates as the former, and often many more. The proof of this fact is given elsewhere [29].

All three Theorems 6.1, 6.2, and 6.3 provide sufficient conditions for the corresponding decompositions to hold. Clearly, the theorems hold for both A, B, C being operators and A, B, C being operator schemes. Alternatively, the theorems hold both when taking into account the database and when not. Nontrivial and practical necessary conditions for $(B + C)^* = B^*C^*$ or $(B + C)^* = B^* + C^*$ are hard to derive if B and C are operators. The following theorems give necessary conditions for decompositions of operator schemes. Moreover, the condition of Theorem 6.3 for the second type of decomposition, that is, $(B + C)^* = B^* + C^*$, is shown to be necessary and sufficient.

THEOREM 6.4. *Consider operator schemes A, B, and C, such that $A = B + C$. If $A^* = (B + C)^* = B^*C^*$, then there exist k and l such that $CB \leq B^kC^l$.*

PROOF. Assume that $A^* = (B + C)^* = B^*C^*$. This implies that $(B + C)^* \leq B^*C^*$. Since we are dealing with operator schemes, we can use the theorem of Sagiv and Yannakakis that every term of the sum $(B + C)^*$ must be $\leq$ to a term of B^*C^* [46].[4] The operator CB is a term in $(B + C)^*$. This yields that there exist k and l such that $CB \leq B^kC^l$. □

THEOREM 6.5. *Consider operator schemes A, B, and C, such that $A = B + C$. Then, $A^* = (B + C)^* = B^* + C^*$ if an only if there exist k and l such that*

$$CB \leq B^k \quad \text{or} \quad CB \leq C^k,$$

and

$$BC \leq B^l \quad \text{or} \quad BC \leq C^l.$$

PROOF. The sufficiency of the condition for the decomposition is implied by Theorem 6.3. For the other direction, assume that $A^* = (B + C)^* = B^* + C^*$. Again, we may apply the theorem of Sagiv and Yannakakis for the terms

[4]Actually, the result of Sagiv and Yannakakis deals with finite sums only, but it can easily be generalized to countably infinite sums. In the sequel, we refer to their result in the more general form.

CB and BC of $(B + C)^*$ [46]. This yields that there exist k and l such that $CB \leq B^k$ or $CB \leq C^k$, and $BC \leq B^l$ or $BC \leq C^l$. □

6.2. Decompositions of $(BC)^*$

Theorem 6.6. *Let $A = BC$. If there exists k such that*

$$CB = B^k C,^5 \tag{6.6}$$

then

$$A^* = (BC)^* = \sum_{m=0}^{\infty} B^{(\sum_{i=0}^{m-1} k^i)} C^m.$$

Proof. Clearly, $A^* = (BC)^* = \Sigma_{m=0}^{\infty} (BC)^m$. Consider an arbitrary term of the sum, $D = (BC)^m$. We show by induction on m that

$$D = B^{(\Sigma_{i=0}^{m-1} k^i)} C^m.$$

Basis. For $m = 0$, $(BC)^0 = B^{(\Sigma_{i=0}^{-1} k^i)} C^0 = B^0 C^0 = 1$.

Induction Step. Assume that the claim is true for some $m \geq 0$. We prove it for $m + 1$ by an application of (6.6).

$(BC)^m = B^{(\Sigma_{i=0}^{m-1} k^i)} C^m$ — Induction hypothesis

$\Rightarrow (BC)^{m+1} = BCB^{(\Sigma_{i=0}^{m-1} k^i)} C^m$

$\Rightarrow (BC)^{m+1} = BB^{(k * \Sigma_{i=0}^{m-1} k^i)} C^{m+1}$ — Repeated applications of (6.6)

$\Rightarrow (BC)^{m+1} = B^{(\Sigma_{i=0}^{m} k^i)} C^{m+1}$.

By the above induction, the claim is true for all terms of the sum of A^*, which is therefore equal to

$$A^* = (BC)^* = \sum_{m=0}^{\infty} B^{(\sum_{i=0}^{m-1} k^i)} C^m.$$ □

Corollary 6.2. *If B and C commute, that is, $BC = CB$, then $A^* = (BC)^* = \Sigma_{m=0}^{\infty} B^m C^m$.*

Proof. Replacing $k = 1$ in Theorem 6.6 yields the desired result. □

Note that Corollary 6.2 states that if B and C commute, the powers of B and C can be computed independently and then the corresponding ones can be multiplied for the final result. The expectation is that computing the powers of B and C separately will often improve performance, because B and C are likely to be operators that involve "smaller" relations than BC, both in terms of the number of tuples and in terms of the arity. Investigation of the precise implications of such decompositions on performance is part of our future plans.

Example 6.3. The classical example of the applicability of Theorem 6.6, and more specifically of Corollary 6.2, is the "same generation" program:

$$\mathbf{up}(x, z) \wedge \mathbf{sg}(z, w) \wedge \mathbf{down}(w, y) \rightarrow \mathbf{sg}(x, y),$$
$$\mathbf{flat}(x, y) \rightarrow \mathbf{sg}(x, y).$$

[5] Similar results are obtained if $CB = BC^k$.

The recursive clause in this program is a product of the following two clauses:

$$\mathbf{up}(x, z) \wedge \mathbf{sg}(z, y) \rightarrow \mathbf{sg}(x, y),$$
$$\mathbf{sg}(x, w) \wedge \mathbf{down}(w, y) \rightarrow \mathbf{sg}(x, y).$$

We have shown in Example 6.1 that the above clauses commute with each other. Hence, by Theorem 6.6, **sg** can be computed by processing the two clauses independently of each other, and then combining the results accordingly. This can be achieved by rewriting the original program in several ways. The most straightforward (but not necessarily the most efficient) one is the following:

$$\mathbf{down\_sg}(x, y, 0) \rightarrow \mathbf{sg}(x, y),$$
$$\mathbf{down\_sg}(x, w, i+1) \wedge \mathbf{down}(w, y) \rightarrow \mathbf{down\_sg}(x, y, i),$$
$$\mathbf{up\_sg}(x, y, i) \rightarrow \mathbf{down\_sg}(x, y, i),$$
$$\mathbf{up}(x, z) \wedge \mathbf{up\_sg}(z, y, i) \rightarrow \mathbf{up\_sg}(x, y, i+1),$$
$$\mathbf{flat}(x, y) \rightarrow \mathbf{up\_sg}(x, y, 0). \qquad \square$$

For the next theorem, we need the following definitions. An operator $B \in R$ is *uniformly bounded*, if there exist K and N, $K < N$, such that $B^N \leq B^K$. An operator $B \in R$ is *torsion*, if there exist K and N, $K < N$, such that $B^N = B^K$. The former definition is inspired by the uniform boundedness property of linear recursive Horn clauses [28, 38]. Clearly, every torsion is uniformly bounded, but the opposite is not true in general.

THEOREM 6.7. *Let $A = BC$. If $CB = BC$ and B is torsion, with N being the lowest number such that, for some $K < N$, the equality $B^K = B^N$ holds, then*

$$A^* = (BC)^* = \sum_{m=0}^{K-1} B^m C^m + (C^{N-K})^* \left(\sum_{m=K}^{N-1} B^m C^m \right).$$

PROOF. Since $B^K = B^N$, it takes an easy induction to show that

$$B^m = B^{m+i(N-K)}, \qquad \text{for all } K \leq m < N \text{ and all } i \geq 0. \tag{6.7}$$

The theorem is the result of the following transformations, which make use of (6.7):

$$\begin{aligned} A^* = (BC)^* &= \sum_{m=0}^{K-1} B^m C^m + \sum_{m=K}^{\infty} B^m C^m && \text{Corollary 6.2} \\ &= \sum_{m=0}^{K-1} B^m C^m + \sum_{m=K}^{N-1} B^m \left(\sum_{i=0}^{\infty} C^{m+i(N-K)} \right) && \text{From (6.7)} \\ &= \sum_{m=0}^{K-1} B^m C^m + (C^{(N-K)})^* \left(\sum_{m=K}^{N-1} B^m C^m \right). \end{aligned}$$ $\square$

Example 6.4 [37]. Consider the following logic program:

$$\mathbf{knows}(x, z) \wedge \mathbf{buys}(z, y) \wedge \mathbf{cheap}(y) \rightarrow \mathbf{buys}(x, y),$$
$$\mathbf{definitely\_buys}(x, y) \rightarrow \mathbf{buys}(x, y).$$

The recursive clause can be written as a product of the following two Horn clauses:

$$\textbf{knows}(x, z) \wedge \textbf{buys}(z, y) \rightarrow \textbf{buys}(x, y),$$
$$\textbf{buys}(x, y) \wedge \textbf{cheap}(y) \rightarrow \textbf{buys}(x, y).$$

The above two clauses commute with each other, since composing them in both ways yields the original recursive clause. Let the second clause correspond to the operator scheme B in the statement of Theorem 6.7. Composing that clause with itself to compute B^2 yields the clause below:

$$\textbf{buys}(x, y) \wedge \textbf{cheap}(y) \wedge \textbf{cheap}(y) \rightarrow \textbf{buys}(x, y).$$

Clearly, the equality $B^2 = B$ holds. Since all the premises of Theorem 6.7 hold, we conclude that B can be ignored from the processing of the original clause beyond its first power, that is, the original program can be replaced by the following one:

$$\textbf{knows}(x, z) \wedge \textbf{buys}(z, y) \rightarrow \textbf{buys}(x, y),$$
$$\textbf{knows}(x, z) \wedge \textbf{definitely\_buys}(z, y) \wedge \textbf{cheap}(y) \rightarrow \textbf{buys}(x, y),$$
$$\textbf{definitely\_buys}(x, y) \rightarrow \textbf{buys}(x, y). \quad \square$$

With respect to operator schemes, Theorem 6.7 identifies cases in which the powers of B and C can be computed separately, and for B, only a finite number of them is necessary, as in the case of computing B^*. This result is related to the work of Naughton on *recursively redundant predicates* [37], which was the source for the program of Example 6.4. The relationship between commutativity, uniform boundedness, and recursively redundant predicates is examined in detail elsewhere [29].

6.3. Replacing A^* with Transitive Closures of Other Operators

THEOREM 6.8. *Let A, B, C be linear operators. If $AB = BA$ and $AC = BC$, then $A^*C = B^*C$.*

PROOF. We prove by induction on $m \geq 0$ that $A^mC = B^mC$.

Basis. For $m = 0$, the claim holds trivially.

Induction Step. Assume that the claim is true for $m \geq 0$. We prove it for $m + 1$. We have the following:

$$A^{m+1}C = A(A^mC) = A(B^mC) = B^m(AC) = B^{m+1}C.$$

The second equality is by the induction hypothesis, the third equality is by commutativity of A and B, and the fourth equality is by the premise that $AC = BC$. From the above, we have that

$$A^*C = \sum_{m=0}^{\infty} (A^mC) = \sum_{m=0}^{\infty} (B^mC) = B^*C. \quad \square$$

Example 6.5. Theorem 6.8 allows the interchange of the two linear forms of transitive closure. More precisely, there are two equivalent sets of linear

Horn clauses that express the calculation of the transitive closure of a binary relation Q:

$$\begin{aligned} \mathbf{P}(x, z) \wedge \mathbf{Q}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{Q}(x, y) &\rightarrow \mathbf{P}(x, y), \end{aligned} \tag{6.8}$$

and

$$\begin{aligned} \mathbf{Q}(x, z) \wedge \mathbf{P}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{Q}(x, y) &\rightarrow \mathbf{P}(x, y). \end{aligned} \tag{6.9}$$

Clearly, the two programs are equivalent. Note, however, that they would not be equivalent if the nonrecursive Horn clauses in them did not have $\mathbf{Q}$ in their antecedent. To take into account that fact, we introduce a Horn clause that corresponds to an operator whose output is always Q for any nonempty input:

$$\mathbf{I}(u) \wedge \mathbf{Q}(x, y) \rightarrow \mathbf{P}(x, y).$$

Using the above, and assuming that I is nonempty, (6.8) and (6.9) are equivalent to the following two programs:

$$\begin{aligned} \mathbf{P}(x, z) \wedge \mathbf{Q}(z, y) &\rightarrow \mathbf{P}(x, y), \\ \mathbf{I}(u) \wedge \mathbf{Q}(x, y) &\rightarrow \mathbf{P}(x, y), \end{aligned} \tag{6.10}$$

and

$$\begin{aligned} \mathbf{Q}(x, z) \wedge \mathbf{P}(z, y) &\rightarrow \mathbf{P}(x, y) \\ \mathbf{I}(u) \wedge \mathbf{Q}(x, y) &\rightarrow \mathbf{P}(x, y). \end{aligned} \tag{6.11}$$

If A and B are the corresponding operator schemes for the recursive clauses of (6.10) and (6.11) respectively, then $AB = BA$ (Example 6.1). In addition, if C is the corresponding operator scheme for the nonrecursive clause of both (6.10) and (6.11), it is easy to see that $AC = BC$, since both products correspond to the same Horn clause:

$$\mathbf{I}(u) \wedge \mathbf{Q}(x, z) \wedge \mathbf{Q}(z, y) \rightarrow \mathbf{P}(x, y).$$

Hence, the conditions of Theorem 6.8 hold, and the two programs can be interchanged. □

THEOREM 6.9. *Let* $A = BC$. *Then*,

$$A^* = (BC)^* = 1 + B(CB)^*C. \tag{6.12}$$

PROOF. The following series of equations proves the theorem:

$$\begin{aligned} A^* = (BC)^* &= \sum_{m=0}^{\infty} (BC)^m = 1 + \sum_{m=1}^{\infty} (BC)^m \\ &= 1 + B\left(\sum_{m=0}^{\infty} (CB)^m \right) C = 1 + B(CB)^*C. \end{aligned}$$

□

Expression (6.12) corresponds to a program that is equivalent to the original one corresponding to A^*. The significant difference of the two programs is in the operator whose transitive closure is computed, namely $(CB)^*$ instead of $(BC)^*$. Depending on what B and C are, the second algorithm may be more efficient.

Example 6.6. As an example of how Theorem 6.9 may be applied, consider the following set of Horn clauses:

$$\begin{aligned} \mathbf{P}(u, v, w) \wedge \mathbf{R}(u, v, w) \wedge \mathbf{S}(w, x, y, z) &\rightarrow \mathbf{P}(x, y, z), \\ \mathbf{Q}(x, y, z) &\rightarrow \mathbf{P}(x, y, z). \end{aligned} \tag{6.13}$$

The recursive Horn clause is a product of the following two ones:

$$\begin{aligned} \mathbf{P}(u, v, t) \wedge \mathbf{R}(u, v, t) &\rightarrow \mathbf{P}'(t), \\ \mathbf{P}'(w) \wedge \mathbf{S}(w, x, y, z) &\rightarrow \mathbf{P}(x, y, z). \end{aligned}$$

The product of these two clauses in the opposite direction corresponds to the following clause:

$$\mathbf{P}'(w) \wedge \mathbf{S}(w, r, s, t) \wedge \mathbf{R}(r, s, t) \rightarrow \mathbf{P}'(t).$$

Given the above, rewriting (6.13) according to Theorem 6.9 yields the following program:

$$\begin{aligned} \mathbf{Q}(u, v, w) \wedge \mathbf{R}(u, v, w) &\rightarrow \mathbf{P}'(w), \\ \mathbf{P}'(w) \wedge S(w, r, s, t) \wedge \mathbf{R}(r, s, t) &\rightarrow \mathbf{P}'(t), \\ \mathbf{P}'(t) \wedge S(t, x, y, z) &\rightarrow \mathbf{P}(x, y, z), \\ \mathbf{Q}(x, y, z) &\rightarrow \mathbf{P}(x, y, z). \end{aligned}$$

Note that the above program is equivalent to the original one, but has a great potential of being more efficient, primarily because its recursive Horn clause is monadic, that is, has arity one, where the one of the original program had arity three. Such reductions in the arity of recursive predicates are known to significantly affect performance [9, 39]. □

6.4. Pushing Selections and Projections Through A^*

Theorem 6.10. *Let A, ρ be linear operators. If there exists another linear operator B such that*

$$\rho A \leq B\rho, \tag{6.14}$$

then $\rho A^* \leq B^*\rho$. If (6.14) holds with equality, then $\rho A^* = B^*\rho$.

Proof. Assume that $\rho A \leq B\rho$. We first show by induction on $m \geq 0$ that $\rho A^m \leq B^m\rho$.

Basis. For $m = 0$, the above formula is satisfied trivially.

Induction Step. Assume that the claim is true for some $m \geq 0$. We prove it for $m + 1$.

$$\begin{aligned} \rho A^{m+1} &= (\rho A^m) A \leq (B^m\rho) A && \text{Induction Hypothesis} \\ &= B^m(\rho A) \leq B^m(B\rho) = B^{m+1}\rho && \text{From (6.14).} \end{aligned}$$

Having established the above, we can proceed in proving the theorem.

$$\rho A^* = \rho \sum_{m=0}^{\infty} A^m \leq \left(\sum_{m=0}^{\infty} B^m \right) \rho = B^*\rho.$$

The case where (6.14) holds with equality is easily seen to be true as well. □

An interesting case arises when (6.14) holds with equality and $B = A$. Then, the above theorem states that if A and ρ commute, then ρ can be pushed through the transitive closure of A. The most common such case is expected to be for ρ being a selection. Such transformations on selection pushing were among the first proposed for recursive programs in database systems [3, 32]. Another interesting case is when (6.14) holds with equality, ρ is a projection π, and $B = A_\pi$, which is the same operator as A but operating only on some of the columns of its input (the ones indicated by the projection π). Again, in some sense, applying the above theorem results in the projection being pushed through the transitive closure of A. Usually, such transformations cause significant improvements in performance.

Example 6.7. As an application example of Theorem 6.10, consider the transitive closure program (6.8) with the query

$$\mathbf{P}(c, y)?,$$

where c is a constant. Assuming that σ is the selection and π is the projection expressed in the above query, and that A is the operator scheme that corresponds to the recursive clause of (6.8), the answer of the above query can be expressed algebraicly as $\pi\sigma A^*$. Clearly, $\sigma A = A\sigma$, since both products are equal to the following clause:

$$\mathbf{P}(x, z) \wedge \mathbf{Q}(z, y) \wedge x = c \rightarrow \mathbf{P}(x, y).$$

Also, $\pi A = A_\pi \pi$ (again, A_π is the same operator scheme as A but accepting as input and operating only on the columns specified by π), since both products are equal to the following clause:

$$\mathbf{P}(x, z) \wedge \mathbf{Q}(z, y) \rightarrow \mathbf{P}'(y).$$

Thus, the conditions of Theorem 6.10 are satisfied, and $\pi\sigma A^*$ can be replaced by $(A_\pi)^*\pi\sigma$, which corresponds to the following linear program:

$$\begin{aligned} \mathbf{P}'(z) \wedge \mathbf{Q}(z, y) &\rightarrow \mathbf{P}'(y), \\ \mathbf{Q}(c, y) &\rightarrow \mathbf{P}'(y). \end{aligned} \tag{6.15}$$

Note that (6.15) is more efficient than (6.8), since the arity of the recursive predicate has been reduced and the query selection is taken into account right from the beginning. □

COROLLARY 6.3. *Let A be a linear operator and π be a projection. If $\pi A \leq \pi$, then $\pi A^* = \pi$.*

PROOF. Assume that $\pi A \leq \pi$. From Theorem 6.10, with $B = 1$, we have that $\pi A^* \leq \pi$. By Proposition 3.3d, however, since $1 \leq A^*$, we have that $\pi \leq \pi A^*$. Hence, we can conclude that $\pi A^* = \pi$. □

Corollary 6.3 allows for the elimination of recursion when its premises hold. The projection π is pushed through A in a way that the latter disappears. Similar work on projection pushing in a logic-based setting has been conducted by Ramakrishnan et al. [40], who derived several syntactic characterizations to capture the conditions of Theorem 6.10 for pushing projections and Corollary 6.3 for elimination of (recursive) clauses. We present two examples that are

essentially taken from the above work and show that applying Corollary 6.3 produces the same results.

Example 6.8. Consider the following program-query pair:

$$\begin{aligned}
\mathbf{P}(x, u) \wedge \mathbf{Q}_2(u, w, z) &\rightarrow \mathbf{P}_1(x, u, z), \\
\mathbf{P}(x, u) \wedge \mathbf{Q}_3(u, w, z) &\rightarrow \mathbf{P}_1(x, u, z), \\
\mathbf{P}_1(x, u, z) \wedge \mathbf{Q}_4(z, y, v) &\rightarrow \mathbf{P}(x, y), \\
\mathbf{Q}_1(x, y) &\rightarrow \mathbf{P}(x, y), \\
\mathbf{P}(x, _)?
\end{aligned}$$

If A, B, and C are the operator schemes that correspond to the first three (recursive) clauses in the above program, then the whole program corresponds to the following matrix equation:

$$\begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P} \end{bmatrix} = \begin{bmatrix} 0 & A + B \\ C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P} \end{bmatrix} \oplus \begin{bmatrix} \varnothing \\ \mathbf{Q}_1 \end{bmatrix}.$$

Solving for $\mathbf{P}$ and incorporating the projection π specified in the query yields $\pi(C(A + B))^*$ (see Section 5). It is easily verifiable, however, that $\pi X \leq \pi$, for all $X \in \{A, B, C\}$. Thus, by Corollary 6.3, $\pi(C(A + B))^* = \pi$, and the above program can be replaced by the nonrecursive

$$\begin{aligned}
\mathbf{Q}_1(x, y) &\rightarrow \mathbf{P}(x, y), \\
\mathbf{P}(x, _)?
\end{aligned}$$

□

Example 6.9. As a second example, consider the following program–query pair:

$$\begin{aligned}
\mathbf{P}_1(x, z, u) \wedge \mathbf{Q}_1(z, u, y) &\rightarrow \mathbf{P}(x, y), \\
\mathbf{P}_1(x, w, w) \wedge \mathbf{Q}_2(w, z, u) &\rightarrow \mathbf{P}_1(x, z, u), \\
\mathbf{P}(x, v) \wedge \mathbf{Q}_3(v, z, u) &\rightarrow \mathbf{P}_1(x, z, u), \\
\mathbf{Q}_4(x, z, u) &\rightarrow \mathbf{P}_1(x, z, u), \\
\mathbf{P}(x, _)?
\end{aligned}$$

If A, B, and C are the operator schemes that correspond to the first three clauses in the above program then the whole program corresponds to the following matrix equation:

$$\begin{bmatrix} \mathbf{P} \\ \mathbf{P}_1 \end{bmatrix} = \begin{bmatrix} 0 & A \\ C & B \end{bmatrix} \begin{bmatrix} \mathbf{P} \\ \mathbf{P}_1 \end{bmatrix} \oplus \begin{bmatrix} \varnothing \\ \mathbf{Q}_4 \end{bmatrix}.$$

Solving for $\mathbf{P}$ and incorporating the projection π specified in the query yields $\pi(AB^*C)^*AB^*$. Again, it is easily verifiable that $\pi X \leq \pi$, for all $X \in \{A, B, C\}$. Thus, by Corollary 6.3, $\pi(AB^*C)^*AB^* = \pi AB^*$. The above operator corresponds to the following program, which can be obtained from the original one by removing the third clause, that is, the one that corresponds to C:

$$\begin{aligned}
\mathbf{P}_1(x, z, u) \wedge \mathbf{Q}_1(z, u, y) &\rightarrow \mathbf{P}(x, y), \\
\mathbf{P}_1(x, w, w) \wedge \mathbf{Q}_2(w, z, u) &\rightarrow \mathbf{P}_1(x, z, u), \\
\mathbf{Q}_4(x, z, u) &\rightarrow \mathbf{P}_1(x, z, u), \\
\mathbf{P}(x, _)?
\end{aligned}$$

□

7. *Algebraic Transformations of Linear Recursion at the Ordering Stage*

In this section, we present algebraic transformations of the query answer that affect the ordering stage of query optimization (Figure 1). We show how several algorithms that have been proposed in the literature are expressed as different parenthesizations of an algebraic representation of the query answer. In the first subsection, we derive expressions that correspond to algorithms that are applicable to any linear recursive program. Practically, there is no limit in the number of expressions that are equal to A^*. We only present a small number of them that have been previously proposed in the literature. In the second subsection, we derive expressions that correspond to algorithms that are applicable only to recursive programs of a specific form.

As in the previous section, all the optimization results presented in this one depend solely on the algebraic properties of closed semirings. Hence, the results can be generalized to mutual linear recursion as well, by simply using linear operator matrices in place of linear operators. Moreover, all results in this section hold for both operators and operator schemes.

7.1 General Transformations of A^*

7.1.1 *Naive Evaluation* [3, 6]. This is the original algorithm proposed for the evaluation of a (not necessarily linear) recursive program. Its algebraic expression in the linear case is based on the fact that $\Sigma_{m=0}^{n} A^m = (1 + A)^n$. Hence, we have that

$$A^* = \lim_{n \to \infty} (1 + A)^n.$$

(Recall that whenever explicit parenthesization is omitted, right associativity is assumed for multiplication.) Moreover, each power is computed from the previous one: $(1 + A)^{m+1} = (1 + A)(1 + A)^m$.

7.1.2 *Semi-Naive Evaluation* [6]. This algorithm corresponds to the definition of A^* as a series, that is, $A^* = \Sigma_{m=0}^{\infty} A^m$. Again, each power is computed from the previous one: $A^{m+1} = AA^m$.

7.1.3 *Smart/Logarithmic Evaluation* [27, 52]. This algorithm corresponds to the following form:

$$A^* = \prod_{k=0}^{\infty} (1 + A^{2^k}) = \cdots (1 + A^4)(1 + A^2)(1 + A).$$

The number of multiplications in the expression of smart is much smaller than that of semi-naive (for any finite expansion of A^*, it is approximately equal to $2 * \log_2 N$ for smart vs. N for semi-naive), but they involve larger operators.

7.1.4 *Minimal Evaluation* [27]. This algorithm corresponds to the following formula:

$$\begin{aligned} A^* &= \prod_{k=0}^{\infty} (1 + A^{3^k} + A^{2*3^k}) \\ &= \cdots (1 + A^9 + A^{18})(1 + A^3 + A^6)(1 + A + A^2). \end{aligned}$$

This expression has approximately $3 \log_3 N$ multiplications (when semi-naive has N). Clearly, algorithms can be created that need $n \log_n N$ multiplications

for arbitrary n. They correspond to the following family of formulas:

$$A^* = \prod_{k=0}^{\infty} \left(\sum_{l=0}^{n-1} A^{l*n^k} \right).$$

The formulas for smart and minimal are special cases of the above for $n = 2$ and $n = 3$, respectively. The expression $n \log_n N$, which gives the number of multiplications of this formula, with n restricted to the integers, has a minimum for $n = 3$, regardless of the value of N.

7.1.5 *Query-Subquery (QSQ) Evaluation* [55]. For the query that involves a selection, the QSQ Evaluation has been proposed by Vieille. QSQ tries to take into account the selection as much as possible. The corresponding algebraic formula is shown below:

$$\sigma A^* = \sigma \sum_{k=0}^{\infty} A^k = \sum_{k=0}^{\infty} (\sigma A^k) = \sigma + \sum_{k=1}^{\infty} (\sigma A^{k-1}) A.$$

Note that the selection σ is taken into account from the beginning, and that the product of σ with each power of A is computed from the product of σ with the previous one: $\sigma A^k = (\sigma A^{k-1}) A$.

7.1.6 *Prolog* [16]. The formula that corresponds to the execution plan of Prolog is the same as QSQ. The difference is in the details of the planning stage, that is, Prolog processes one tuple at a time, whereas QSQ processes one relation at a time. This is an example of the fact that the algebra developed in this paper is not a useful tool for the planning stage of a recursive query optimizer: two different execution plans are represented by the same formula at the ordering stage.

7.2. Specialized Transformations of A^*. Let $A = BC = CB$. By Corollary 6.2, we have that $A^* = \Sigma_{m=0}^{\infty} B^m C^m$. This formula corresponds to a rewritten program for A^* that keeps track of the powers m in B and C and then multiplies the corresponding ones (Example 6.3). There are several different parenthesizations of the above formula, each one of which corresponds to an algorithm that has been previously proposed in the literature. The algorithms and their corresponding parenthesizations are presented below, for the case where the product of a selection σ with A^* is desired, where $\sigma B = B\sigma$. In that case, we have that

$$\sigma A^* = \sum_{m=0}^{\infty} B^m \sigma C^m. \tag{7.1}$$

7.2.1 *Henschen–Naqvi* [25]. According to the algorithm proposed by Henschen and Naqvi, (7.1) is parenthesized as

$$\sigma A^* = \sum_{m=0}^{\infty} B^m (\sigma C^m) = \sigma + \sum_{m=1}^{\infty} B^m ((\sigma C^{m-1}) C).$$

Moreover, the product of σ with a power of C is computed from the same product with the previous power: $\sigma C^m = (\sigma C^{m-1}) C$.

7.2.2 *Counting* [7, 43, 44]. The Counting algorithm as proposed by Bancilhon et al. [7] and by Sacca and Zaniolo [44] corresponds to the same

formula as Henschen–Naqvi. The two algorithms are different in implementation details that belong to the planning stage of the query optimizer, that is, whether all the products σC^m are computed first, before any multiplications with B (Counting), or every product is immediately multiplied with the corresponding number of B's (Henschen–Naqvi). (Note that explicit parenthesization of a formula only partially specifies the order of processing of each operator.) This difference can be further enhanced by the form of duplicate elimination performed by the two algorithms. (It has been noted elsewhere as well that Counting can be thought of as an efficient implementation of Henschen–Naqvi [7].)

We want to emphasize that, although Corollary 6.2 deals with the product of two operators only, its results can be generalized to products of an arbitrary number of operators, thus capturing algebraicly the results of previous work by Sacca and Zaniolo [43, 44]. In particular, given a set of operators $\{A_i\}$, $1 \le i \le n$, that are mutually commutative, and a set of selections $\{\sigma_i\}$, $0 \le i \le n$, such that σ_i commutes with all operators except A_i, the following holds:

$$\sigma_0\sigma_1\sigma_2 \cdots \sigma_n(A_1A_2 \cdots A_n)^* = \sum_{m=0}^{\infty} (\sigma_1 A_1^m)(\sigma_2 A_2^m) \cdots (\sigma_n A_n^m)\sigma_0.$$

One can now apply the Counting algorithm computing the powers of each A_i separately and then multiplying the corresponding ones together. Usually, most of the selections will not be present. In the presense of multiple selections, for each i such that σ_i is present, it is an interesting optimization problem to decide whether to compute $\sigma_i A_i^{m+1}$ from $\sigma_i A_i^m$ or not.

7.2.3 *Shapiro–McKay* [49]. Shapiro and McKay presented an algorithm that corresponds to the following parenthesization of (7.1):

$$\sigma A^* = \sigma + \sigma \sum_{m=1}^{\infty} B(B^{m-1}C^{m-1})C.$$

Only two multiplications for each power of A are performed, but the selection σ is not taken into account in the evaluation of the transitive closure.

7.2.4 *Han–Lu* [24]. In a performance evaluation conducted by Han and Lu, three algorithms were presented for (7.1). The first was Henschen–Naqvi, the second was Shapiro–McKay, and the third we call Han–Lu. The Han–Lu algorithm corresponds to the following parenthesization of (7.1):

$$\sigma A^* = \sigma + \sum_{m=1}^{\infty} (B^{m-1}B)((\sigma C^{m-1})C).$$

Note that not only σC^{m+1} is computed from σC^m (as in Henschen–Naqvi), but also B^{m+1} is computed from B^m.

8. *Multilinear Recursion*

We now turn our attention to nonlinear recursion. In contrast to our approach to linear recursion, where we first studied immediate recursion and then generalized to mutual recursion, we study mutual nonlinear recursion in its general form directly.

Recall that D is a database and C_D is a fixed set of constants in D. Consider a set of mutually recursive Horn clauses, and let $\underline{\mathbf{P}} = (\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_n)$ be the vector of relations that appear in the consequents of its elements with arities $\{a_1, a_2, \ldots, a_n\}$. Again, we make no assumptions about the relations in the Horn clauses being finite. Also consider a set of n nonrecursive Horn clauses of the form $\mathbf{Q}_i \rightarrow \mathbf{P}_i$, $i = 1, 2, \ldots, n$, and let $\underline{\mathbf{Q}}$ be the vector $\underline{\mathbf{Q}} = (\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_n)$. These two sets of Horn clauses can be expressed in relational terms as follows. Consider the set of recursive Horn clauses having $\mathbf{P}_i$ in their consequent, for some $1 \leq i \leq n$. Let f_i be the function that represents the operations on $\underline{\mathbf{P}}$ that these Horn clauses express. (If there are multiple Horn clauses in that set, then f_i involves taking the union of relations.) Then, the complete set of Horn clauses takes the following functional form:

$$\begin{aligned} f_i(\underline{\mathbf{P}}) &\subseteq \mathbf{P}_i, \qquad i = 1, 2, \ldots, n, \\ \mathbf{Q}_i &\subseteq \mathbf{P}_i, \qquad i = 1, 2, \ldots, n. \end{aligned} \tag{8.1}$$

As in Section 4, the minimal solution of (8.1) is the minimal solution of the set of equations

$$f_i(\underline{\mathbf{P}}) \cup \mathbf{Q}_i = \mathbf{P}_i, \qquad i = 1, 2, \ldots, n. \tag{8.2}$$

Let $\underline{P} = 2^{C_D^{a_1}} \times \cdots \times 2^{C_D^{a_n}}$, $a_i \geq 1$. Clearly, $\underline{\mathbf{P}} \in \underline{P}$. Recall that *addition* $\oplus$ in $\underline{P}$ is defined as

$$\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}' = (\mathbf{P}_1 \cup \mathbf{P}_1', \mathbf{P}_2 \cup \mathbf{P}_2', \ldots, \mathbf{P}_n \cup \mathbf{P}_n').$$

In addition, if $\underline{\varnothing}$ is the n-vector $(\varnothing, \varnothing, \ldots, \varnothing)$, note that $\underline{\varnothing} \oplus \underline{\mathbf{P}} = \underline{\mathbf{P}} \oplus \underline{\varnothing} = \mathbf{P}$. The system of equations (8.2) can now be written as a single equation

$$\underline{\mathbf{P}} = f(\underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}}. \tag{8.3}$$

The following definitions introduce some classes of functions on $\underline{P}$ that are of specific interest to the algebraic formulation of Horn clause recursion.

Definition 8.1. A function $f: \underline{P} \rightarrow \underline{P}$ is *linear* if

(a) for all vectors $\underline{\mathbf{P}}$, $\underline{\mathbf{P}}'$ in its domain, $f(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}') = f(\underline{\mathbf{P}}) \oplus f(\underline{\mathbf{P}}')$, and
(b) $f(\underline{\varnothing}) = \underline{\varnothing}$.

Definition 8.2. A function $g: \underline{P} \times \underline{P} \rightarrow \underline{P}$ is *bilinear* if for a given $\underline{\mathbf{P}}'$, $g(\underline{\mathbf{P}}, \underline{\mathbf{P}}')$ is linear in $\underline{\mathbf{P}}$, and for a given $\underline{\mathbf{P}}$, $g(\underline{\mathbf{P}}, \underline{\mathbf{P}}')$ is linear in $\underline{\mathbf{P}}'$.

Definition 8.3. A function $g: \underline{P} \times \underline{P} \times \cdots \times \underline{P} \rightarrow \underline{P}$ (the domain of g is the product of $\underline{P}$ m times) is *m-linear* if for all i, given $\underline{\mathbf{P}}_1, \ldots, \underline{\mathbf{P}}_{i-1}$, $\underline{\mathbf{P}}_{i+1}, \ldots, \underline{\mathbf{P}}_m$, the function $g(\underline{\mathbf{P}}_1, \ldots, \underline{\mathbf{P}}_i, \ldots, \underline{\mathbf{P}}_m)$ is linear in $\underline{\mathbf{P}}_i$. When m is not specified, such functions are called *multilinear*.

A system of recursive equations of the form (8.3) is *linear, bilinear,* or *m-linear*, if f is linear, bilinear, or m-linear, respectively. Linear systems of recursive equations can further be put into the form shown in Section 5 and analyzed using the properties of closed semirings. Unfortunately, this is not possible for nonlinear systems. The importance of m-linear functions in the study of Horn clause recursion is demonstrated in the following proposition.

PROPOSITION 8.1. *If* $\underline{\mathbf{P}} = f(\underline{\mathbf{P}}, \underline{\mathbf{P}}, \ldots, \underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}}$ *is the equation representing a set of mutually recursive Horn clauses, where* $f:(\underline{P})^m \to \underline{P}$, *then* f *is* m*-linear.*

PROOF. Clearly, the m-linearity of f depends on the k-linearity, $k \leq m$, of the individual recursive Horn clauses. Let g be the function corresponding to one such clause, and assume that $k \leq m$ elements of $\underline{\mathbf{P}}$ appear in it. Given fixed $\underline{\mathbf{Q}}_1, \ldots, \underline{\mathbf{Q}}_{i-1}, \underline{\mathbf{Q}}_{i+1}, \ldots, \underline{\mathbf{Q}}_k$, consider $g(\underline{\mathbf{Q}}_1, \ldots, \underline{\mathbf{Q}}_i, \ldots, \underline{\mathbf{Q}}_k)$ as a function of $\underline{\mathbf{Q}}_i$. Clearly, g can be expressed as a composition of projections, selections, and cross products of $\underline{\mathbf{Q}}_1, \ldots, \underline{\mathbf{Q}}_{i-1}, \underline{\mathbf{Q}}_{i+1}, \ldots, \underline{\mathbf{Q}}_k$. Thus, in general, g corresponds to an operator in R, which is a set of linear relational operators. By Proposition 3.1, g is linear in $\underline{\mathbf{Q}}_i$. This is true for all i, so by Definition 8.3, g is k-linear. Since g was chosen arbitrarily among the functions of the individual Horn clauses, we can conclude that f is m-linear. □

Having established the multilinearity of all recursive Horn clause systems, we proceed by showing the universality of bilinear recursion in the vector form as we have defined it. This is achieved by the following propositions.

PROPOSITION 8.2. *A recursion consisting of only linear and bilinear terms is equivalent to one with only bilinear terms.*

PROOF. Consider the following equation, consisting of only linear and bilinear terms:

$$\underline{\mathbf{P}} = \underline{A}\underline{\mathbf{P}} \oplus g(\underline{\mathbf{P}}, \underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}}.$$

The linear term, which involves the operator matrix $\underline{A}$, can always be eliminated by treating the bilinear term as constant and solving the above equation using the techniques of Sections 4 and 5:

$$\underline{\mathbf{P}} = \underline{A}^* g(\underline{\mathbf{P}}, \underline{\mathbf{P}}) \oplus \underline{A}^* \underline{\mathbf{Q}} = g'(\underline{\mathbf{P}}, \underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}}'.$$

Given a fixed $\underline{\mathbf{P}}'$, the linearity of $\underline{A}^*$ and the bilinearity of g establishes the linearity of $g'(\underline{\mathbf{P}}, \underline{\mathbf{P}}')$ in $\underline{\mathbf{P}}$:

$$\begin{aligned} g'(\underline{\mathbf{P}} \oplus \underline{\mathbf{Q}}, \underline{\mathbf{P}}') &= \underline{A}^* g(\underline{\mathbf{P}} \oplus \underline{\mathbf{Q}}, \underline{\mathbf{P}}') = \underline{A}^* \big(g(\underline{\mathbf{P}}, \underline{\mathbf{P}}') \oplus g(\underline{\mathbf{Q}}, \underline{\mathbf{P}}') \big) \\ &= A^* g(\underline{\mathbf{P}}, \underline{\mathbf{P}}') \oplus \underline{A}^* g(\underline{\mathbf{Q}}, \underline{\mathbf{P}}') = g'(\underline{\mathbf{P}}, \underline{\mathbf{P}}') \oplus g'(\underline{\mathbf{Q}}, \underline{\mathbf{P}}'), \\ g'(\underline{\varnothing}, \underline{\mathbf{P}}') &= \underline{A}^* g(\underline{\varnothing}, \underline{\mathbf{P}}') = \underline{A}^* \underline{\varnothing} = \varnothing. \end{aligned}$$

Linearity of $g'(\underline{\mathbf{P}}, \underline{\mathbf{P}}')$ on $\underline{\mathbf{P}}'$ is established similarly. Hence, the function $g' = \underline{A}^* g$ is bilinear. □

PROPOSITION 8.3. *Any multilinear recursion can be reduced to a bilinear recursion.*

PROOF. Consider the m-linear recursive equation.

$$\underline{\mathbf{P}} = f(\underline{\mathbf{P}}, \underline{\mathbf{P}}, \ldots, \underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}}. \tag{8.4}$$

Define $m-1$ bilinear functions $g_1, \ldots, g_{m-1}$, whose composition is equal to f, and $m-2$ new vectors $\underline{\mathbf{P}}_1, \ldots, \underline{\mathbf{P}}_{m-2}$ such that

$$\begin{aligned}
\underline{\mathbf{P}}_1 &= g_1(\underline{\mathbf{P}}, \underline{\mathbf{P}}) \\
\underline{\mathbf{P}}_2 &= g_2(\underline{\mathbf{P}}, \underline{\mathbf{P}}_1) \\
&\vdots \\
\underline{\mathbf{P}}_{m-2} &= g_{m-2}(\underline{\mathbf{P}}, \underline{\mathbf{P}}_{m-3}) \\
\underline{\mathbf{P}} &= g_{m-1}(\underline{\mathbf{P}}, \underline{\mathbf{P}}_{m-2}) \oplus \underline{\mathbf{Q}}.
\end{aligned}$$

Clearly, the above system is equivalent to (8.4) and it is bilinear. □

Because of the universal character of bilinearity expressed in Propositions 8.2 and 8.3, we can confine our consideration to bilinear recursion only of the form

$$\underline{\mathbf{P}} = g(\underline{\mathbf{P}}, \underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}}. \tag{8.5}$$

All Horn-clause derived recursions can be treated in this way. The minimal solution to (8.5) is provided by Tarski's Theorem [51]. For its proof, a partial order on $\underline{P}$ is needed, which is defined as $\underline{\mathbf{P}} \subseteq \underline{\mathbf{P}}' \Leftrightarrow (\mathbf{P}_1 \subseteq \mathbf{P}'_1, \ldots, \mathbf{P}_n \subseteq \mathbf{P}'_n)$.

THEOREM 8.1. *The minimal solution of eq.* (8.5) *is the limit of the sequence*

$$\begin{aligned}
\underline{\mathbf{P}}_{m+1} &= g(\underline{\mathbf{P}}_m, \underline{\mathbf{P}}_m) \oplus \underline{\mathbf{Q}}, \\
\underline{\mathbf{P}}_0 &= \underline{\varnothing}.
\end{aligned} \tag{8.6}$$

PROOF. Because g is bilinear, g is also monotone on both arguments of it (by Definition 8.2 and Proposition 3.2). The following induction on m proves that $\{\underline{\mathbf{P}}_m\}$ is an increasing sequence, with respect to the partial order in $\underline{P}$.

Basis. For $m = 0$, it is $\underline{\mathbf{P}}_0 = \underline{\varnothing} \subseteq \underline{\mathbf{Q}} = \underline{\mathbf{P}}_1$.

Induction Step. Assume that, for some $m \geq 0$, it is $\underline{\mathbf{P}}_m \subseteq \underline{\mathbf{P}}_{m+1}$. This, together with the monotonicity of g, yields

$$\underline{\mathbf{P}}_m \subseteq \underline{\mathbf{P}}_{m+1} \Rightarrow g(\underline{\mathbf{P}}_m, \underline{\mathbf{P}}_m) \subseteq g(\underline{\mathbf{P}}_{m+1}, \underline{\mathbf{P}}_{m+1}) \Rightarrow \underline{\mathbf{P}}_{m+1} \subseteq \underline{\mathbf{P}}_{m+2}.$$

If follows that $\{\underline{\mathbf{P}}_m\}$ monotonically converges, and its limit, which is the solution of (8.5), is equal to $\underline{\mathbf{P}} = \lim_{m \to \infty} \underline{\mathbf{P}}_m = \bigcup_{m=0}^{\infty} \underline{\mathbf{P}}_m$. □

Note that straightforward adoption of (8.6) as an iterative procedure to compute the limit of the sequence is equivalent to the naive evaluation [3, 6], which was described in Section 7.1 in its special form for linear recursion.

For the rest of the paper, g is viewed as multiplication, that is, $g(\underline{\mathbf{P}}, \underline{\mathbf{Q}}) = \underline{\mathbf{P}} \cdot \underline{\mathbf{Q}}$. Note that $\cdot$ is not necessarily associative. Define the system

$$E_{\underline{P}} = (\underline{P}, \oplus, \cdot, \underline{\varnothing}),$$

As follows:

$\underline{P}$ The set of n-vectors of relations from C_D with arities $\{a_1, a_2, \ldots, a_n\}$.
$\oplus$ Addition of vectors as defined above.
$\cdot$ Multiplication of vectors defined as $\underline{\mathbf{P}} \cdot \underline{\mathbf{Q}} = g(\underline{\mathbf{P}}, \underline{\mathbf{Q}})$.
$\underline{\varnothing}$ The additive identity, that is, the n-vector of empty relations.

The following theorem characterizes the algebraic structure of $E_{\underline{P}}$.

THEOREM 8.2. *The system $E_{\underline{P}}$ is a nonassociative closed semiring without identity.*

PROOF. The proof is straightforward and is omitted. It depends on well-known properties of sets and unions of sets and on the bilinearity of $\cdot$. □

In $E_{\underline{P}}$, power is defined as

$$\underline{\mathbf{P}}^1 = \underline{\mathbf{P}}, \underline{\mathbf{P}}^n = \underline{\mathbf{P}} \cdot \underline{\mathbf{P}}^{n-1}, \qquad \text{for all } n \geq 1.$$

Note that, since $\cdot$ is nonassociative, it is not necessarily true that $\underline{\mathbf{P}}^n = \underline{\mathbf{P}}^{n-1} \cdot \underline{\mathbf{P}}$.

With the above, we have established the appropriate algebraic framework to study multilinear recursion, namely the system $E_{\underline{P}}$. We have shown that any such recursion is equivalent to a purely bilinear one. In the next two sections, we investigate various conditions under which a bilinear recursion is equivalent to a linear one.

9. *Equivalence of Bilinear to Linear Recursion*

In the sequel, since g has been represented syntactically as multiplication, the following equation is used instead of (8.5):

$$\underline{\mathbf{P}} = \underline{\mathbf{P}} \cdot \underline{\mathbf{P}} \oplus \underline{\mathbf{Q}}. \tag{9.1}$$

Moreover, none of the forthcoming results is of any value when $\cdot$ is parameterized with actual relations, so we shall always be concerned with $\cdot$ being parameterized by relation schemes (i.e., the database will not be taken into account). If a bilinear recursion (9.1) is equivalent to a linear one, it is called *linearizable*. Linearizability is not known to be decidable [21]. We restrict our attention to a specific type of linearizability. In particular, we want to derive conditions that ensure the equivalence of (9.1) to a linear equation of the form

$$\underline{\mathbf{P}} = \underline{\mathbf{P}} \cdot \underline{\mathbf{Q}} \oplus \underline{\mathbf{Q}}$$

or of the form

$$\underline{\mathbf{P}} = \underline{\mathbf{Q}} \cdot \underline{\mathbf{P}} \oplus \underline{\mathbf{Q}}. \tag{9.2}$$

In the former case, (9.1) is called *right-linearizable*, whereas in the latter case it is called *left-linearizable*. The two cases are completely symmetrical, so we are mostly concerned with left-linearizability. Note that the solution of (9.2) is $\sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k$.[6]

LEMMA 9.1. *Let $\underline{\mathbf{P}}^*$ denote the solution of* (9.1). *Then, the following holds*:

$$\underline{\mathbf{P}}^* \supseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k.$$

PROOF. Using (8.6), we prove by induction on m, $m \geq 0$, that

$$\underline{\mathbf{P}}_{m+1} \supseteq \sum_{k=1}^{m+1} \underline{\mathbf{Q}}^k.$$

[6] $\sum$ denotes a series with respect of $\oplus$.

Basis. For $m = 0$, the above yields $\underline{\mathbf{P}}_1 \supseteq \underline{\mathbf{Q}}$, which is a consequence of (8.6) with $\underline{\mathbf{P}}_0 = \underline{\varnothing}$.

Induction Step. Assume that the claim is true for some $m \geq 0$. We prove it for $m + 1$. From (8.6) we have that

$$\begin{aligned}
\underline{\mathbf{P}}_{m+2} &= \underline{\mathbf{P}}_{m+1} \cdot \underline{\mathbf{P}}_{m+1} \oplus \underline{\mathbf{Q}} \\
&\supseteq \sum_{l=1}^{m+1} \underline{\mathbf{Q}}^l \cdot \sum_{k=1}^{m+1} \underline{\mathbf{Q}}^k \oplus \underline{\mathbf{Q}} && \text{Induction hypothesis and monotonicity of } \cdot \\
&\supseteq \underline{\mathbf{Q}} \cdot \sum_{k=1}^{m+1} \underline{\mathbf{Q}}^k \oplus \underline{\mathbf{Q}} = \sum_{k=1}^{m+2} \underline{\mathbf{Q}}^k. && \text{Monotonicity of } \cdot
\end{aligned}$$

Taking the limits of the two sequences, $\underline{\mathbf{P}}_m$ and $\sum_{k=1}^{m} \underline{\mathbf{Q}}^k$, we concluded that $\underline{\mathbf{P}}^* \supseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k$. □

As we prove later, associativity of $\cdot$ is a simple, albeit strong, condition to ensure linearizability. The latter, however, is ensured by a range of conditions that are weaker than associativity. They are all variants of the notions of power-associativity and alternativeness. We have borrowed both terms from the study of nonassociative algebras, where they are in common use [48]. We proceed from the strongest (least general) to the weakest (most general) condition.

Definition 9.1. The bilinear multiplication $\cdot$ is called *left-subalternative* if, for all $\underline{\mathbf{P}}$, $\underline{\mathbf{Q}} \in \underline{P}$, there exists $n \geq 1$ such that

$$\underline{\mathbf{P}}^2 \cdot \underline{\mathbf{Q}} \subseteq \underbrace{\underline{\mathbf{P}} \cdot \Big(\cdots \big(\underline{\mathbf{P}} \cdot (\underline{\mathbf{P}} \cdot \underline{\mathbf{Q}})\big) \cdots \Big)}_{n}, \tag{9.3}$$

and it is called *right-subalternative* if, for all $\underline{\mathbf{P}}$, $\underline{\mathbf{Q}} \in \underline{\mathbf{P}}$, there exists $n \geq 1$ such that

$$\underline{\mathbf{P}} \cdot \underline{\mathbf{Q}}^2 \subseteq \underbrace{\Big(\cdots \big((\underline{\mathbf{P}} \cdot \underline{\mathbf{Q}}) \cdot \underline{\mathbf{Q}}\big) \cdots \Big)}_{n} \cdot \underline{\mathbf{Q}}.$$

It is *subalternative* if it is both left- and right-subalternative. If $n = 2$ and the above equations hold with equality, $\cdot$ is *left-alternative, right-alternative*, and *alternative*, respectively.

THEOREM 9.1. *If $\cdot$ is left-subalternative, then* (9.1) *is left-linearizable.*

PROOF. Using (8.6), we first prove by induction on $m \geq 0$ that, if $\cdot$ is left-subalternative, then $\underline{\mathbf{P}}_m$ satisfies the following two formulas:

$$\underline{\mathbf{P}}_{m+1} \subseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k, \tag{9.4}$$

and

$$\underline{\mathbf{P}}_{m+1} \cdot x \subseteq \sum_{k=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\underline{\mathbf{Q}} \cdot \Big(\cdots (\underline{\mathbf{Q}} \cdot x) \cdots \Big)\Big)}_{n}, \quad \text{for all } x. \tag{9.5}$$

Basis. For $m = 0$, (9.4) is derived from $\underline{\mathbf{P}}_1 \subseteq \underline{\mathbf{Q}}$, which is a consequence of (8.6) with $\underline{\mathbf{P}}_0 = \underline{\varnothing}$. Similarly, (9.5) follows immediately from (8.6): $\underline{\mathbf{P}}_1 \cdot x \subseteq \underline{\mathbf{Q}} \cdot x$.

Induction Step. Assume that the claim is true for some $m \geq 0$. We prove it for $m + 1$. From (8.6), we have that

$$
\begin{aligned}
&(\underline{\mathbf{P}}_{m+2}) \cdot x \\
&\quad = (\underline{\mathbf{P}}_{m+1} \cdot \underline{\mathbf{P}}_{m+1} \oplus \underline{\mathbf{Q}}) \cdot x \\
&\quad = (\underline{\mathbf{P}}_{m+1} \cdot \underline{\mathbf{P}}_{m+1}) \cdot x \oplus \underline{\mathbf{Q}} \cdot x && \text{Bilinearity of } \cdot \\
&\quad \subseteq \underbrace{\underline{\mathbf{P}}_{m+1} \cdot \Big(\cdots (\underline{\mathbf{P}}_{m+1} \cdot (\underline{\mathbf{P}}_{m+1} \cdot x)) \cdots \Big)}_{n} \oplus \underline{\mathbf{Q}} \cdot x \\
&\qquad && \text{Left-subalternativeness of } \cdot \\
&\quad \subseteq \sum_{j=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\cdots \Big(\underline{\mathbf{Q}} \cdot \Big(\cdots \Big(\sum_{k=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\cdots (\underline{\mathbf{Q}} \cdot x) \cdots \Big)}_{k} \Big) \cdots \Big) \Big) \cdots \Big)}_{j} \cdots \Big) \oplus \underline{\mathbf{Q}} \cdot x \\
&\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{n} \\
&\qquad \text{Repeated applications of induction hypothesis (9.5) and} \\
&\qquad \text{monotonicity of } \cdot \\
&\quad = \sum_{k=n}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\underline{\mathbf{Q}} \cdot \Big(\cdots (\underline{\mathbf{Q}} \cdot x) \cdots \Big) \Big)}_{k} \oplus \underline{\mathbf{Q}} \cdot x && \text{Bilinearity of } \cdot \\
&\quad \subseteq \sum_{k=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\underline{\mathbf{Q}} \cdot \Big(\cdots (\underline{\mathbf{Q}} \cdot x) \cdots \Big) \Big)}_{k}
\end{aligned}
$$

Similarly, we have that

$$
\begin{aligned}
\underline{\mathbf{P}}_{m+2} &= \underline{\mathbf{P}}_{m+1} \cdot \underline{\mathbf{P}}_{m+1} \oplus \underline{\mathbf{Q}} \\
&\subseteq (\underline{\mathbf{P}}_{m+1}) \cdot \Big(\sum_{l=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\underline{\mathbf{Q}} \cdot \Big(\cdots (\underline{\mathbf{Q}} \cdot \underline{\mathbf{Q}}) \cdots \Big) \Big)}_{l} \Big) \oplus \underline{\mathbf{Q}} \\
&\qquad \text{Induction hypothesis (9.4) and monotonicity of } \cdot \\
&\subseteq \sum_{k=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\cdots \Big(\underline{\mathbf{Q}} \cdot \Big(\sum_{l=1}^{\infty} \underbrace{\underline{\mathbf{Q}} \cdot \Big(\cdots (\underline{\mathbf{Q}} \cdot \underline{\mathbf{Q}}) \cdots \Big)}_{l} \Big) \Big) \cdots \Big)}_{k} \\
&\qquad \text{Induction hypohtesis (9.5)} \\
&= \sum_{k=1}^{\infty} \underline{\mathbf{Q}} \cdot \Big(\underline{\mathbf{Q}} \cdot \Big(\underbrace{\cdots (\underline{\mathbf{Q}} \cdot \underline{\mathbf{Q}}) \cdots}_{k} \Big) \Big). && \text{Bilinearity of } \cdot
\end{aligned}
$$

This concludes the induction step, which proves (9.4) and (9.5). Equation (9.4) implies that the solution $\underline{\mathbf{P}}^*$ of (9.1) satisfies $\underline{\mathbf{P}}^* \subseteq \Sigma_{k=1}^{\infty} \underline{\mathbf{Q}}^k$. Together with Lemma 9.1, this implies that the solution of (9.1) is given by $\underline{\mathbf{P}}^* = \Sigma_{k=1}^{\infty} \underline{\mathbf{Q}}^k$, that is, it is equal to the solution of (9.2). Hence, because of the left-subalternativeness of $\cdot$, the bilinear (9.1) is left-linearizable. □

COROLLARY 9.1. *If $\cdot$ is right-subalternative, then* (9.1) *is right-linearizable.*

PROOF. This is the symmetric case of Theorem 9.1 and can be proved similarly. □

COROLLARY 9.2. *If $\cdot$ is subalternative or alternative, then* (9.1) *is both left- and right-linearizable.*

PROOF. This is a straightforward consequence of Theorem 9.1, Corollary 9.1, and Definition 9.1. □

The following is an example of a bilinear Horn clause that is right-alternative but not left-alternative.

Example 9.1. Consider the bilinear Horn clause

$$\mathbf{P}(x, z) \wedge \mathbf{P}(y, v) \rightarrow \mathbf{P}(x, y).$$

Let $\cdot$ represent the function of the Horn clause. We show that $(\mathbf{Q} \cdot \mathbf{P}) \cdot \mathbf{P} = \mathbf{Q} \cdot (\mathbf{P} \cdot \mathbf{P})$, whereas $(\mathbf{P} \cdot \mathbf{P}) \cdot \mathbf{Q} \neq \mathbf{P} \cdot (\mathbf{P} \cdot \mathbf{Q})$. The Horn clauses that correspond to the above algebraic equations are

$$\mathbf{Q}(x, z') \wedge \mathbf{P}(z, v') \wedge \mathbf{P}(y, v) \rightarrow \mathbf{P}(x, y), \tag{9.6}$$
$$\mathbf{Q}(x, z) \wedge \mathbf{P}(y, z'') \wedge \mathbf{P}(v, v'') \rightarrow \mathbf{P}(x, y), \tag{9.7}$$
$$\mathbf{P}(x, z') \wedge \mathbf{P}(z, v') \wedge \mathbf{Q}(y, v) \rightarrow \mathbf{P}(x, y), \tag{9.8}$$
$$\mathbf{P}(x, z) \wedge \mathbf{P}(y, z'') \wedge \mathbf{Q}(v, v'') \rightarrow \mathbf{P}(x, y). \tag{9.9}$$

Clearly, (9.6) and (9.7) are equivalent, whereas (9.8) and (9.9) are not. This implies that the original Horn clause corresponds to a function that is right-alternative but not left-alternative. Nevertheless, Corollary 9.1 guarantees that it is equivalent to a linear recursion. □

COROLLARY 9.3. *If $\cdot$ is associative, then* (9.1) *is both left- and right-linearizable.*

PROOF. We show that associativity implies alternativeness, whence by Corollary 9.2, the claim follows *a fortiori*. Associativity implies that for all $\underline{\mathbf{P}}$, $\underline{\mathbf{Q}}$, $\underline{\mathbf{R}} \in \underline{P}$, $\underline{\mathbf{P}} \cdot (\underline{\mathbf{Q}} \cdot \underline{\mathbf{R}}) = (\underline{\mathbf{P}} \cdot \underline{\mathbf{Q}}) \cdot \underline{\mathbf{R}}$. Left-alternativeness is obtained by taking $\underline{\mathbf{P}} = \underline{\mathbf{Q}}$ in the above formula, whereas right-alternativeness is obtained by taking $\underline{\mathbf{Q}} = \underline{\mathbf{R}}$. □

In addition to the various conditions related to alternativeness, linearizability is also ensured by properties related to the notion of *power-associativity*.

Definition 9.2. The bilinear multiplication $\cdot$ is *power-subassociative* if, for all $\underline{\mathbf{P}} \in \underline{P}$, and for all $m, n \geq 1$, there exists $k \geq 1$ such that

$$\underline{\mathbf{P}}^m \cdot \underline{\mathbf{P}}^n \subseteq \underline{\mathbf{P}}^k. \tag{9.10}$$

It is *power-associative* if $k = m + n$ and (9.10) holds with equality.

THEOREM 9.2. (9.1) *is left-linearizable if and only if $\cdot$ is power-subassociative.*

PROOF. Assume that $\cdot$ is power-subassociative. Using (8.6), we prove by induction on m, $m \geq 0$, that the following holds:

$$\underline{\mathbf{P}}_{m+1} \subseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k. \tag{9.11}$$

Basis. For $m = 0$, (9.11) is derived from $\underline{\mathbf{P}}_1 \subseteq \underline{\mathbf{Q}}$, which is a consequence of (8.6) with $\underline{\mathbf{P}}_0 = \underline{\varnothing}$.

Induction Step. Assume that the claim is true for some $m \geq 0$. We prove it for $m + 1$. From (8.6), we have that

$$\begin{aligned}
\underline{\mathbf{P}}_{m+2} &= \underline{\mathbf{P}}_{m+1} \cdot \underline{\mathbf{P}}_{m+1} \oplus \underline{\mathbf{Q}} \\
&\subseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k \cdot \sum_{l=1}^{\infty} \underline{\mathbf{Q}}^l \oplus \underline{\mathbf{Q}} && \text{Induction hypothesis and} \\
& && \text{monotonicity of } \cdot \\
&= \sum_{k=1,\, l=1}^{\infty} \underline{\mathbf{Q}}^k \cdot \underline{\mathbf{Q}}^l \oplus \underline{\mathbf{Q}} && \text{Bilinearity of } \cdot \\
&\subseteq \sum_{k \in K} \underline{\mathbf{Q}}^k \oplus \underline{\mathbf{Q}} \subseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k. && \text{Power-subassociativity of } \cdot
\end{aligned}$$

In the next to last expression, K denotes some subset of the natural numbers. Formula (9.11) implies that the solution $\underline{\mathbf{P}}^*$ of (9.1) satisfies $\underline{\mathbf{P}}^* \subseteq \Sigma_{k=1}^{\infty} \underline{\mathbf{Q}}^k$. Together with Lemma 9.1, this implies that the solution of (9.1) is given by

$$\underline{\mathbf{P}}^* = \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k. \tag{9.12}$$

The right-hand side of (9.12) represents the solution of (9.2). Hence, because of the power-subassociativity of $\cdot$, the bilinear (9.1) is left-linearizable.

For the other direction, assume that (9.1) is left-linearizable, which implies that (9.12) holds. Clearly, for all k, l, $\underline{\mathbf{Q}}^k \cdot \underline{\mathbf{Q}}^l \subseteq \underline{\mathbf{P}}^*$. Hence, from (9.12), we have that for all k, l, $\underline{\mathbf{Q}}^k \cdot \underline{\mathbf{Q}}^l \subseteq \Sigma_{m=1}^{\infty} \underline{\mathbf{Q}}^m$. The left-hand side of the above formula is a conjunctive query [15] and the right-hand side is a set of conjunctive queries. Applying the theorem of Sagiv and Yannakakis [46] yields that for all k, l, there exists some m such that $\underline{\mathbf{Q}}^k \cdot \underline{\mathbf{Q}}^l \subseteq \underline{\mathbf{Q}}^m$, that is, that $\cdot$ is power-subassociative. □

COROLLARY 9.4. *If $\cdot$ is power-associative, then* (9.1) *is both left- and right-linearizable.*

PROOF. Power-associativity implies power-subassociativity. Hence, by Theorem 9.2, (9.1) is left-linearizable. In addition, power-associativity implies that $\underline{\mathbf{Q}}^m \cdot \underline{\mathbf{Q}} = \underline{\mathbf{Q}} \cdot \underline{\mathbf{Q}}^m = \underline{\mathbf{Q}}^{m+1}$. Thus, when $\cdot$ is power-associative, (9.1) is right-linearizable as well. □

Example 9.2. Consider the following bilinear recursive Horn clause:

$$\mathbf{P}(x, z) \wedge \mathbf{P}(z, w) \wedge \mathbf{R}(y) \rightarrow \mathbf{P}(x, y).$$

Let $\cdot$ represent the function of the Horn clause. We show that, for all $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{S}$, $(\mathbf{P} \cdot \mathbf{Q}) \cdot \mathbf{S} \subseteq \mathbf{P} \cdot \mathbf{Q}$. The Horn clauses that correspond to the above algebraic formulas are

$$\mathbf{P}(x, z') \wedge \mathbf{Q}(z', w') \wedge \mathbf{R}(z) \wedge \mathbf{S}(z, w) \wedge \mathbf{R}(y) \rightarrow \mathbf{P}(x, y) \quad (9.13)$$

$$\mathbf{P}(x, z) \wedge \mathbf{Q}(z, w) \wedge \mathbf{R}(y) \rightarrow \mathbf{P}(x, y). \quad (9.14)$$

Clearly, viewed as conjunctive queries, (9.13) is contained in (9.14). Replacing $\mathbf{Q}$ and $\mathbf{S}$ with arbitrary powers of $\mathbf{P}$ yields $(\mathbf{P} \cdot \mathbf{P}^k) \cdot \mathbf{P}^l \subseteq \mathbf{P} \cdot \mathbf{P}^k$, or $\mathbf{P}^{k+1} \cdot \mathbf{P}^l \subseteq \mathbf{P}^{k+1}$, that is, $\cdot$ is power-subassociative. Theorem 9.2 guarantees that the given bilinear recursion is left-linearizable. It is easy to verify that $\cdot$ is not associative, so the above example establishes the usefulness of the condition of power-subassociativity over associativity. □

By Corollary 9.4, power-associativity implies that (9.1) is both left- and right-linearizable. To the contrary, power-subassociativity implies that (9.1) is left-linearizable only. Naturally, there is a condition similar to power-subassociativity that implies that (9.1) is right-linearizable. The condition is called *reverse power-subassociativity* and it is defined as follows: The multiplication $\cdot$ is reverse power-subassociative if, for all $\underline{\mathbf{P}} \in \underline{P}$, and for all m, $n \geq 1$, there exists $k \geq 1$, such that

$$\underline{\mathbf{P}}^m \cdot \underline{\mathbf{P}}^n \subseteq \Big(\cdots \big(\underbrace{(\underline{\mathbf{P}} \cdot \underline{\mathbf{P}}) \cdot \underline{\mathbf{P}}) \cdots}_{k} \big) \cdot \underline{\mathbf{P}}.$$

It is straightforward to see that reverse power-subassociativity is implied by power-associativity.

Recently, the result of Theorem 9.2 has been made tighter by Ramakrishnan et al. [41]. Before their result can be presented, some definitions and a lemma are necessary.

Definition 9.3. The bilinear multiplication $\cdot$ is *power-left-subalternative* if, for all $\underline{\mathbf{P}} \in \underline{P}$, and for all $l \geq 1$, there exists $m \geq 1$ such that

$$\underline{\mathbf{P}}^2 \cdot \underline{\mathbf{P}}^l \subseteq \underline{\mathbf{P}}^m. \quad (9.15)$$

Note that the form of (9.15) is a special case of both (9.10) and (9.3), which define power-subassociativity and left-subalternativeness respectively. The name of the property expressed in (9.15) is due to this observation.

LEMMA 9.2. *If $\cdot$ is power-left-subalternative, then for all $\underline{\mathbf{P}} \in \underline{P}$ and for all $n \geq 1$, $(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2)^n \subseteq \Sigma_{m=1}^{\infty} \underline{\mathbf{P}}^m$.*

PROOF. The proof is by induction on n.

Basis. For $n = 1$, the lemma is satisfied trivially, since $\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2$ is a finite sum of powers of $\underline{\mathbf{P}}$.

Induction Step. Assume that the lemma is true for some $n \geq 1$. We prove it for $n + 1$.

$$\begin{aligned}
\left(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2\right)^{n+1} &= \left(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2\right) \cdot \left(\underline{\mathbf{P}} \oplus \mathbf{P}^2\right)^n \\
&\subseteq \left(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2\right) \cdot \sum_{m=1}^{\infty} \underline{\mathbf{P}}^m && \text{Induction hypothesis and monotonicity of } \cdot \\
&= \sum_{m=1}^{\infty} \underline{\mathbf{P}}^{m+1} \oplus \sum_{m=1}^{\infty} \underline{\mathbf{P}}^2 \cdot \underline{\mathbf{P}}^m && \text{Bilinearity of } \cdot \\
&\subseteq \sum_{m=1}^{\infty} \underline{\mathbf{P}}^{m+1} \oplus \sum_{m \in K} \underline{\mathbf{P}}^m && \text{Power-left-subalternativeness of } \cdot \\
&\subseteq \sum_{\mathrm{m}=1}^{\infty} \underline{\mathbf{P}}^{\mathrm{m}}
\end{aligned}$$

In the above, K is some set of natural numbers. This concludes the proof of the lemma. □

We now proceed in proving the theorem of Ramakrishnan et al. [41] in the algebraic framework of this paper. Although the two proofs are different, they essentially use the same techniques.

THEOREM 9.3 [41]. *The bilinear multiplication $\cdot$ is power-left-subalternative if and only if it is power-subassociative.*

PROOF. Clearly, if $\cdot$ is power-subassociative, then it is power-left-subalternative as well. For the other direction, assume that $\cdot$ is power-left-subalternative. We prove that, for all $\underline{\mathbf{P}} \in \underline{P}$ and for all $k, l \geq 1$, there exists $m \geq 1$ such that $\underline{\mathbf{P}}^k \cdot \underline{\mathbf{P}}^l \subseteq \underline{\mathbf{P}}^m$. The proof is by induction on k.

Basis. For $k = 1$, the above claim is satisfied trivially, since $\underline{\mathbf{P}} \cdot \underline{\mathbf{P}}^l = \underline{\mathbf{P}}^{l+1}$.

Induction Step. Assume that the claim is true for some $k \geq 1$. We prove it for $k + 1$. Let $\underline{\mathbf{Q}}$ be defined as $\underline{\mathbf{Q}} = \underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2$. Clearly, for all $k \geq 1$, the following hold:

$$\underline{\mathbf{P}}^k \subseteq \left(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2\right)^k = \underline{\mathbf{Q}}^k,$$
$$\underline{\mathbf{P}}^{k+1} \subseteq \left(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2\right)^k = \underline{\mathbf{Q}}^k.$$

Hence, we have that

$$\begin{aligned}
\underline{\mathbf{P}}^{k+1} \cdot \underline{\mathbf{P}}^l &\subseteq \underline{\mathbf{Q}}^k \cdot \underline{\mathbf{Q}}^l && \text{Monotonicity of } \cdot \\
&\subseteq \underline{\mathbf{Q}}^n = \left(\underline{\mathbf{P}} \oplus \underline{\mathbf{P}}^2\right)^n, \quad \text{for some } n \geq 1 && \text{Induction hypothesis} \\
&\subseteq \sum_{m=1}^{\infty} \underline{\mathbf{P}}^m. && \text{Lemma 9.2}
\end{aligned}$$

Thus, we have shown that $\underline{\mathbf{P}}^k \cdot \underline{\mathbf{P}}^l \subseteq \Sigma_{m=1}^{\infty} \underline{\mathbf{P}}^m$. Since the left-hand side of the above formula is a conjunctive query and its right-hand side is a set of conjunctive queries, we can again use the result of Sagiv and Yannakakis [46] to yield $\underline{\mathbf{P}}^k \cdot \underline{\mathbf{P}}^l \subseteq \underline{\mathbf{P}}^m$, for some $m \geq 1$. This concludes the induction step, and the proof of the theorem is complete. □

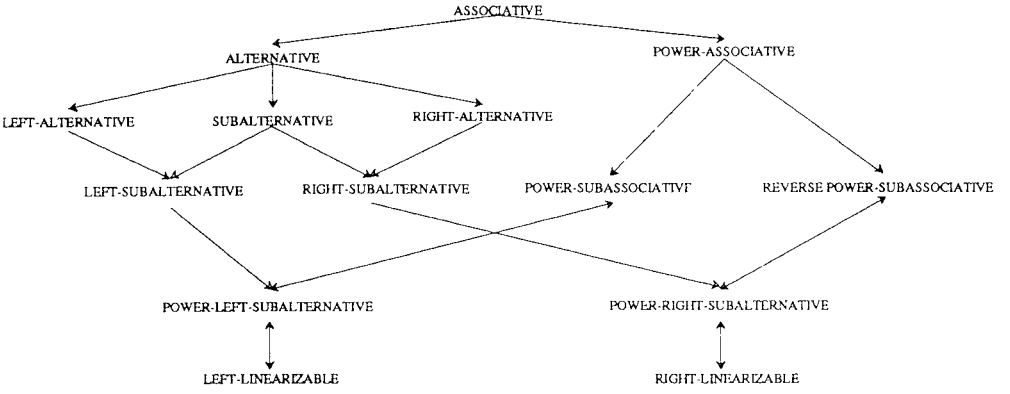

FIG. 2. Relationship of properties of E_P implying linearizability

Power-right-subalternativeness is defined symmetrically to Definition 9.3, and results similar to those of Lemma 9.2 and Theorem 9.3 can be derived.

The sufficient and necessary-and-sufficient conditions for linearizability that have been presented in this section are summarized in Figure 2. There, an arrow from property x to property y indicates that property x implies property y. Some properties are linked with bidirectional arrows, signifying that they are equivalent. An interesting by-product of this study is that, in E_P, left-subalternativeness implies power-subassociativity. We do not expect this to hold in general, that is, for all nonassociative closed semirings. In that respect, nonassociative algebras are different: for all such structures, alternativeness always implies power-associativity [48].

10. *Embedding E_P in an Algebra*

Unfortunately, all properties that are equivalent to left-linearizability, that is, power-subassociativity and power-left-subalternativeness, require testing for containment of recursive programs, which is in general undecidable [50]. One could consider *uniform containment* [45] of the programs involved in these conditions and obtain decidable sufficient conditions for linearizability [41], but the required tests are still expensive. In fact, among the conditions in Figure 2, only those that are at least as strong as left-alternativeness can be easily tested, in the sense of only requiring testing for equivalence of conjunctive queries. In this section, we derive another sufficient condition for left-linearizability, which is related to power-associativity and requires testing for equivalence of nonrecursive programs as well. This is done by embedding E_P in an algebra.

In general, given a system E_S of a given algebraic structure A, one can embed it in a richer algebraic structure B by extending the set of properties of its elements and/or its operations, so that the requirements of B are satisfied. By appropriate mappings of the properties of B to the properties of A, theorems that are derived within B can be used to derive additional ones that hold within A. Schematically, this is shown in Figure 3, where T_A and T_B denote theorems that hold within A and B, respectively. System E_S is embedded in B (in the opposite direction of the arc $B \rightarrow A$), T_B is derived within the embedded system, and T_A is derived by reversing the mappings used in the embedding. The final result is a theorem within A.

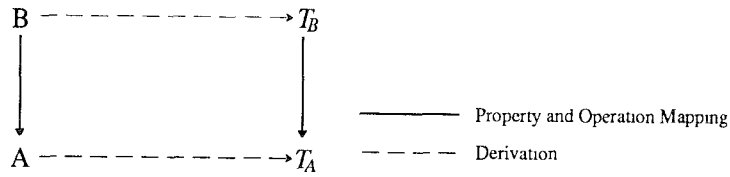

FIG. 3. Schematic representation of embedding.

In our specific case, the system of interest is $E_{\underline{P}}$, A is a closed semiring, B is an algebra, T_A is Theorem 10.2, and T_B is a known theorem in power-associative algebras [48]. The main effect of embedding $E_{\underline{P}}$ in an algebra is that relations are treated as multisets instead of sets. Relations may contain duplicate tuples, and each tuple is associated with a number that indicates the number of occurrences of the tuple in the relation. In fact, the notion of "occurrence of a tuple in a relation" becomes fuzzy, since the number associated with a tuple in a relation can be an arbitrary real number, although allowing nonintegers is only a technicality so that the embedding is realized. If one needed to evaluate programs within the algebra, duplicates would have to be retained, and the processing cost would most likely be prohibitive. As we shall see in this section, however, we only use the embedding to derive conditions for left-linearizability. If a program does satisfy these conditions, its equivalent linear form can be executed within the closed semiring of linear operators with no need to retain duplicates.

We now proceed with the embedding. Recall that $2^{C_D^{a_i}}$, $a_i \geq 1$, $1 \leq i \leq n$, denotes the collection of all relations having arity a_i, and that $\underline{P}$ is defined as

$$\underline{P} = 2C_D^{a_1} \times \cdots \times 2C_D^{a_n}.$$

Also, define

$$\underline{C}_D = C_D^{a_1} \times \cdots \times C_D^{a_n}.$$

Consider a relation $\mathbf{Q} \subseteq C_D^{a_i}$. As such, $\mathbf{Q}$ can be viewed as a function $\mathbf{Q}$: $C_D^{a_i} \to \{0, 1\}$, defined by

$$\mathbf{Q}(t) = \begin{cases} 1 & \text{if the tuple } t \text{ is in } \mathbf{Q}, \\ 0 & \text{otherwise} \end{cases}.$$

Based on that, $\underline{P}$ corresponds to the set $\underline{P} = \{\text{all functions } \rho: \underline{C}_D \to \{0, 1\}^n\}$. We extend $\underline{P}$ to the set $\underline{P}^r$ of all functions mapping $\underline{C}_D$ into $\mathbb{R}^n$, where $\mathbb{R}$ is the set of real numbers. That is, $\underline{P}^r = \{\text{all functions } \rho: \underline{C}_D \to \mathbb{R}^n\}$,[7] which is well known to be a vector space over the field of reals $\mathbb{R}$ [26]. Each member of $\underline{P}^r$ is called an *extended relation vector*. Clearly, every member of $\underline{P}$ is also a member of $\underline{P}^r$. The appropriate addition in $\underline{P}^r$ is the ordinary addition of real numbers: For all $\underline{t} \in \underline{C}_D$, $(\underline{\mathbf{P}}^r + \underline{\mathbf{Q}}^r)(\underline{t}) = \underline{\mathbf{P}}^r(\underline{t}) + \underline{\mathbf{Q}}^r(\underline{t})$. (We use + with the understanding that it is not to be confused with + as defined in Section 3 for addition of linear relational operators.)

Consider a bilinear function g: $\underline{P} \times \underline{P} \to \underline{P}$, which corresponds to the multiplication $\cdot$. Define its extension g^r: $\underline{P}^r \times \underline{P}^r \to \underline{P}^r$, which corresponds

[7]The superscript r is for *r*eals. It will be used to tag elements, operations, and functions of $\underline{P}^r$.

to the multiplication $\cdot^r$, as follows:

$$\underline{\mathbf{P}}^r \cdot^r \underline{\mathbf{Q}}^r = \sum_{\underline{s},\underline{t} \in \underline{C}_D} \underline{\mathbf{P}}^r(\underline{s})\underline{\mathbf{Q}}^r(\underline{t})(1_{\underline{s}} \cdot 1_{\underline{t}}),$$

where $1_{\underline{t}}$ is the indicator function

$$1_{\underline{t}}(\underline{s}) = \begin{cases} 1 & \text{if } \underline{s} = \underline{t}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that the summation is with respect to ordinary addition, that is, the addition in $\underline{P}^r$. An intuitive meaning of the definition of $\cdot^r$ is that it operates on (extended) relations one tuple at a time and retains duplicates in the result. It is easy to show that $\cdot^r$ is bilinear. In fact, this is the case whether $\cdot$ is bilinear or not.

For any member $\underline{\mathbf{P}}^r$ of $\underline{P}^r$, define the function $|\underline{\mathbf{P}}^r| \in \underline{P}$ as follows:

$$|\underline{\mathbf{P}}^r|(\underline{t}) = \begin{cases} 1 & \text{if } \underline{\mathbf{P}}^r(\underline{t}) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, when $|\ \ |$ is applied on an extended relation vector, it removes the duplicates. For the following, recall that any element of $\underline{P}$ is an element of $\underline{P}^r$ as well.

LEMMA 10.1. *If* $\underline{\mathbf{P}}, \underline{\mathbf{Q}} \in \underline{P}$, *then* $|\underline{\mathbf{P}} \cdot^r \underline{\mathbf{Q}}| = \underline{\mathbf{P}} \cdot \underline{\mathbf{Q}}$.

PROOF. This is a direct consequence of the definitions of $\cdot$ and $\cdot^r$. □

THEOREM 10.1. *The system* $(\underline{P}^r, +, \cdot^r, \underline{\varnothing}, \mathbb{R})$ *is a nonassociative algebra without identity over the field* $\mathbb{R}$.

PROOF. We have already mentioned that the system $(\underline{P}^r, +, \underline{\varnothing}, \mathbb{R})$ is a vector space over the field of reals $\mathbb{R}$. It is also straightforward to verify that $\underline{P}^r$ is closed under multiplication, $\cdot^r$ distributes over $+$, and that multiplying any product of two elements of $\underline{P}^r$ with a real number is equivalent to multiplying one of the elements and then taking the product. Based on Definition 2.4, the above imply the theorem. □

The multiplications $\cdot$ and $\cdot^r$ define powers on $\underline{P}$ and $\underline{P}^r$, respectively, as follows:

$$\underline{\mathbf{P}}^1 = \underline{\mathbf{P}}, \underline{\mathbf{P}}^k = \underline{\mathbf{P}} \cdot \underline{\mathbf{P}}^{k-1}, \underline{\mathbf{P}} \in \underline{P},$$
$$(\underline{\mathbf{P}}^r)^{(1)} = \underline{\mathbf{P}}^r, (\underline{\mathbf{P}}^r)^{(k)} = \underline{\mathbf{P}}^r \cdot^r (\underline{\mathbf{P}}^r)^{(k-1)}, \underline{\mathbf{P}}^r \in \underline{P}^r.$$

The concept of power-associativity can now be extended to $\cdot^r$.

Definition 10.1. The bilinear multiplication $\cdot^r$ on $\underline{P}^r \times \underline{P}^r$ is *power-associative* if

$$(\underline{\mathbf{P}}^r)^{(m)} \cdot^r (\underline{\mathbf{P}}^r)^{(n)} = (\underline{\mathbf{P}}^r)^{(m+n)}.$$

LEMMA 10.2. *The bilinear multiplication* $\cdot$ *on* $\underline{P} \times \underline{P}$ *is power-associative, if its extension* $\cdot^r$ *on* $\underline{P}^r \times \underline{P}^r$ *is power-associative.*

PROOF. Suppose $\cdot^r$ is power-associative. Then, for $\underline{\mathbf{P}} \in \underline{P}$, it is

$$\underline{\mathbf{P}}^m \cdot \underline{\mathbf{P}}^n = |\underline{\mathbf{P}}^{(m)}| \cdot |\underline{\mathbf{P}}^{(n)}| = \|\underline{\mathbf{P}}^{(m)}| \cdot^r |\underline{\mathbf{P}}^{(n)}\|$$
$$= |\underline{\mathbf{P}}^{(m)} \cdot^r \underline{\mathbf{P}}^{(n)}| = |\underline{\mathbf{P}}^{(m+n)}| = \underline{\mathbf{P}}^{m+n}.$$

The first, second, and fifth equalities are derived from Lemma 10.1. The third one is a consequence of the fact that $\underline{\mathbf{P}} \in \underline{P}$, and the fourth one is due to the power-associativity of $\cdot^r$. □

THEOREM 10.2. *If for all* $\underline{\mathbf{P}}^r \in \underline{P}^r$

$$(\underline{\mathbf{P}}^r)^{(2)} \cdot^r \underline{\mathbf{P}}^r = (\underline{\mathbf{P}}^r)^{(3)} \quad \text{and} \quad (\underline{\mathbf{P}}^r)^{(2)} \cdot (\underline{\mathbf{P}}^r)^{(2)} = (\underline{\mathbf{P}}^r)^{(4)}, \quad (10.4)$$

then $\cdot$ *is left- and right-linearizable.*

PROOF. It is known that, within an algebra over a field with characteristic 0 (like $\mathbb{R}$), (10.4) is a necessary and sufficient condition for a bilinear multiplication $\cdot^r$ to be power-associative [48]. By Lemma 10.2, this further implies that $\cdot$ is power-associative. Hence, by Corollary 9.4, (10.4) implies that $\cdot$ is both left- and right-linearizable. □

Theorem 10.2 provides a sufficient condition for left-linearizability. The only disadvantage is that the condition requires testing for equivalence of conjunctive queries within the embedded system, that is, taking into account duplicate retention. The classical theorem by Chandra and Merlin does not hold, because it treats relations as sets and not multisets [15]. We are not aware of any decision procedure for this type of equivalence. In general, there is almost no theory on the properties of queries and programs that retain duplicates. The development of such a theory is part of our future plans.

11. *Comparison to the Logic-Based Approach*

As we mentioned in the introduction, the great majority of the work on recursion has been based on a first-order logic representation of recursive programs, Horn clauses in particular. In this section, we give a brief comparison of the algebraic approach developed in this paper with the traditional logic-based approach. Clearly, by Proposition 4.1, every linear Horn clause can be represented by a linear operator in R and vice versa. Similarly, every multilinear program can be expressed as a bilinear multiplication of two relation vectors and vice versa. Thus, the two approaches are equivalent in terms of expressive power. Their difference is in the ease with which certain properties are expressed. We claim that certain properties are fundamentally algebraic in nature and they can be studied more naturally in the algebraic framework, whereas others are logic-based in nature and they can be studied more naturally in the logic-based framework. We do not attempt here to define precisely which properties are algebraic in nature and which are not, since this may enter the realms of philosophy. Instead, we compare the results presented in this paper with similar ones that have been derived within the logic-based framework, if such results exist. We also describe some known results in the logic approach that do not seem to lend themselves naturally in the algebraic approach.

11.1. STRENGTHS OF THE ALGEBRAIC APPROACH. The fundamental advantage of the algebraic over the logic-based approach is the ability of the former to explicitly represent query answers of recursive programs (especially in the linear case), which can then be manipulated algebraicly. In the previous sections, we have presented several theorems that can be derived by taking advantage of this ability. Although some of these results have been derived based on logic as well, we feel that their algebraic derivations are more natural.

In Section 7, we have seen how the algebraic framework can be used to express several algorithms for recursive query processing at the ordering stage of query optimization (Figure 1). Some of the algorithms have been proposed under the logic-based framework as well (e.g., Henschen–Naqvi), whereas others have not (e.g., Minimal). By the very nature of ordering stage, the algebra seems to be the only appropriate tool to explore the complete variety of processing algorithms. Thus, we do not discuss the ordering stage in any more detail, but we concentrate on the rewriting stage, where the algebraic and logic-based approaches seem to complement each other.

11.1.1. *Linear Recursion.* Section 6 describes several results that can be used at the rewriting stage of an optimizer. Theorem 6.1 on the decomposition of $(B + C)^*$ has been observed by Ramakrishana et al. as well, who in essence used the algebraic notation developed in this paper [41]. They actually used regular expressions over rule names, but the equivalence to the algebra is straightforward, since there is a one-to-one correspondence between rules and operators, and regular expressions form a closed semiring [2]. Theorems 6.2 and 6.3 are new and similar in nature with Theorem 6.1. Other results of the same nature have been obtained by Lassez and Maher [33, 35] and by Dong [19]. Theorems 6.4 and 6.5 on necessary conditions for decompositions of $(B + C)^*$ make use of the theorem of Sagiv–Yannakakis [46], which is logic-based in nature.

Theorem 6.6 on the decomposition of $(BC)^*$ is new, but its corollary (Corollary 6.2) has been used by many researchers, although not with any explicit reference to the algebra behind it. An example of such a use is in the performance study of Han and Lu [24], where the algorithms of Henschen–Naqvi, Shapiro–McKay, and Han–Lu were expressed in the way that was shown in Section 7. Theorem 6.7 is also new and captures part of the essence of the study of Naughton on recursively redundant predicates [37].

Theorem 6.8 allows the interchange of the two linear forms of transitive closure. This together with the fact that the bilinear version of the transitive closure program is easily proven to be associative, and therefore linearizable (Corollary 9.3), result in the ability to modify any form of the transitive closure program-query pair into its most efficient form, thus capturing all possible such transformations [10]. Theorem 6.9 can be used to reduce the arity of recursive predicates, which often has significant effects on performance [9, 39].

Finally, Theorem 6.10 and Corollary 6.3 provide the foundation for selection and projection pushing. Transformations like those of Theorem 6.10 on selection pushing were among the first proposed for recursive programs in database systems [3, 32]. In the logic-based approach, projection pushing has been studied by Ramakrishnan et al. [40], who derived several syntactic characterizations to capture the conditions of Theorem 6.10 for applying projections early and Corollary 6.3 for elimination of (recursive) clauses. We have presented several examples that demonstrated the applicability of the algebraic approach in such issues, by producing the same results as the logic-based approach.

As a final comment on the merits of the algebraic approach in the study of linear recursion, we want to mention that several other researchers have employed it in their work, although not explicitly. We have already mentioned the work on commutativity by Ramakrishnan et al. [41], and the performance

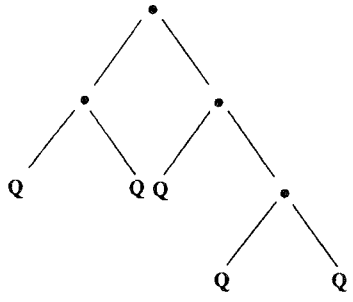

FIG. 4. Tree representation of $\mathbf{Q}^2 \cdot \mathbf{Q}^3$.

evaluation by Han and Lu [24]. We would also like to mention the work on *magic functions* by Gardarin and Maindreville [22] and Gardarin [23], where Horn clauses are viewed as functions and manipulated appropriately (precisely in the way operators are), and the work on parallel algorithms for transitive closure by Valduriez and Khoshafian [53], where they develop algorithms based on the fact that, for two linear operators A, $B \neq 0$, if $A^* = A$ and $B^* = B$, then for all $k \geq 0$, $A^k \leq A$ and $B^k \leq B$, which further implies that $(A + B)^* = (1 + A)(BA)^* + (1 + B)(AB)^*$.

11.1.2. *Bilinear/Multilinear Recursion.* The algebraic approach for bilinear recursion can have again several applications at the ordering stage of query optimization. No such results have been reported in this paper, however, since they are either obvious or straightforward extension of results for the linear case.

For the rewriting stage, the algebraic approach has provided several results on the problem of linearizability in Sections 9 and 10. The results in the latter rely on embedding relation vectors in an algebra and on known algebraic properties of such structures. The logic-based approach lacks the tools that would enable similar discoveries. From the results in Section 9, Theorem 9.3 is the most general of all and is part of the work by Ramakrishnan et al. [41]. Their effort was based on proof trees and their transformations, but essentially, a proof tree is another representation of a product in a nonassociative closed semiring (the only difference being that the leaves of a tree are individual tuples, whereas the factors in an algebraic product are set of tuples, i.e., relations). An example of the correspondence is shown in Figure 4.

The earliest work that we are aware of on the problem of linearizability is that by Zhang and Yu [56]. They focus their attention to a restricted class of bilinear Horn clauses, that is, a restricted class of multiplications $\cdot$, and for that class they provide a necessary and sufficient syntactic condition for left linearizability. It is rather straightforward to prove (and we do not do it in this paper) that if a Horn clause satisfies their syntactic condition, then the corresponding multiplication $\cdot$ is associative, so by Corollary 9.3 it is left- and right-linearizable. Thus, for the restricted class of multiplications they examined, all conditions studied in Section 9 together with the condition by Zhang and Yu are mutually equivalent. The algebraic approach provides tools to study these conditions, whereas the logic-based approach, which was employed by Zhang and Yu, provides syntactic characterizations of them. More recent results on syntactic conditions for linearizability of restricted classes of bilinear

Horn clauses have been given by Zhang et al. [57] and Saraiya [47]. Both these conditions are generalized by Theorem 9.3 [41].

11.2. Limitation of the Algebraic Approach. The algebraic approach is limited to only addressing problems at the rewriting and ordering stages of query processing and optimization. The abstraction it offers is at a higher level than is necessary for studying the details of the planning level. In addition, even at the rewriting stage, certain transformations seem to have a nonalgebraic flavor, for example, magic sets and generalized counting [7, 11], and factoring [39] cannot be derived algebraicly. It is hard to identify exactly what makes a property algebraic and what not. In that sense, we do not know which characteristics of the above transformations make them unnatural to express algebraicly. One problem seems to be notational, since the above transformations tend to produce multilinear programs from linear ones, and the appropriate algebraic structures for the two are different. Another problem seems to be that these transformations tend to modify the structure of the original operators/clauses. Expressing such transformations algebraicly requires specifying much detail about the operators, for example, their parameter relations and the specific manipulations of their columns. Doing that would simply be a change in notation from logic to algebra, which seems pointless, since the latter, when expressing all the required details, offers no advantages over the former.

Negation was excluded from the algebras developed in this paper, since we only deal with Horn clauses. Hence, the current study has very little to offer in the study of programs that include it. It is conceivable that more powerful algebras will be able to capture negation and offer insights into its properties. Embedding relation vectors in an algebra (Section 10) may be a good starting point for developing such an algebraic structure, but this requires further investigation.

Finally, as we have already mentioned, the algebra does not offer any tools that can lead to deriving syntactic characterizations of interesting properties of Horn clauses programs (even algebraic properties). To be more precise, the algebra should take into account the details of the manipulations of the columns of the relations in the operators in order to become a usable tool for such work. This offers no advantage over the logic-based approach. Hence, syntactic characterizations of such properties are likely to be based on the logic form of the operators. As examples of such work, we offer characterizations of the properties of bounded recursion [28, 38], linearizability [47, 56, 57], commutativity of selections with arbitrary operators [1], and commutativity of arbitrary operators [29].

12. *Conclusions*

A significant subset of all linear relational operators have been embedded into a closed semiring. Within this algebraic structure, processing recursive Horn clauses has been reduced to solving recursive equations. For a single linear Horn clause, the solution to the corresponding operator equation is equal to the transitive closure of the operator representing the Horn clause. This approach can be extended to multiple Horn clauses that are linearly mutual recursive. In that case, inference is reduced to solving linear systems of operator equations, in the same manner that immediate recursion is reduced to solving a single such

equation. The ability to algebraicly manipulate an operator representing the query answer has important implications. We have presented several specialized transformations of the query answer at the rewriting stage of query optimization, which when applicable, have the potential to speed up the process of answering the query. We have also described several general and specialized transformations of the query answer at the ordering stage.

Nonlinear recursion has also been treated similarly by embedding all bilinear recursions into a nonassociative closed semiring. The universality of the approach has been demonstrated by showing that any nonlinear recursion can be reduced to a bilinear one. We have given several conditions for a bilinear recursion to be left-linearizable. All conditions are variations of two properties, namely, alternativeness and power-associativity. Most of these conditions require testing for equivalence or containment of recursive programs, which is computationally undesirable. By embedding all bilinear recursions into an algebra, we have been able to derive a simple sufficient condition for linearizability that requires testing for equivalence of nonrecursive programs only.

ACKNOWLEDGMENTS. We would like to thank Jeff Ullman for many useful discussions, as well as the anonymous referees whose comments had a significant impact on the readability of the paper.

REFERENCES

1. AGRAWAL, R., AND DEVANBU, P. Moving selections into linear least fixpoint queries. *IEEE Trans. Knowl. Data Eng 1*, 4 (Dec. 1989), 424–432.
2. AHO, A., HOPCROFT, J., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
3. AHO, A., AND ULLMAN, J. Universality of data retrieval languages. In *Proceeding of the 6th ACM Symposium on Principles of Programming Languages* (San Antonio, Tex., Jan.). ACM, New York, 1979, pp. 110–117.
4. APT, K. R., AND VANEMDEN, M. H. Contributions to the theory of logic programming. *JACM 29*, 3 (July 1982), 841–862.
5. BACKHOUSE, R. C., AND CARRE, B. A. Regular algebra applied to path-finding problems. *J. Inst. Math. Appl. 15*, 2 (Apr. 1975), 161–186.
6. BANCILHON, F. Noise evaluation of recursively defined relations. In M. Brodie and J. Mylopoulos, eds., *On Knowledge Base Management: Integrated Artificial Intelligence and Database Technologies*. Springer-Verlag, New York, 1986.
7. BANCILHON, F., MAIER, D., SAGIV, Y., AND ULLMAN, J. D. Magic sets and other strange ways to implement logic programs. In *Proceedings of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database System* (Cambridge, Mass., Mar.). ACM, New York, 1986, pp. 1–15.
8. BANCILHON, F., AND RAMAKRISHNAN, R. An amateur's introduction to recursive query processing strategies. In *Proceedings of the 1986 ACM-SIGMOD Conference on the Management of Data* (Washington, D.C., May). ACM, New York, 1986, pp. 16–52.
9. BANCILHON, F. AND RAMAKRISHNAN, R. Performance evaluation of data intensive logic programs. In J. Minker, ed., *Foundations of Deductive Databases and Logic Programming*, Morgan-Kaufmann, San Mateo, Calif., 1988, pp. 439–517.
10. BEERI, C., KANELLAKIS P., BANCILHON, F., AND RAMAKRISHNAN, R. Bounds on the propagation of selection in logic programs. In *Proceedings of the 6th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (San Diego, Calif., Mar.). ACM, New York, 1987, pp. 214–226.
11. BEERI, C., AND RAMAKRISHNAN, R. On the power of magic. In *Proceedings of the 6th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (San Diego, Calif., Mar.). ACM, New York, 1987, pp. 269–283.
12. CARRE, B., *Graphs and Networks*, Oxford University Press, Oxford, England, 1979.
13. CERI, S., GOTTLOB, G., AND LAVAZZA, L. Translation and optimization of logic queries: The algebraic approach. In *Proceedings of the 12th International VLDB Conference* (Kyoto, Japan, Aug.). Morgan Kaufmann, San Mateo, Calif., 1986, pp. 395–402.

14. CERI, S., AND TANCA, L. Optimization of systems of algebraic equations for evaluating datalog queries. In *Proceedings 13th International VLDB Conference* (Brighton, England, Sept). Morgan Kaufmann, San Mateo, Calif., 1987, pp. 31–41
15 CHANDRA, A. K., AND MERLIN, P M. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th Annual ACM Symposium of Theory of Computing* (Boulder, Colo , May). ACM, New York, 1977, pp. 77–90.
16. CLOCKSIN, W. F., AND MELLISH, C. S. *Programming in Prolog.* Springer-Verlag, New York, 1981
17. CODD, E F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (1970), 377–387.
18. COSMADAKIS, S., AND KANELLAKIS, P. Parallel evaluation of recursive rule queries. In *Proceedings of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems* (Cambridge, Mass, Mar.) ACM, New York, 1986, pp. 280–293
19. DONG, G. *On the Composition of Datalog Program Mappings.* Unpublished Manuscript, Univ. Southern California, Nov., 1988
20. EILENBERG, S. *Automata, Languages, and Machines.* Acedamic Press, New York, 1974.
21. GAIFMAN, H , MAIRSON, H, SAGIV, Y., AND VARDI, M Undecidable optimization problems for database logic programs. In *Proceedings of the 2nd ACM Symposium on Logic in Computer Science* (Ithaca, NY, June). ACM, New York, 1987, pp. 106–115
22. GARDARIN, G , AND DE MAINDREVILLE, C. Evaluation of database recursive logic programs as recurrent function series In *Proceedings of the 1986 ACM-SIGMOD Conference of the Management of Data* (Washington, D.C., May). ACM, New York, 1986, pp. 177–186.
23. GARDARIN, G. Magic functions: A technique to optimize extended datalog recursive programs. In *Proceedings of the 13th International VLDB Conference* (Brighton, England, Sept.). Morgan Kaufmann, San Mateo, Calif., 1987, pp. 21–30.
24. HAN, J , AND LU, H Some performance results on recursive query processing in relational database systems In *Proceedings of the 2nd International Conference on Data Engineering* (Los Angeles, Calif., Jan.). IEEE Computer Society Press, Washington, D C , 1986, pp. 533–539
25 HENSCHEN, L., AND NAQVI, S. On compiling queries in recursive first-order databases *JACM 31*, 1 (Jan. 1984), 47–85
26. HERSTEIN, I N. *Topics in Algebra.* Wiley, New York, N Y., 1975.
27. IOANNIDIS, Y. E. On the computation of the transitive closure of relational operators. In *Proceedings of the 12th International VLDB Conference* (Kyoto, Japan, Aug). Morgan Kaufmann, San Mateo, Calif., 1986, pp. 403–411
28. IOANNIDIS, Y. E. A time bound on the materialization of some recursively defined views. *Algorithmica 1*, 4 (Oct. 1986), 361–385.
29 IOANNIDIS, Y.E. Commutativity and its role in the processing of linear recursion. *J. Logic Prog.*, to appear.
30 IOANNIDIS, Y. E , AND WONG, E An algebraic approach to recursive inference. In *Proceedings of the 1st International Conference on Expert Database Systems* (Charleston, S.C., Apr.). Benjamin/Cummings, Redwood City, Calif., 1987, pp. 295–309.
31. IOANNIDIS, Y. E., AND WONG, E. Query optimization by simulated annealing. In *Proceedings of the 1987 ACM-SIGMOD Conference on the Management of Data* (San Francisco, Calif., May). ACM, New York, 1987, pp. 9–22
32. KIFER, M., AND LOZINSKII, E. On compile time query optimization in deductive databases by means of static filtering *ACM Trans. Datab. Syst. 15*, 3 (Sept. 1990), 385–426.
33. LASSEZ, J L. AND MAHER, M. J. Closures and fairness in the semantics of programming logic *Theoret. Comput. Sci. 29* (1984), 167–184.
34. LEHMANN, D J Algebraic structures for transitive closure. *Theoret. Comput. Sci. 4* (1977), 59–76.
35. MAHER, M. J Semantics of Logic Program. Ph D dissertation. Dep. Comput. Sci., Univ Melbourne, Parkville, Australia (also available as Tech. Rep No. M85/14), 1985.
36. NAUGHTON, J. Compiling separable recursions. In *Proceedings of the 1988 ACM-SIGMOD Conference on the Management of Data* (Chicago, Ill , June). ACM, New York, 1988, pp 312–319.
37. NAUGHTON, J. Minimizing function-free recursive inference rules. *JACM 36*, 1 (Jan. 1989), 69–91

38. NAUGHTON, J. Data independent recursion in deductive databases. *J. Comput. Syst. Sci. 38*, 2 (Apr. 1989), 259–289.
39. NAUGHTON, J. F., RAMAKRISHNAN, R., SAGIV, Y., AND ULLMAN, J. D. Factoring can reduce arguments. In *Proceedings of the 15th International VLDB Conference* (Amsterdam, The Netherlands, Aug.). Morgan Kaufmann, San Mateo, Calif., 1989, pp. 173–182
40. RAMAKRISHNAN, R., BEERI, C., AND KRISHNAMURTHY, R. Optimizing existential datalog queries. In *Proceedings of the 7th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Austin, Tex., Mar.). ACM, New York, 1988, pp. 89–102.
41. RAMAKRISHNAN, R., SAGIV, Y , ULLMAN, J. D., AND VARDI, M. Proof tree transformation theorems and their applications *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Philadelphia, Pa., Mar.). ACM, New York, 1989, pp. 172–181.
42. ROSENTHAL, A., HEILER, S., DAYAL, U., AND MANOLA, F. Traversal recursion: A practical approach to supporting recursive applications. In *Proceedings of the 1986 ACM-SIGMOD Conference on the Management of Data* (Washington, DC, May). ACM, New York, 1986, pp. 166–176.
43. SACCA, D., AND ZANIOLO, C. On the implementation of a simple class of logic queries for databases. In *Proceedings of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems* (Cambridge, Mass., Mar.) ACM, New York, 1986, pp. 16–23.
44. SACCA, D., AND ZANIOLO, C. The generalized counting method for recursive logic queries. In *Proceedings of the International Conference on Database Theory* (Rome, Italy, Oct.). Springer-Verlag, Berlin, 1986, pp. 31–53.
45. SAGIV, Y Optimizing datalog programs. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System* (San Diego, Calif., Mar.). ACM, New York, 1987, pp. 349–362.
46. SAGIV, Y., AND YANNAKAKIS, M. Equivalences among relational expressions with the union and difference operator. *JACM 27*, 4 (Oct. 1980), pp. 633–655.
47. SARAIYA, Y. P. Linearizing nonlinear recursions in polynomial time In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Philadelphia, Pa., Mar.). ACM, New York, 1989, pp. 182–189.
48. SCHAFER, R. D. *An Introduction to Nonassociative Algebras*. Academic Press, Orlando, Fla., 1966.
49. SHAPIRO, S., AND MCKAY, D. Inference with recursive rules. In *Proceedings of the 1st Annual National Conference on Artificial Intelligence* (Palo Alto, Calif., Aug.). 1980, pp. 151–153.
50. SHMUELI, O. Decidability and expressiveness aspects of logic queries. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (San Diego, Calif., Mar). ACM, New York, 1987, pp. 237–249.
51. TARSKI, A. A lattice theoretical fixpoint theorem and its applications. *Pacific J. Math. 5* (1955), pp. 285–309.
52. VALDURIEZ, P. AND BORAL, H. Evaluation of recursive queries using join indices. In *Proceedings of the 1st International Conference on Expert Database Systems* (Charleston, S.C., Apr.). Benjamin/Cummings, Redwood City, Calif., 1987, pp. 271–293.
53. VALDURIEZ, P., AND KHOSHAFIAN, S. Transitive closure of transitively closed relations. In *Proceedings of the 2nd International Conference on Expert Database Systems* (Tysons Corner, Va., Apr.). Benjamin/Cummings, Redwood City, Calif., 1989, pp. 377–400.
54. M. H. VANEMDEN, AND KOWALSKI, R. A. The sematics of predicate logic as a programming language. *JACM 23*, 4 (Oct. 1976). 733–742.
55. VIEILLE, L. Recursive axioms in deductive databases: The query/subquery approach. In *Proceedings of the 1st International Conference on Expert Database Systems* (Charleston, S.C., Apr.) Benjamin/Cummings, Redwood City, Calif., 1987, pp. 253–267.
56. ZHANG, W., AND YU, C. T. A necessary condition for a doubly recursive rule to be equivalent to a linear recursive rule. In *Proceedings of the 1987 ACM-SIGMOD Conference on the Management of Data* (San Francisco, Calif., May). ACM, New York, 1987, pp. 345–356.
57. ZHANG, W., YU, C. T., AND TROY, D. A necessary and sufficient condition to linearize doubly recursive programs in logic databases. *ACM Trans. Datab. Syst. 15*, 3 (Sept. 1990), 459–482.

RECEIVED NOVEMBER 1988; REVISED JULY 1989 AND FEBRUARY 1990; ACCEPTED MARCH 1990