TOWARDS AN ALGEBRAIC THEORY OF RECURSION

by

Yannis E. Ioannidis
Eugene Wong

Computer Sciences Technical Report #801

(Revised Version)
July 1989

TOWARDS AN ALGEBRAIC THEORY OF RECURSION [1]

Yannis E. Ioannidis [2]
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Eugene Wong
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

Abstract

We have developed an algebraic framework for the study of recursion. For immediate linear recursion, a Horn clause is represented by a relational algebra operator. We show 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, we are able to have the query answer 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. We present several possible decompositions of a given operator 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, we show that mutual linear recursion can also be studied within a closed semiring, by using relation vectors and operator matrices. Regarding nonlinear recursion, we first show that Horn clauses always give rise to multilinear recursion, which can always be reduced to vector bilinear recursion. We then show that bilinear recursion forms a nonassociative closed semiring. Finally, we give several sufficient and necessary & sufficient conditions for bilinear recursion to be equivalent to a linear one of a specific form. One of the sufficient conditions is derived by embedding bilinear recursion in a linear algebra.

---

[1] Some results of this work are contained in the paper "An Algebraic Approach to Recursive Inference", which appears in *Proc. 1st Inter. Conf. on Expert Database Systems*, Charleston, SC, April 1986, and in the paper "Transforming Nonlinear Recursion into Linear Recursion", which appears in *Proc. 2nd Inter. Conf. on Expert Database Systems*, Tysons Corner, VI, April 1988.

[2] Partially supported by the National Science Foundation under Grant IRI-8703592.

1. INTRODUCTION

Thus far, with few exceptions [Zani85, Ceri86, Ceri87], 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 [Ioan86a], to study several properties of recursive programs that allow transformations of such programs into more efficient ones [Ioan86b, Ioan89], and to develop a firm algebraic framework for query optimization by simulated annealing [Ioan87]. Most of the aforementioned results are hard and unnatural to obtain in a non-algebraic setting, if at all possible. The effectiveness of the algebraic approach is based on the ability 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 shall 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 ways 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 1.1. 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 1.1, and even if they do, the process of generating all the alternatives may involve significant interaction between them. Nevertheless, for our purposes, Figure 1.1 is an appropriate abstraction. The three stages are analyzed below.

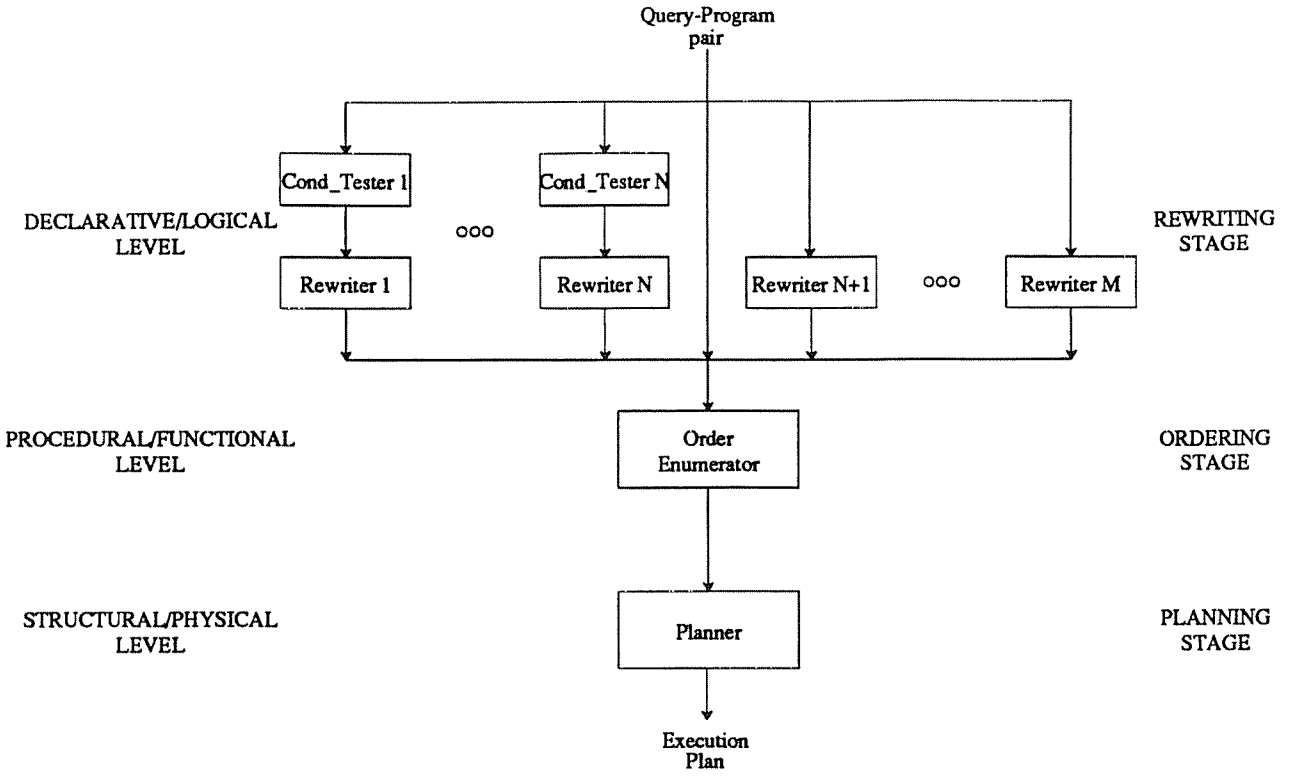

Figure 1.1: Query optimizer architecture.

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 program-query pair has certain properties (modules 1 to N in Figure 1.1), and some are always applicable (modules N+1 to M in Figure 1.1). For a transformation of the former type, the original program is first passed through a decision module ("Cond_Tester" in Figure 1.1) 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 rewritting 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, i.e., 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 to eliminate duplicates, 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 evaluation strategies. Some of these algebraic properties are useful in identifying alternatives at the rewriting stage and some at the ordering stage (Figure 1.1). It should be noted that, in this paper, we put the foundations of the algebra and only present the alternative evaluation strategies that these algebraic properties imply. We do not discuss any decision algorithms for these properties, i.e., any algorithms for the "Cond_Tester" modules of Figure 1.1. Such algorithms for some of the discussed properties appear elsewhere [Ioan89]. 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 [Ioan86a]. 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 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 rewritting 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, some of which, however, cannot be tested in finite time. Section 10 embeds bilinear recursion into a nonassociative algebra and describes the derivation of one more sufficient condition for linearizability that can be tested in finite time. 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 shall say that a series $\{x_i\}$ converges if the sequence $\{y_k = \sum_{i=1}^{k} x_i\}$ converges. In this case, the limit of the series is denoted by $\sum_{i=1}^{\infty} x_i$.

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 +, i.e., $a + b \in S$.
(2) The operator + is associative, i.e., $(a + b) + c = a + (b + c)$.
(3) 0 is an *identity element* with respect to +, i.e., $a + 0 = 0 + a = a$.
(4) For all elements in S, there is an *inverse element* in S, i.e., 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, i.e., $a + b = b + a$ for all $a, b \in S$, then the above structures are called *abelian 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 $(S-\{0\},*,1)$ is an abelian group.
In addition, for all $a, b, c \in S$ the following holds:

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

Example 2.1: The system $E_{\mathbf{R}}=(\mathbb{R},+,*,0,1)$, 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, $+$ is a binary operator on V, such that the following holds:

(1) $(V,+,0)$ is an 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 unit element of F under 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 $*$ 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 $+$, i.e., $u*(v+w) = u*v + u*w$ 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 an abelian monoid and the operation $+$ is idempotent, i.e., $a+a=a$.

(2) $(S,*,1)$ is a monoid and 0 is an *annihilator*, i.e., then $a*0 = 0*a = 0$.

(3) The operation $*$ distributes over $+$, i.e., $a*(b+c) = a*b + a*c$ 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, i.e., if $(S,+,0)$ is an abelian group, then E_S is a *closed ring*.

Example 2.2: The following system is a closed semiring: $(\{false,true\},\text{OR},\text{AND},false,true)$ [Aho74]. □

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

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

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

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

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* [Carr79, Rose86]. 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 [Hers75]. Closed semirings have been defined elsewhere as well [Aho74, Eile74]. Our definition is the one used by Aho, Hopcroft, and Ullman [Aho74], 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 [Back75, Lehm77]. 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 $D = (C_D \cup \{ERROR\}, \mathbf{R}_1, \ldots, \mathbf{R}_n)^\dagger$, 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.

† All relations appear in bold.

Relational algebra was introduced by Codd to formally describe the operations performed on relations in a database system [Codd70]. 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 on 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 $D = (C_D, \mathbf{R}_1, ..., \mathbf{R}_n)$ 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, 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, i.e., A takes as input relations of arity a and produces as output relations of arity b. Operators with the same domain are called *domain-compatible* and operators with the same range are called *range-compatible*. Likewise, if the domain of an operator A is the same as the range of an 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, i.e., 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, i.e., 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 $\emptyset$, it returns $\emptyset$, i.e., $\omega\emptyset = \emptyset$. 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 by $(A + B)\mathbf{P} = A\mathbf{P} \cup B\mathbf{P}$. Otherwise, $A + B = \omega$.

$*$ For A, B operators in R with A dr-compatible to B, multiplication is defined by $(A * B)\mathbf{P} = A(B\mathbf{P})$. Otherwise, $A * B = \omega$.

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

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 shall often omit the parentheses around them. In that case, we shall assume right associativity for them. Equality of relational operators (even outside of R) is naturally defined through set equality as

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

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:

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

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 **P**, **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 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) $A = B\ C \Rightarrow A\ (\mathbf{P} \cup \mathbf{Q}) = (B\ C)\ (\mathbf{P} \cup \mathbf{Q})$

$\Rightarrow A\ (\mathbf{P} \cup \mathbf{Q}) = B\ (C\ (\mathbf{P} \cup \mathbf{Q}))$ Definition of *

$\Rightarrow A\ (\mathbf{P} \cup \mathbf{Q}) = B\ (C\ \mathbf{P} \cup C\ \mathbf{Q})$ Induction hypothesis

$\Rightarrow A\ (\mathbf{P} \cup \mathbf{Q}) = B\ C\ \mathbf{P} \cup B\ C\ \mathbf{Q}$ Induction hypothesis

$\Rightarrow A\ (\mathbf{P} \cup \mathbf{Q}) = A\ \mathbf{P} \cup A\ \mathbf{Q}.$

$A = B\ C \Rightarrow A\ \emptyset = (B\ C)\emptyset$

$\Rightarrow A\ \emptyset = B\ (C\ \emptyset)$ Definition of *

$\Rightarrow A\ \emptyset = B\ \emptyset$ Induction hypothesis

$\Rightarrow A\ \emptyset = \emptyset.$ 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* iff for all relations **P**, **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: $\mathbf{P} \subseteq \mathbf{Q} \Rightarrow \mathbf{P} \cup \Delta\mathbf{Q} = \mathbf{Q} \Rightarrow A\ (\mathbf{P} \cup \Delta\mathbf{Q}) = A\ \mathbf{Q} \Rightarrow A\ \mathbf{P} \cup A\ \Delta\mathbf{Q} = A\ \mathbf{Q} \Rightarrow A\ \mathbf{P} \subseteq A\ \mathbf{Q}.$ □

Propositions 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, i.e., 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 $\Rightarrow A\ C \leq B\ D$

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: $A = \pi_p\ \sigma_{q1}\sigma_{q2} \cdots \sigma_{qk}\ (\mathbf{Q}X)$, i.e., 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 $1 \le k \le n$, if $B_k = 1$, then $A = B_1 \cdots B_{k-1} B_{k+1} \cdots B_n$.
- For all $1 \le k \le n$, if $B_k = 0$, then $A = 0$.
- For all $1 \le k \le n$, if $B_k = \omega$, and for all $1 \le j \le n$, $B_j \neq 0$, 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 the 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 algebraically 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. †

Proof: Let $A : 2^{C_D^a} \rightarrow 2^{C_D^b}$ 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 shall 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: {(col op col_i): col_i is a column of $\mathbf{Q}$ that does not appear in p } $\cup$ {(col op c_i): $\mathrm{c}_i \in C_D$ }. 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 $\mathbf{Q}$ $\{\mathrm{col}_i\}$ and between these columns and the constants $\{\mathrm{c}_i\}$ 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

† These operator schemes may involve relation schemes that are not present in the original database.

replaced by a single selection of the form (col = col′).

(b2) If *op* is <, the columns of 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 ≤.)

(b3) If *op* is >, the columns of Q are replaced by a single column col′ containing the maximum 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 ≥.)

(c) All columns of the original relation 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 Q′ can be derived as follows. Because of (c), the only columns of Q that are kept in Q′ are those that contribute to the range of A. In addition, columns in 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 Q kept in 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 Q′ is at most $5a+b$. An upper bound on the arity of Q′ implies an upper bound on the number of selections that can be applied on the cross product of 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 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 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 $A = \sum_{k=0}^{\infty} A_k$, with 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 B has the domain of the operators $\{A_k\}$ as the parameter relation 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 $\lim_{n\to\infty} \sum_{i=1}^{n} A_i = \omega$. 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 these 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, $A = \lim_{n\to\infty} \sum_{i=1}^{n} A_i$ 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 $d(A + B) = dA + dB$ 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 n-th power and the transitive closure of a relational operator A that is dr-compatible to itself follow directly:

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

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, i.e., for every $A \in R$ another operator $B \in R$ must exist such that $A + B = 0$. It 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 **P**,

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

which is a contradiction, since **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 **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 **P** appears in both the antecedent and the consequent. It is linear because **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 $f(\mathbf{P}, \{\mathbf{Q}_i\})$ be a function on **P**, $f: 2^{C_D^a} \rightarrow 2^{C_D^a}$. Then, (4.1) takes on the form $f(\mathbf{P}, \{\mathbf{Q}_i\}) \subseteq \mathbf{P}$, or equivalently $\mathbf{P} \cup f(\mathbf{P}, \{\mathbf{Q}_i\}) = \mathbf{P}$. 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, find $\mathbf{P}$ such that

(i) $f(\mathbf{P}, \{\mathbf{Q}_i\}) \subseteq \mathbf{P}$
(ii) $\mathbf{Q} \subseteq \mathbf{P}$
(iii) $\mathbf{P}$ is minimal with respect to (i) and (ii), i.e., 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) $\mathbf{Q} \cup f(\mathbf{P}, \{\mathbf{Q}_i\}) = \mathbf{P}$
(B) $\mathbf{P}$ is minimal with respect to (A), i.e., 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 $f(\mathbf{P}, \{\mathbf{Q}_i\})$ on $\mathbf{P}$, where $f: 2^{C_D^a} \rightarrow 2^{C_D^a}$, $a \geq 1$. The function f represents a linear recursive Horn clause of the form (4.1) iff 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* [Chan77]. Every conjunctive query can be expressed as a composition of projections, selections, and cross products and vice versa [Chan77]. Therefore, a function f, having $\{\mathbf{Q}_i\}$ as parameters and $\mathbf{P}$ as input, corresponds to a linear recursive Horn clause of the form (4.1) iff 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 $\mathbf{Q}$. We restrict our attention to functions that correspond to operators in R. Hence, $\mathbf{P}$ is written as $\mathbf{P} = B\,\mathbf{Q}$, $B \in R$, and 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:

(a) $1+A\,B = B$ (4.3)

(b) B is minimal with respect to (a), i.e., $1+A\,C = C \Rightarrow B \leq C$.

Theorem 4.1: Consider equation (5.3a) with restriction (5.3b). 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 (5.3a):

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

The second equality is due to the property that multiplication distributes over countable sums. Second, A^* is indeed the minimal solution (least fixpoint) of $1 + A\,B = B$. That is, for all operators B that satisfy (5.3a), $A^* \leq B$. This is shown by induction on the number of terms in $A^* = \sum_{k=0}^{\infty} A^k$.

Basis: For $n=0$, $\sum_{k=0}^{0} A^k = 1$, and from (5.3a) and Proposition 3.3a, $1 \leq B$.

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

$$\sum_{k=0}^{n} A^k \leq B \Rightarrow A \sum_{k=0}^{n} A^k \leq A\,B \qquad \text{Proposition 3.3d}$$

$$\Rightarrow 1 + A \sum_{k=0}^{n} A^k \leq 1 + A\,B \qquad \text{Proposition 3.3c}$$

$$\Rightarrow \sum_{k=0}^{n+1} A^k \leq 1 + A\,B \qquad \text{Closed semiring properties}$$

$$\Rightarrow \sum_{k=0}^{n+1} A^k \leq B. \qquad \text{From (5.3a)}$$

So, for all $n \geq 0$, $\sum_{k=0}^{n} A^k \leq B$. 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 $B = 1 + A\,B$, $A^* \leq B$, and A^* is the least fixpoint of (5.3a). □

Theorem 4.1, originally due to Tarski [Tars55], has been used in the study of the semantics of logic programs extensively [VanE76, Apt82]. In the database context, it was first examined by Aho and Ullman [Aho79]. 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 algebraically manipulate the query answer, which is represented by A^* possibly multiplied with other operators also, e.g., 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, i.e., they are not produced by some nonrecursive Horn clause.

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

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

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

$$\mathbf{P}(y,x) \rightarrow \mathbf{P}(x,y)$$

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

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

To the contrary, the next one is not, because of the presence of both **P** and **Q** in the antecedent of the first Horn clause.

$$\mathbf{P}(w,z) \wedge \mathbf{Q}(x,z) \wedge \mathbf{T}(z,y) \rightarrow \mathbf{P}(x,y)$$

$$\mathbf{P}(y,x) \rightarrow \mathbf{P}(x,y)$$

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

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

□

Note that this definition of linear is different (more restrictive) from the one given by Bancilhon and Ramakrishnan [Banc86a]. 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*, and we use the term linear according to Definition 5.1 [Cosm86].

Consider a linear system of mutually recursive Horn clauses defining relations $\mathbf{P}_1,\mathbf{P}_2,...,\mathbf{P}_n$. Each Horn clause is represented algebraically using a linear operator in R. Thus, a system of n equations is generated with n unknown variables $\mathbf{P}_1,\mathbf{P}_2,...,\mathbf{P}_n$ 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 $\mathbf{P}_1$ and $\mathbf{P}_2$, 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:

$$\mathbf{P}_1 = A\,\mathbf{P}_1 \cup B\,\mathbf{P}_2 \cup \mathbf{Q}_1$$

$$\mathbf{P}_2 = C\,\mathbf{P}_1 \cup D\,\mathbf{P}_2 \cup \mathbf{Q}_2.$$ □

Define $M_n(R)$ as the set of nXn 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 $\underline{A} * \underline{B} = [\sum_{k=1}^{n} A_{ik} B_{kj}]$.
- $\underline{0}$ 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: System $E_{M_n(R)} = (M_n(R),+,*,\underline{0},\underline{1})$ is a closed semiring.

Proof: It is known that matrices over a closed semiring form a closed semiring [Aho74]. 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

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

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 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

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

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:

$$\mathbf{P}_1 = (A + B D^* C)^* (B D^* \mathbf{Q}_2 \cup \mathbf{Q}_1)$$

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

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 [Hens84]. 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 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. □

6. ALGEBRAIC TRANSFORMATIONS OF LINEAR RECURSION AT THE REWRITING STAGE

In this section, we shall 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 1.1). 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 $A = B\ C$. 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.

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

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

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

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

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

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

Consider an arbitrary term $D = B^{i_1}C^{j_1}B^{i_2}C^{j_2}\cdots B^{i_m}C^{j_m}$. Assume that $C\,B \leq B^k\,C^l$, with $k \in \{0,1\}$ (the case of $l \in \{0,1\}$ is handled similarly). We shall 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

[†] The condition of Theorem 6.1 can be easily tightened to $C\,B \leq (1+B)C^*$ or $C\,B \leq B^*\,(1+C)$. For operator schemes, the two conditions are equivalent [Sagi80], 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.

$I = n = i_1 + \cdots + i_m$ 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 shall 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 $D = (B^{i_1} C^{j_1} \cdots B^{i_m-1}) B C^{j_m}$. By the induction hypothesis, we have that $D \leq B^{i_1} C^J B C^{j_m}$, 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 $D = (B^{i_1} C^{j_1} \cdots B^{i_m-1}) B C^{j_m}$. By the induction hypothesis, we have that $D \leq B^n C^J B C^{j_m}$, 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 $A^* = \sum_{I=0,J=0}^{\infty} B^I C^J = B^* C^*$. □

Corollary 6.1: If B and C commute, i.e., $B\,C = C\,B$, 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 [Naug88]. Further elaboration on the relationship between commutativity and separability appears elsewhere [Ioan89].

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

$$C\,B \leq B^k\,C^l,$$

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 $m=2$, i.e., if D is of the form $D = B^{i_1} C B C^{j_2}$, then applying (6.1) 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 (6.1) yields either $D \leq D_1 C^2 B D_2$ or $D \leq D_1 C B^2 D_2$. We shall show that in all cases $D=0$. Equation (6.1) yields

$$D_1C^2BD_2 \le D_1CB^kC^lD_2 \quad \text{and}$$

$$D_1CB^2D_2 \le D_1B^kC^lBD_2 \le D_1B^kC^{l-1}B^kC^lD_2.$$

Replacing $D_1B^kC^{l-2}$ by D_1 in the latter expression yields $D_1CB^kC^lD_2$. Hence, whether $j_1 \ge 2$, or $i_2 \ge 2$, or $m > 2$, we have shown that $D \le D_1CB^kC^lD_2$. Our goal is to prove that $D_1CB^kC^lD_2 = 0$. We shall achieve this by showing that

$$\text{for all } g, \quad CB^kC^l \le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g CB^kC^{(g+1)l}. \tag{6.3}$$

The formula to the right of $\le$ is well defined, since both $k \ge 2$ and $l \ge 2$. We shall prove (6.3) by induction on g.

Basis: For $g=0$, (6.3) yields $CB^kC^l \le CB^kC^l$, which clearly holds.

Induction Step: Assume that the claim is true for some $g \ge 0$. We shall prove it for $g+1$ by repeated applications of (6.1).

$$\begin{aligned}
CB^kC^l &\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g CB^kC^{(g+1)l} && \text{Induction hypothesis}\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g B^kC^lB^{k-1}C^{(g+1)l} && \text{Applying (6.1) once}\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g B^kC^{l-1}B^kC^lB^{k-2}C^{(g+1)l} && \text{Applying (6.1) twice}\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g B^kC^{l-1}B^kC^{l-1}B^kC^lB^{k-3}C^{(g+1)l} && \text{Applying (6.1) three times}\\
&\quad \cdots\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g (B^kC^{l-1})^hB^kC^lB^{k-h-1}C^{(g+1)l} && \text{Applying (6.1) h times}\\
&\quad \cdots\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g (B^kC^{l-1})^{k-1}B^kC^lC^{(g+1)l} && \text{Applying (6.1) k times}\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^g (B^kC^{l-1})^{k-2}B^kC^{l-1}B^kC^{(g+2)l}\\
&\le ((B^kC^{l-1})^{k-2}B^kC^{l-2})^{g+1}CB^kC^{(g+2)l}.
\end{aligned}$$

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 $(g+2)l \ge p$, since $C^p=0$, (7.3) yields that $CB^kC^l=0$, which in turn implies that $D=0$. Thus, considering all cases, either $D \le B^IC^J$, for some I, J, or $D=0$. Hence, we may conclude that $A^* = (B+C)^* = B^*C^*$. □

In Theorem 6.2, the meaning of $B^p=0$ ($C^p=0$) is that the data in the parameter relation of B (C) is acyclic. 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

$$CB \le B^k \text{ or } CB \le C^k \tag{6.4a}$$

and

$$B\,C \le B^l \text{ or } B\,C \le C^l, \tag{6.4b}$$

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

Proof: Condition (6.4a) implies that the conditions of Theorem 6.1 hold. Thus, $A^* = (B+C)^* = B^*\,C^* = B^* + C^* + B^*\,B\,C\,C^*$. Assume that in (6.4b) it is $B\,C \le B^l$ that holds (the other case is treated similarly). Consider an arbitrary term of $B^*\,B\,C\,C^*$, i.e., $D = B^i C^j$, $i, j \ge 1$. We distinguish two cases:

$l \ge 1$: Clearly, (6.4b) implies that $B^i C^j \le B^{i-1+l} C^{j-1} \le B^{i-2+2l} C^{j-2} \le \cdots \le B^{i+(l-1)j}$.

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

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

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 omitted).

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. Theorem 6.2 is vacuously true for operator schemes, since for an operator scheme B, $B^p=0$ can only be true when $B=0$. 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, i.e., $(B+C)^* = B^* + C^*$, 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 $C\,B \le B^k\,C^l$.

Proof: Assume that $A^* = (B+C)^* = B^*\,C^*$. This implies that $(B+C)^* \le B^*\,C^*$. 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 $\le$ to a term

of $B^* C^*$ [Sagi80]. † The operator $C B$ is a term in $(B+C)^*$. This yields that there exist k and l such that $C B \leq B^k C^l$. □

Theorem 6.5: Consider operator schemes A, B, and C, such that $A = B + C$. Then, $A^* = (B+C)^* = B^* + C^*$ iff there exist k and l such that

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

and

$$B C \leq B^l \text{ or } B C \leq C^l,$$

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 $C B$ and $B C$ of $(B+C)^*$ [Sagi80]. This yields that there exist k and l such that $C B \leq B^k$ or $C B \leq C^k$, and $B C \leq B^l$ or $B C \leq C^l$. □

6.2. Decompositions of $(B C)^*$

Theorem 6.6: Let $A = B C$. If there exists k such that

$$C B = B^k C,^{\ddagger} \tag{6.5}$$

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

Proof: Clearly, $A^* = (B C)^* = \sum_{m=0}^{\infty} (B C)^m$. Consider an arbitrary term of the sum, $D = (B C)^m$. We shall show by induction on m that $D = B^{(\sum_{i=0}^{m-1} k^i)} C^m$.

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

Induction Step: Assume that the claim is true for some $m \geq 0$. We shall prove it for $m+1$ by an application of (6.5).

$(B C)^m = B^{\sum_{i=0}^{m-1} k^i} C^m \Rightarrow (B C)^{m+1} = B C B^{\sum_{i=0}^{m-1} k^i} C^m$ Induction hypothesis

$\Rightarrow (B C)^{m+1} = B B^{k * \sum_{i=0}^{m-1} k^i} C^{m+1}$ Repeated applications of (6.5)

† 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 shall refer to their result in the more general form.

‡ Similar results are obtained if $C B = B C^k$.

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

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

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

Corollary 6.2: If B and C commute, i.e., $B\ C = C\ B$, then $A^* = (B\ C)^* = \sum_{m=0}^{\infty} B^m\ C^m$.

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 $B\ C$, both in terms of the number of tuples and in terms of the arity. Investigation of the precise implications of such decompositions on performance are part of our future plans.

Theorem 6.7: Let $A = B\ C$. If $C\ B = B\ C$ and there exist K and N, $K<N$, such that $B^N = B^K$, then

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

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

$$B^m = B^{m+i(N-K)}, \text{ for all } K \le m < N \text{ and all } i \ge 0. \tag{6.6}$$

The theorem is the result of the following transformations, which make use of (6.6):

$$A^* = (B\ C)^* = \sum_{m=0}^{K-1} B^m C^m + \sum_{m=K}^{\infty} B^m\ C^m \qquad \text{Corollary 6.2}$$

$$= \sum_{m=0}^{K-1} B^m C^m + \sum_{m=K}^{N-1} B^m\ (\sum_{i=0}^{\infty} C^{m+i(N-K)}) \qquad \text{From (6.6)}$$

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

With respect to operator schemes, Theorem 6.7 examines commutativity of B and C in conjunction with B being *torsion*, which is a special case of *uniform boundedness* [Ioan85, Naug86]. It shows that, in such cases, 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^*. Theorem 6.7 is related to the work of Naughton on *recursively redundant predicates* [Naug89a]. The relationship between commutativity, uniform boundedness, and recursively redundant predicates is examined in detail elsewhere [Ioan89].

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

Theorem 6.8: Let A, B, C be linear operators. If $A\,B = B\,A$ and $A\,C = B\,C$, then $A^*\,C = B^*\,C$.

Proof: We shall prove by induction on $m \geq 0$ that $A^m\,C = B^m\,C$.

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

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

$$A^{m+1}\,C = A\,(A^m\,C) = A\,(B^m\,C) = B^m\,(A\,C) = B^{m+1}\,C.$$

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

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

Theorem 6.9: Let $A = B\,C$. Then,

$$A^* = (BC)^* = 1 + B\,(CB)^*\,C. \tag{6.7}$$

Proof: The following series of equations prove the theorem:

$$A^* = (BC)^* = \sum_{m=0}^{\infty}(BC)^m = 1 + \sum_{m=1}^{\infty}(BC)^m = 1 + B\,(\sum_{m=0}^{\infty}(CB)^m)\,C = 1 + B\,(CB)^*\,C. \qquad \square$$

Expression (6.7) 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.

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

$$\rho\,A \leq B\,\rho \tag{6.8}$$

then $\rho\,A^* \leq B^*\,\rho$. If (6.8) holds with equality, then $\rho\,A^* = B^*\,\rho$.

Proof: Assume that $\rho\,A \leq B\,\rho$. We shall 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 shall prove it for $m+1$.

$\rho\,A^{m+1} = (\rho\,A^m)\,A \leq (B^m\,\rho)\,A$ Induction hypothesis

$$= B^m (\rho A) \leq B^m (B \rho) = B^{m+1} \rho \qquad \text{From (6.8)}$$

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

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

The case where (6.8) holds with equality is easily seen to be true as well. □

An interesting case arises when (6.8) 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. Another interesting case is when (6.8) holds with equality, ρ is a projection π, and $B=\pi A_\pi$, where A_π is the same operator as A but operating only 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. The above applications of Theorem 6.10 allow the query processor to take into account any selections or projections involved in a query. Usually, such transformations cause significant improvements in performance.

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.

7. ALGEBRAIC TRANSFORMATIONS OF LINEAR RECURSION AT THE ORDERING STAGE

In this section, we shall present algebraic transformations of the query answer that affect the ordering stage of query optimization (Figure 1.1). We shall 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 shall 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 shall only present a small number of them that have been previously proposed in the literature. In the second subsection, we shall 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^*

Naive Evaluation [Aho79, Banc85]

This is the original algorithm proposed for the evaluation of a recursive program. Its algebraic expression is based on the fact that $\sum_{m=0}^{n} A^m = (1+A)^n$. Thus, we have that

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

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

Seminaive Evaluation [Banc85]

This algorithm corresponds to the definition of A^* as a series, i.e., $A^* = \sum_{m=0}^{\infty} A^m$. Again, each power is computed from the previous one: $A^{m+1} = AA^m$.

Smart/Logarithmic Evaluation [Vald86, Ioan86a]

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 seminaive (for any finite expansion of A^*, it is $2*\log_2 N$ for smart vs. N for seminaive), but they involve larger operators.

Minimal Evaluation [Ioan86a]

This algorithm corresponds to the following formula:

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

This expression has about $3 \log_3 N$ multiplications (when seminaive needs N). Clearly, algorithms can be created that need $n\log_n N$ multiplications for arbitrary n. They correspond to the following family of formulas:

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

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. This is were the minimal evaluation owes its name.

Query-Subquery (QSQ) Evaluation [Viei86]

For a 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: $\sigma A^k = (\sigma A^{k-1}) A$.

Prolog [Cloc81]

The formula that corresponds to the evaluation strategy of Prolog is the same as QSQ. The difference is in details of the planning stage, i.e., 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 useful at the planning stage of a recursive query optimizer: two different algorithms are represented by the same formula at the ordering stage.

1.1. Specialized Transformations of A^*

Let $A = B\ C = C\ B$. By Corollary 7.2, we have that $A^* = \sum_{m=0}^{\infty} B^m C^m$. 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. 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

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

Henschen-Naqvi [Hens84]

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: $\sigma C^{m+1}=(\sigma C^m)C$.

Counting [Banc86b, Sacc86a, Sacc86b]

The counting algorithm as proposed by Bancilhon et al. and by Sacca and Zaniolo 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, i.e., 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). This difference can be further enhanced or can be eliminated depending on 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 [Banc86b].) Thus we have another example that shows that the developed algebra is not useful in the planning stage of query optimization.

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 arbitrary number of operators, thus capturing algebraicly the results of previous work by Sacca and Zaniolo [Sacc86a, Sacc86b]. 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:

$$\sigma_0\sigma_1\sigma_2\cdots\sigma_n(A_1A_2\cdots A_n)^* = \sum_{m=0}^{\infty}(\sigma_1A_1^m)(\sigma_2A_2^m)\cdots(\sigma_nA_n^m)\sigma_0.$$

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 $\sigma_iA_i^{m+1}$ from $\sigma_iA_i^m$ or not.

Shapiro-McKay [Shap80]

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.

Han-Lu [Han86]

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 shall 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 shall 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 $\underline{\mathbf{P}} = (\mathbf{P}_1,\mathbf{P}_2,...,\mathbf{P}_n)$ be the vector of relations that appear in the consequents of its elements with arities $\{a_1,a_2,\cdots,a_n\}$. 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

$$\mathbf{Q}_i \rightarrow \mathbf{P}_i,$$

and let $\underline{\mathbf{Q}}$ be the vector $\underline{\mathbf{Q}} = (\mathbf{Q}_1,\mathbf{Q}_2,...,\mathbf{Q}_n)$. 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. 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:

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

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

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

Let $\underline{P} = 2^{C\hat{\sigma}} X \cdots X 2^{C\hat{\sigma}}$, $a_i \geq 1$. Clearly, $\underline{\mathbf{P}} \in \underline{P}$. Recall that *addition* $\oplus$ in $\underline{P}$ is defined as

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

Define $\underline{\varnothing}$ as the n-vector $(\varnothing,\varnothing, \ldots ,\varnothing)$. Note that $\underline{\varnothing} \oplus \underline{\mathbf{P}} = \underline{\mathbf{P}} \oplus \underline{\varnothing} = \underline{\mathbf{P}}$. 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 $g : \underline{P} \; X \; \underline{P} \rightarrow \underline{P}$ 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 $g : \underline{P} \; X \; \underline{P} \; X \; \cdots \; X \; \underline{P} \rightarrow \underline{P}$ (the domain of g is the product of $\underline{P}$ m times) is *m-linear* if for all i, given $\underline{\mathbf{P}}_1, \ldots, \underline{\mathbf{P}}_{i-1}, \underline{\mathbf{P}}_{i+1}, \ldots, \underline{\mathbf{P}}_m$, $g(\underline{\mathbf{P}}_1, \ldots, \underline{\mathbf{P}}_i, \ldots, \underline{\mathbf{P}}_m)$ 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 $f(\underline{\mathbf{P}},\underline{\mathbf{P}}, \cdots ,\underline{\mathbf{P}}) \oplus \underline{\mathbf{Q}} = \underline{\mathbf{P}}$ is the equation representing a set of mutually recursive Horn clauses, where $f : (\underline{P})^m \rightarrow \underline{P}$, 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, with $g : P_{i_1} \; X \; P_{i_2} \; X \; \cdots \; X \; P_{i_k} \rightarrow P_{i_0}$, where for all $0 \leq j \leq k$, $1 \leq i_j \leq n$. Given fixed $\underline{\mathbf{Q}}_1, \cdots ,\underline{\mathbf{Q}}_{i-1},\underline{\mathbf{Q}}_{i+1}, \cdots ,\underline{\mathbf{Q}}_k$, consider $g(\underline{\mathbf{Q}}_1, \cdots ,\underline{\mathbf{Q}}_i, \cdots ,\underline{\mathbf{Q}}_k)$ as a function of $\underline{\mathbf{Q}}_i$. Clearly, g can be expressed as a composition of projections, selections, and cross products of $\underline{\mathbf{Q}}_1, \cdots ,\underline{\mathbf{Q}}_{i-1},\underline{\mathbf{Q}}_{i+1}, \cdots ,\underline{\mathbf{Q}}_k$, 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 can always be eliminated by solving it as a linear equation with operator matrix $\underline{A}$.

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

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 $\underline{\mathbf{P}}$:

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

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

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

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

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

$$\underline{\mathbf{P}}_1 = g_1(\underline{\mathbf{P}},\underline{\mathbf{P}})$$

$$\underline{\mathbf{P}}_2 = g_2(\underline{\mathbf{P}},\underline{\mathbf{P}}_1)$$

$$\cdots$$

$$\underline{\mathbf{P}}_{m-2} = g_{m-2}(\underline{\mathbf{P}},\underline{\mathbf{P}}_{m-3})$$

$$\underline{\mathbf{P}} = g_{m-1}(\underline{\mathbf{P}},\underline{\mathbf{P}}_{m-2}) \oplus \underline{\mathbf{Q}}.$$

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

Because of the generic 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 [Tars55]. For its proof, a partial order on $\underline{P}$ is needed, which is defined as $\underline{\mathbf{P}} \subseteq \underline{\mathbf{P}}' \Leftrightarrow (\mathbf{P}_1 \subseteq \mathbf{P}_1', \cdots, \mathbf{P}_n \subseteq \mathbf{P}_n')$.

Theorem 8.1: The minimal solution of equation (8.5) is the limit of the sequence

$$\underline{\mathbf{P}}_{m+1} = g(\underline{\mathbf{P}}_m,\underline{\mathbf{P}}_m) \oplus \underline{\mathbf{Q}}, \tag{8.6}$$
$$\underline{\mathbf{P}}_0 = \underline{\emptyset}.$$

Proof: Because g is bilinear, g is also monotone on either of its two arguments (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 $\underline{\mathbf{P}}_0 = \underline{\emptyset} \subseteq \underline{\mathbf{Q}} = \underline{\mathbf{P}}_1$.

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

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

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

For the rest of the paper, g is viewed as multiplication, i.e., $g(\underline{\mathbf{P}},\underline{\mathbf{Q}}) = \underline{\mathbf{P}} \bullet \underline{\mathbf{Q}}$. Note that $\bullet$ is not necessarily associative. The system $E_{\underline{P}} = (\underline{P}, \oplus, \bullet, \underline{\emptyset})$ can now be defined as follows:

- $\underline{P}$ The set of n-vectors of relations from C_D with arities $\{a_1, a_2, \cdots, a_n\}$.
- $\oplus$ Addition of vectors as defined above.
- $\bullet$ Multiplication of vectors defined as $\underline{\mathbf{P}} \bullet \underline{\mathbf{Q}} = g(\underline{\mathbf{P}},\underline{\mathbf{Q}})$.
- $\underline{\emptyset}$ The additive identity, i.e., 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 $\bullet$. □

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

$$\mathbf{P}^1 = \mathbf{P}, \quad \mathbf{P}^n = \mathbf{P} \bullet \mathbf{P}^{n-1}, \quad \text{for all } n \geq 1.$$

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

In this section, we 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 will be used instead of (8.5):

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

Moreover, none of the forthcoming results is of any value when $\bullet$ is parameterized with actual relations, so we shall always be concerned with $\bullet$ 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 [Gaif87]. 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

$$\underline{P} = \underline{P} \bullet \underline{Q} \oplus \underline{Q}$$

or of the form

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

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 shall mostly be concerned with left-linearizability. Note that the solution of (9.2) is $\underline{\sum}_{k=1}^{\infty} \underline{Q}^k$.[†]

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

$$\underline{P}^* \supseteq \underline{\sum}_{k=1}^{\infty} \underline{Q}^k.$$

Proof: Using (8.6), we shall prove by induction on m, $m \geq 0$, that

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

[†] $\sum$ denotes a series with respect to $\oplus$.

Basis: For $m=0$, it is $\underline{\mathbf{P}}_1 = \underline{\mathbf{Q}}$, which is a consequence of (8.6) with $\underline{\mathbf{P}}_0 = \underline{\emptyset}$.

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

$$\underline{\mathbf{P}}_{m+2} = \underline{\mathbf{P}}_{m+1} \bullet \underline{\mathbf{P}}_{m+1} \oplus \underline{\mathbf{Q}}$$

$$\supseteq \sum_{l=1}^{m+1} \underline{\mathbf{Q}}^l \bullet \sum_{k=1}^{m+1} \underline{\mathbf{Q}}^k \oplus \underline{\mathbf{Q}} \qquad \text{Induction hypothesis and monotonicity of } \bullet$$

$$\supseteq \underline{\mathbf{Q}} \bullet \sum_{k=1}^{m+1} \underline{\mathbf{Q}}^k \oplus \underline{\mathbf{Q}} = \sum_{k=1}^{m+2} \underline{\mathbf{Q}}^k. \qquad \text{Monotonicity of } \bullet$$

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

Intuitively, one expects that, since $E_{\underline{P}}$ is a nonassociative closed semiring and E_R is a closed semiring, associativity should be sufficient to ensure equivalence of bilinear to linear recursion. Indeed, as we prove later, associativity of $\bullet$ is a simple, albeit strong, condition to ensure linearizability. Linearizability, however, is ensured by a range of conditions weaker than associativity. They are all variants of the notions of power-associativity and alternativeness. We borrowed both terms from the study of nonassociative algebras, where they are in common use [Scha66]. We shall proceed from the stronger (less general) to the weaker (more general) condition.

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

$$\underline{\mathbf{P}}^2 \bullet \underline{\mathbf{Q}} \subseteq \underline{\mathbf{P}} \bullet (\cdots (\underline{\mathbf{P}} \bullet (\underline{\mathbf{P}} \bullet \underline{\mathbf{Q}})) \cdots), \qquad (9.3)$$

(n times)

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

$$\underline{\mathbf{P}} \bullet \underline{\mathbf{Q}}^2 \subseteq (\cdots ((\underline{\mathbf{P}} \bullet \underline{\mathbf{Q}}) \bullet \underline{\mathbf{Q}}) \cdots) \bullet \underline{\mathbf{Q}}.$$

(n times)

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

Theorem 9.1: If $\bullet$ is left-subalternative, then (9.1) is left-linearizable.

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

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

and

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

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

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

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

Similarly, we have that

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

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 $\underline{\mathbf{P}}^* \subseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k$. Together with Lemma 9.1, this implies that the solution of (9.1) is given by

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

that is, it is equal to the solution of (9.2). Hence, because of the left-subalternativeness of $\bullet$, the bilinear (9.1) is left-linearizable. □

Corollary 9.1: If $\bullet$ 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 $\bullet$ 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 that is right-alternative, but it is 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 $\bullet$ represent the function of the Horn clause. We shall show that $(\mathbf{Q} \bullet \mathbf{P}) \bullet \mathbf{P} = \mathbf{Q} \bullet (\mathbf{P} \bullet \mathbf{P})$, whereas $(\mathbf{P} \bullet \mathbf{P}) \bullet \mathbf{Q} \neq \mathbf{P} \bullet (\mathbf{P} \bullet \mathbf{Q})$. 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 $\bullet$ is associative then (9.1) is both left- and right-linearizable.

Proof: We shall show that associativity implies alternativeness, whence by Corollary 9.2, the claim follows *a fortiori*. Associativity implies that for all $\underline{\mathbf{P}},\underline{\mathbf{Q}},\underline{\mathbf{R}} \in \underline{P}$, $\underline{\mathbf{P}} \bullet (\underline{\mathbf{Q}} \bullet \underline{\mathbf{R}}) = (\underline{\mathbf{P}} \bullet \underline{\mathbf{Q}}) \bullet \underline{\mathbf{R}}$. 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 $\bullet$ 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

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

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 $\bullet$ is power-subassociative,

Proof: Assume that $\bullet$ is power-subassociative. Using (8.6), we shall 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) yields $\underline{\mathbf{P}}_1 \subseteq \underline{\mathbf{Q}}$, which is a consequence of (8.6) with $\underline{\mathbf{P}}_0 = \underline{\emptyset}$.

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

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

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 $\underline{\mathbf{P}}^* \subseteq \sum_{k=1}^{\infty} \underline{\mathbf{Q}}^k$. 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 $\bullet$, 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, $\underline{\mathbf{Q}}^k \underline{\mathbf{Q}}^l \subseteq \underline{\mathbf{P}}^*$. Hence, from (9.12), we have that for all k,l,

$$\underline{\mathbf{Q}}^k \underline{\mathbf{Q}}^l \subseteq \sum_{m=1}^{\infty} \underline{\mathbf{Q}}^m.$$

The left-hand side of the above formula is a conjunctive query [Chan77], and the right-hand side is a set of conjunctive queries. Applying the theorem of Sagiv and Yannakakis [Sagi80] yields that for all k,l, there exists some m such that $\underline{\mathbf{Q}}^k \underline{\mathbf{Q}}^l \subseteq \underline{\mathbf{Q}}^m$, i.e., that $\bullet$ is power-subassociative. □

Corollary 9.4: If $\bullet$ 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 $\underline{Q}^m \bullet \underline{Q} = \underline{Q} \bullet \underline{Q}^m = \underline{Q}^{m+1}$. Thus, when $\bullet$ is power-associative, (9.1) is right-linearizable as well. □

Example 9.2: Consider the bilinear recursive Horn clause:

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

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

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

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

Clearly, viewed as conjunctive queries, (9.14) is contained (9.13). Replacing **Q** and **S** with arbitrary powers of **P** yields $(\mathbf{P} \bullet \mathbf{P}^k) \bullet \mathbf{P}^l \subseteq \mathbf{P} \bullet \mathbf{P}^k$, or $\mathbf{P}^{k+1} \bullet \mathbf{P}^l \subseteq \mathbf{P}^{k+1}$, i.e., $\bullet$ is power-subassociative. Theorem 9.2 guarantees that the given bilinear recursion is left-linearizable. It is easy to verify that $\bullet$ 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. Clearly, 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 $\bullet$ 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

$$\underline{\mathbf{P}}^m \bullet \underline{\mathbf{P}}^n \subseteq (\cdots((\underline{\mathbf{P}} \bullet \underline{\mathbf{P}}) \bullet \underline{\mathbf{P}}) \cdots) \bullet \underline{\mathbf{P}}.$$

(k times)

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

Recently, the result of Theorem 9.2 have been made tighter by Ramakrishnan et al. [Rama89]. We shall proceed in proving their theorem in the algebraic framework of this paper. (Although the two proofs are different, they essentially use the same techniques.)

Definition 9.3: The bilinear multiplication $\bullet$ 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

$$\underline{\mathbf{P}}^2 \bullet \underline{\mathbf{P}}^l \subseteq \underline{\mathbf{P}}^m. \tag{9.15}$$

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 $\bullet$ is power-left-subalternative, then for all $\underline{P} \in \underline{P}$ and for all $n \geq 1$, $(\underline{P} \oplus \underline{P}^2)^n \subseteq \sum_{m=1}^{\infty} \underline{P}^m$.

Proof: The proof is by induction on n.

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

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

$$(\underline{P} \oplus \underline{P}^2)^{n+1} = (\underline{P} \oplus \underline{P}^2) \bullet (\underline{P} \oplus \underline{P}^2)^n$$

$$\subseteq (\underline{P} \oplus \underline{P}^2) \bullet \sum_{m=1}^{\infty} \underline{P}^m \qquad \text{Induction hypothesis and monotonicity of } \bullet$$

$$= \sum_{m=1}^{\infty} \underline{P}^{m+1} \oplus \sum_{m=1}^{\infty} \underline{P}^2 \bullet \underline{P}^m \qquad \text{Bilinearity of } \bullet$$

$$\subseteq \sum_{m=1}^{\infty} \underline{P}^{m+1} \oplus \sum_{m \in K} \underline{P}^m \subseteq \sum_{m=1}^{\infty} \underline{P}^m \qquad \text{Power-left-subalternativeness of } \bullet$$

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

The following theorem has been shown to hold.

Theorem 9.3 [Rama89] The bilinear multiplication $\bullet$ is power-left-subalternative if and only if it is power-subassociative.

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

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

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

$$\underline{P}^k \subseteq (\underline{P} \oplus \underline{P}^2)^k = \underline{Q}^k,$$

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

Hence, we have that

$$\underline{P}^{k+1} \bullet \underline{P}^l \subseteq \underline{Q}^k \bullet \underline{Q}^l \qquad \text{Monotonicity of } \bullet$$

$$\subseteq \underline{Q}^n = (\underline{P} \oplus \underline{P}^2)^n, \text{ for some } n \geq 1 \qquad \text{Induction hypothesis}$$

$$\subseteq \sum_{m=1}^{\infty} \underline{P}^m. \qquad \text{Lemma 9.2}$$

Thus we have shown that $\underline{P}^k \bullet \underline{P}^l \subseteq \sum_{m=1}^{\infty} \underline{P}^m$. Since the left-hand side of the above formula is a conjunctive query and its right-hand side one is a set of conjunctive queries, we can again use the result of Sagiv and Yannakakis [Sagi80] to yield $\underline{P}^k \bullet \underline{P}^l \subseteq \underline{P}^m$, for some $m \geq 1$. This concludes the induction step and the proof of the theorem is complete. □

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 results of this section on sufficient and necessary & sufficient conditions for linearizability are summarized in Figure 9.1. 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 $E_{\underline{P}}$, left-subalternativeness implies power-subassociativity. In general, we do not expect this to hold for all nonassociative closed semirings. To the contrary, for all nonassociative algebras, alternativeness implies power-associativity [Scha66].

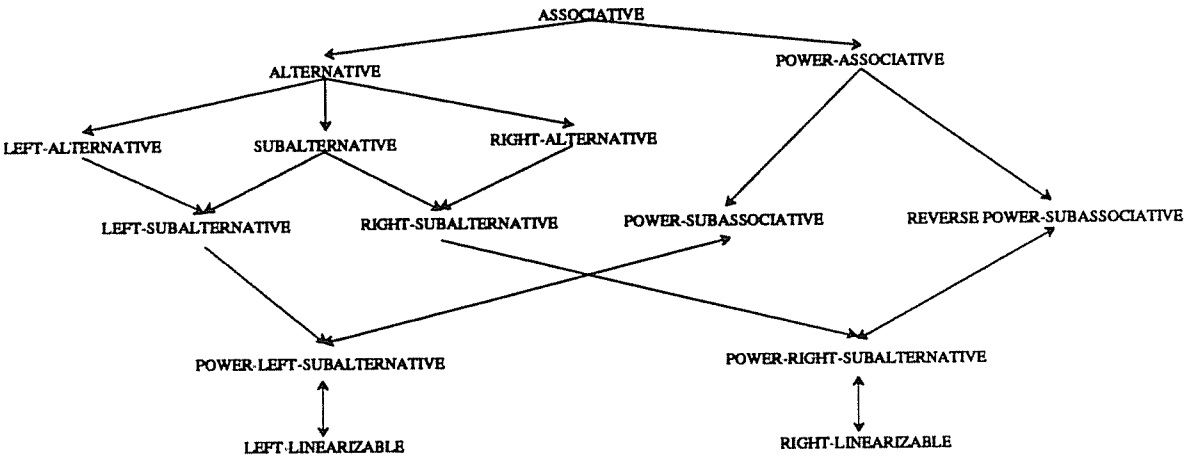

Figure 9.1: Relationship of properties of • implying linearizability.

10. EMBEDDING $\underline{P}$ IN A LINEAR ALGEBRA

Unfortunately, none of the properties that are equivalent to left-linearizability, i.e., power-subassociativity and power-left-subalternativeness, can be tested in finite time, since in general, tests have to be conducted for all possible values of one or two integer parameters. On the other hand, left-(sub)alternativeness can be tested in finite time. In this section, we shall derive another sufficient condition for left-linearizability, which is related to power-associativity and can be tested in a finite amount of time as well. This is done by embedding the operations $\oplus$ and • in a linear algebra. The

main effect of this embedding 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 linear 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 shall 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. One disadvantage of this approach is that currently 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.

Recall that P_i, $1 \le i \le n$, denotes the collection of all relations having the same scheme (arity) as $\mathbf{P}_i$, i.e., $P_i = 2^{C_D^{a_i}}$, and that $\underline{P}$ is defined as $\underline{P} = 2^{C_D^{a_1}} X \cdots X 2^{C_D^{a_n}}$, $a_i \ge 1$. Also, define $\underline{C}_D = C_D^{a_1} X \cdots X C_D^{a_n}$. Consider a relation $\mathbf{Q}$ with the same arity as $\mathbf{P}_i$, i.e., $\mathbf{Q} \in \underline{P}_i$ or $\mathbf{Q} \subseteq C_D^{a_i}$. As such, $\mathbf{Q}$ can be viewed as a function $\mathbf{Q}: C_D^{a_i} \to \{0,1\}$, defined by

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

The reason why we want to look at $\mathbf{Q}$ this way is to be able to embed relations in a richer algebraic structure. The addition operation $\oplus$ defined on $\underline{P}$ can now be interpreted as a binary operator on $\{0,1\}$ given by

$$\text{for all } \underline{t} \in \underline{C}_D, \quad (\underline{\mathbf{P}} \oplus \underline{\mathbf{Q}})(\underline{t}) = \begin{cases} 1 & \text{if } \max(\underline{\mathbf{P}}(\underline{t}), \underline{\mathbf{Q}}(\underline{t})) = 1 \\ 0 & \text{otherwise} \end{cases}.$$

This is so, because if $\underline{\mathbf{P}}(\underline{t})=1$ or $\underline{\mathbf{Q}}(\underline{t})=1$, then $\underline{t}$ belongs to the sum of $\underline{\mathbf{P}}$ and $\underline{\mathbf{Q}}$ as well.

Each P_i corresponds to the space of all functions mapping $C_D^{a_i}$ into $\{0,1\}$, and therefore, $\underline{P}$ corresponds to the set

$$\underline{P} = \{ \text{all functions } \rho: \underline{C}_D \to \{0,1\} \}.$$

We can now extend each P_i by considering all functions mapping $C_D^{a_i}$ into $\mathbb{R}$, the set of real numbers. Each such function will be called an *extended relation*. In this way, we can extend $\underline{P}$ to

$$\underline{P}^r = \{\text{all functions } \rho: \underline{C}_D \to \mathbb{R}\}^\dagger,$$

which is well known to be a vector space over the field of reals $\mathbb{R}$ [Hers75]. 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

† The superscript r is for reals. It will be used to tag elements, operations, and functions of $\underline{P}^r$.

be confused with + as defined in Section 3 for addition of linear relational operators.) If we define for real numbers a

$$|a| = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases},$$

then

$$|a+b| = |a| \oplus |b|, \quad \text{whenever } a, b > 0.$$

Applying this to relations, we can use $|\ |$ to map $\underline{P}^r$ into $\underline{P}$ and have the relationship:

$$|\underline{\mathbf{Q}}^r + \underline{\mathbf{R}}^r| = |\underline{\mathbf{Q}}^r| \oplus |\underline{\mathbf{R}}^r|, \quad \text{for all } \underline{\mathbf{Q}}^r, \underline{\mathbf{R}}^r \in \underline{P}^r.$$

Given a function $f: \underline{P} \to \underline{P}$, define an extension $f^r: \underline{P}^r \to \underline{P}^r$ as follows:

$$f^r(\rho) = \sum_{\underline{t} \in \underline{C_D}} \rho(\underline{t}) f(1_{\underline{t}}), \tag{10.1}$$

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 in (10.1) is with respect to ordinary addition, i.e., the addition in $\underline{P}^r$. Intuitively, the meaning of f^r is that it operates on relations a tuple-at-a-time and it retains duplicates in the result. Applying $|\ |$ on a relation vector removes the duplicates.

The function f^r defined by (10.1) is linear (on the vector space $\underline{P}^r$) whether f is linear or not. (Observe that in (10.1) only the values of f on single tuple vectors $1_{\underline{t}}$ are used.) However, f is recoverable from f^r if and only if f is linear.

Proposition 10.1: For any $A \in \underline{P}$, let $1_A \in \underline{P}$ (and $1_A \in \underline{P}^r$) denote its indicator function, i.e.,

$$1_A(\underline{t}) = \begin{cases} 1 & \text{if } \underline{t} \in A \\ 0 & \text{otherwise} \end{cases}.$$

Then,

$$f(1_A) = |f^r(1_A)| \tag{10.2}$$

if and only if f is linear.

Proof: Suppose f is linear. Then, $f(1_A) = \sum_{t \in A} f(1_t)$, while $f^r(1_A) = \sum_{\underline{t} \in A} f(1_{\underline{t}})$, and (10.2) follows by the definition of $|\ |$.

Conversely, suppose (10.2) holds. Then, the definition of $|\ |$ and (10.1) yield

$$f(1_A) = |\sum_{\underline{t} \in A} f(1_{\underline{t}})| = \sum_{\underline{t} \in A} f(1_{\underline{t}}),$$

which implies that f is linear. □

Intuitively, Proposition 10.1 says that operating on relations a tuple-at-a-time does not affect the result of applying a function on a relation if and only if the function is linear. For a linear f, we shall continue to write it as $f(\rho) = A\rho$, with A a linear operator, and similarly for its extension, $f^r(\rho) = A^r\rho$. We now have the following relationship between A and A^r.

Proposition 10.2: For all integers m and all $\underline{Q}^r \in \underline{P}^r$

$$|(A^r)^m \underline{Q}^r| = A^m |\underline{Q}^r| \tag{10.3}$$

Proof: The proof is by induction on m.

Basis: For m=0, (10.3) yields $|\underline{Q}^r| = |\underline{Q}^r|$.

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

$$|(A^r)^{m+1}\underline{Q}^r| = |A^r((A^r)^m\underline{Q}^r)| = |\sum_{\underline{t} \in \underline{C}_D} ((A^r)^m\underline{Q}^r)(\underline{t})A^r 1_{\underline{t}}| \qquad \text{From (10.1)}$$

$$= |\sum_{\underline{t} \in \underline{C}_D} |(A^r)^m\underline{Q}^r|(\underline{t})A^r 1_{\underline{t}}| \qquad \text{Definition of } |\ |$$

$$= |\sum_{\underline{t} \in \underline{C}_D} (A^m|\underline{Q}^r|)(\underline{t})A^r 1_{\underline{t}}| \qquad \text{Induction hypothesis}$$

$$= |A^r(A^m\underline{Q}^r)| \qquad \text{From (10.1)}$$

$$= A^{m+1}|\underline{Q}^r|. \qquad \text{Induction hypothesis (twice)}$$

□

Proposition 10.2 implies that A^* can be computed from any positive power series of A^r, e.g.,

$$A^* = |e^{A^r}| \quad \text{or} \quad A^* = |(I - \alpha A^r)^{-1}|, \ \alpha > 0.$$

It is an interesting open question whether the above connection can be exploited to computational advantage or not.

Given a function $g: \underline{P} \times \underline{P} \to \underline{P}$, define an extension $g^r: \underline{P}^r \times \underline{P}^r \to \underline{P}^r$, as follows:

$$g^r(\rho,\sigma) = \sum_{\underline{s},\underline{t} \in \underline{C}_D} \rho(\underline{s})\sigma(\underline{t})g(1_{\underline{s}}, 1_{\underline{t}})$$

As in the case of (10.1), g^r is bilinear whether g is bilinear or not, but g is recoverable from g^r if and only if g is bilinear.

Proposition 10.3: For any $A, B \in \underline{P}$, the following relationship holds if and only if g is bilinear:

$$g(1_A, 1_B) = |g^r(1_A, 1_B)|.$$

The proof of Proposition 10.3 is similar to that of Proposition 10.1 and will be omitted.

The bilinear function g^r on the vector space $\underline{P}^r$ defines a multiplication $\circledast^r$ that distributes over addition, i.e.,

$$(a+b) \circledast^r c = a \circledast^r c + b \circledast^r c,$$

$$a \circledast^r (b+c) = a \circledast^r b + a \circledast^r c.$$

It is straightforward to verify that the system $(\underline{P}^r, +, \circledast^r, \underline{\varnothing}, \mathbb{R})$ is a nonassociative algebra without identity on the field $\mathbb{R}$.

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

$$a^1 = a, \; a^k = a \bullet a^{k-1}, \; a \in \underline{P},$$

$$a^{(1)} = a, \; a^{(k)} = a \circledast^r a^{(k-1)}, \; a \in \underline{P}^r.$$

Equation (10.3) implies that $|a^{(k)}| = |a|^k$. The concept of power-associativity can now be extended to $\circledast^r$.

Definition 10.1: The bilinear multiplication $\circledast^r$ on $\underline{P}^r \times \underline{P}^r$ is *power-associative (power-subassociative)* if

$$a^{(k)} \circledast^r a^{(j)} = a^{(k+j)} \quad (\subseteq a^{(k+j)}).$$

Proposition 10.4: The bilinear multiplication $\bullet$ on $\underline{P} \times \underline{P}$ is power-associative (power-subassociative), if its extension $\circledast^r$ is power-associative (power-subassociative).

Proof: Suppose $\circledast^r$ is power-associative. Then, for $a \in \underline{P}$, it is

$$a^m \bullet a^n = |a^{(m)}| \bullet |a^{(n)}| = |a^{(m)} \circledast^r a^{(n)}| = |a^{(m+n)}| = a^{m+n}.$$

For power-subassociativity, we only need to replace the next-to-last equality by $\subseteq$. □

Theorem 10.1: If

$$a^{(2)} \circledast^r a = a^{(3)} \quad \text{and} \quad a^{(2)} \circledast^r a^{(2)} = a^{(4)} \tag{10.4}$$

then $\bullet$ is power-associative.

Proof: It is known that $\circledast^r$, a bilinear multiplication on an algebra on a field with characteristic 0 (like $\mathbb{R}$) is power-associative if and only if (10.4) holds [Scha66]. The desired result of Theorem 10.1 follows from the above fact and Proposition 10.4 about the relationship of the power-associativity property in $\underline{P}$ and $\underline{P}^r$. □

Theorem 10.1 provides an easy-to-test sufficient condition for left-linearizability. Two tests for small m and n are enough to ensure power-associativity. This is a significant computational improvement over the result of Theorem 9.2, which required an infinite number of tests. The only disadvantage of Theorem 10.1 is that the test has to be performed in

the embedded system and not the original one.

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 shall 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 shall 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 shall compare the results presented in this paper with similar ones that have been derived within the logic-based framework, if such results exist. We shall 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 difference between the algebraic and 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. Some of these 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 the ordering stage, the algebra seems the only appropriate tool to explore the complete variety of processing algorithms. Thus, we shall not discuss the ordering stage in any more detail, but we shall concentrate on the rewriting stage.

11.1.1. Linear Recursion

Section 6 describes several results that can be used at the rewritting stage of an optimizer. Theorem 6.1 on the decomposition of $(B+C)^*$ has been observed by Ramakrishnan et al. as well, who in essence used the algebraic notation developed in this paper [Rama89]. 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 [Aho74]. Theorems 6.2 and 6.3 are new and similar in nature with Theorem 6.1. Theorems 6.4 and 6.5 on necessary conditions for decompositions of $(B+C)^*$ make use of the theorem of Sagiv-Yannakakis [Sagi80], which is logic-based in nature.

Theorem 6.6 on a 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 [Han86], 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 [Naug89a].

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:

$$\begin{aligned} \mathbf{P}(x,z) \wedge \mathbf{Q}(z,y) &\rightarrow \mathbf{P}(x,y) \\ \mathbf{Q}(x,y) &\rightarrow \mathbf{P}(x,y), \end{aligned} \tag{11.1}$$

and

$$\begin{aligned} \mathbf{Q}(x,z) \wedge \mathbf{P}(z,y) &\rightarrow \mathbf{P}(x,y) \\ \mathbf{Q}(x,y) &\rightarrow \mathbf{P}(x,y). \end{aligned} \tag{11.2}$$

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

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

Using the above, and assuming that I is nonempty, (11.1) and (11.2) are equivalent to the following two programs:

$$\begin{aligned} \mathbf{P}(x,z) \wedge \mathbf{Q}(z,y) &\rightarrow \mathbf{P}(x,y) \\ \mathbf{I}(u) \wedge \mathbf{Q}(x,y) &\rightarrow \mathbf{P}(x,y), \end{aligned} \tag{11.3}$$

and

$$\mathbf{Q}(x,z) \wedge \mathbf{P}(z,y) \rightarrow \mathbf{P}(x,y) \qquad (11.4)$$

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

If A and B are the corresponding operators for the recursive clauses of (11.3) and (11.4) respectively, and C is the corresponding operator for the nonrecursive clause of both (11.3) and (11.4), 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. The above, together with the fact that the bilinear version of the transitive closure program is easily proven to be associative, which implies that it can be replaced by (11.1) or (11.2), result in the ability to modify any form of the transitive closure program with the given query above into its most efficient form, thus capturing all possible such transformations [Beer87a].

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

$$\mathbf{P}(u,v,w) \wedge \mathbf{R}(u,v,w) \wedge \mathbf{S}(w,x,y,z) \rightarrow \mathbf{P}(x,y,z), \qquad (11.5)$$

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

Rewriting (11.5) according to Theorem 6.9 yields the following program:

$$\mathbf{Q}(u,v,w) \wedge \mathbf{R}(u,v,w) \rightarrow \mathbf{P}'(w)$$

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

$$\mathbf{P}'(t) \wedge \mathbf{S}(t,x,y,z) \rightarrow \mathbf{P}(x,y,z).$$

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, i.e., 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 [Banc86c, Naug89b].

Section 6.4 provides the foundation for selection and projection pushing. As an application example of Theorem 6.10, consider the transitive closure program (11.1) 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 that corresponds to the recursive clause of (11.1), 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 following clause:

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

Also, $\pi A = \pi A_\pi \pi$ (A_π is the same operator as A but accepting as input only the columns specified by π), since both products

are equal to the following clause:

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

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

$$\begin{aligned} &\mathbf{P}'(z) \wedge \mathbf{Q}(z,y) \rightarrow \mathbf{P}'(y) \\ &\mathbf{Q}(c,y) \rightarrow \mathbf{P}'(y) \end{aligned} \tag{11.6}$$

Note that (11.6) is more efficient than (11.1), since the arity of the recursive predicate has been reduced and the query selection is taken into account right from the beginning. Transformations like those of Theorem 6.10 on selection pushing were among the first proposed for recursive programs in database systems [Aho79, Kife85].

In the logic-based approach, projection pushing has been studied by Ramakrishnan et al. [Rama88], who derived several syntactic characterizations to capture the conditions of Theorem 6.10 for pushing projections and Corollary 6.4 for elimination of (recursive) clauses. We have already seen an example of applying Theorem 6.10 in the previous paragraph. We shall now present two examples that are essentially taken from the above work and show that applying Corollary 6.4 produces the same results. Consider the following program-query pair:

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

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

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

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 6.4, $\pi(C(A+B))^* = \pi$, and the above program can be replaced by the nonrecursive

$$\mathbf{Q}_1(x,y) \rightarrow \mathbf{P}(x,y)$$

$$\mathbf{P}(x,\_)?$$

As a second example, consider the following program-query pair:

$$\mathbf{P}_1(x,z,u) \wedge \mathbf{Q}_1(z,u,y) \rightarrow \mathbf{P}(x,y)$$

$$\mathbf{P}_1(x,w,w) \wedge \mathbf{Q}_2(w,z,u) \rightarrow \mathbf{P}_1(x,z,u)$$

$$\mathbf{P}(x,v) \wedge \mathbf{Q}_3(v,z,u) \rightarrow \mathbf{P}_1(x,z,u)$$

$$\mathbf{Q}_4(x,z,u) \rightarrow \mathbf{P}_1(x,z,u)$$

$$\mathbf{P}(x,\_)?$$

If A, B, and C are the operators 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 6.4, $\pi(AB^*C)^*AB^* = \pi AB^*$. The above operator corresponds to the following program, which can be obtained from the original one by removing the third clause, i.e., the one that corresponds to C:

$$\mathbf{P}_1(x,z,u) \wedge \mathbf{Q}_1(z,u,y) \rightarrow \mathbf{P}(x,y)$$

$$\mathbf{P}_1(x,w,w) \wedge \mathbf{Q}_2(w,z,u) \rightarrow \mathbf{P}_1(x,z,u)$$

$$\mathbf{Q}_4(x,z,u) \rightarrow \mathbf{P}_1(x,z,u)$$

$$\mathbf{P}(x,\_)?$$

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. [Rama89], and the performance evaluation by Han and Lu [Han86]. We would also like to mention the work on *magic functions* by Gardarin and Maindreville [Gard86] and Gardarin [Gard87], 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 [Vald88], 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 extensions 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 a linear 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. [Rama89]. 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 sets of tuples, i.e., relations). For example, the corresponding tree for $Q^2 \bullet Q^3$ is shown in Figure 11.1.

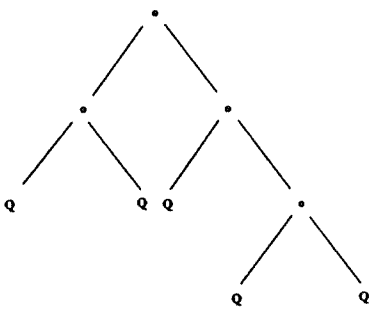

Figure 11.1: Tree representation of $Q^2 \bullet Q^3$

The only other relevant work we are aware of on the problem of linearizability is that by Zhang and Yu [Zhan87]. They focus their attention to a restricted class of bilinear Horn clauses, i.e., a restricted class of multiplications $\bullet$, and for that class they provide a syntactic necessary and sufficient condition for left linearizability. It is rather straightforward to prove (and we shall not do it in this paper) that if a Horn clause satisfies their syntactic condition, then the corresponding multiplication $\bullet$ 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 powerful tools to study these conditions, whereas the logic-based approach, which was employeed by Zhang and Yu, provides syntactic characterizations of them.

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, e.g., magic sets and generalized counting [Banc86b, Beer87b], factoring [Naug89b], and 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, e.g., their parameter relations and the specific manipulations of their columns. We feel that this would only be a change in notation from logic to algebra that is unnecessary, 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 a linear 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 into deriving syntactic characterizations of interesting properties of Horn clause 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, we shall offer characterizations of the properties of bounded recursion [Ioan85, Naug86], linearizability [Zhan87, Sara89], commutativity of selections with arbitrary operators [Deva86], and commutativity of arbitrary operators [Ioan89].

12. CONCLUSIONS

A significant subset of all linear relational operators have been embedded into a closed semiring. Within this algebraic structure, recursive inference by 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 algebraically manipulate an operator representing the query answer has important implications. We have presented several specialized transformations of the query answer at the rewriting stage, 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 secured by our 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. Although alternativeness can be tested within a closed semiring in a finite amount of time, power-associativity cannot. We have overcome this problem, by embedding all bilinear recursions into an algebra. Within that algebraic structure, we have been able to derive a finite test for power-associativity as well.

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

13. REFERENCES

[Aho74]
Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading, MA, 1974.

[Aho79]
Aho, A. and J. Ullman, "Universality of Data Retrieval Languages", in *Proc. of the 6th ACM Symposium on Principles of Programming Languages*, San Antonio, TX, January 1979, pp. 110-117.

[Apt82]
Apt, K. R. and M. H. VanEmden, "Contributions to the Theory of Logic Programming", *JACM* **29**, 3 (July 1982), pp. 841-862.

[Back75]
Backhouse, R. C. and B. A. Carre, "Regular Algebra Applied to Path-Finding Problems", *Journal of the Institute of Mathematics and its Applications* **15**, 2 (April 1975), pp. 161-186.

[Banc85]
Bancilhon, F., "Naive Evaluation of Recursively Defined Relations", in *Proc. of the Islamorada Workshop on Large Scale Knowledge Base and Reasoning Systems*, Islamorada, FL, February 1985.

[Banc86a]
Bancilhon, F. and R. Ramakrishnan, "An Amateur's Introduction to Recursive Query Processing Strategies", in *Proc.*

of the 1986 ACM-SIGMOD Conference on the Management of Data, Washington, DC, May 1986, pp. 16-52.

[Banc86b]
Bancilhon, F., D. Maier, Y. Sagiv, and J. D. Ullman, "Magic Sets and Other Strange Ways to Implement Logic Programs", in *Proc. of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, Cambridge, MA, March 1986, pp. 1-15.

[Banc86c]
Bancilhon, F. and R. Ramakrishnan, "Performance Evaluation of Data Intensive Logic Programs", in *Preprints of the Workshop on Foundations of Deductive Databases and Logic Programming*, Washington, DC, August 1986, pp. 284-314.

[Beer87a]
Beeri, C., P. Kanellakis, F. Bancilhon, and R. Ramakrishnan, "Bounds on the Propagation of Selection in Logic Programs", in *Proc. of the 6th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, San Diego, CA, March 1987, pp. 214-226.

[Beer87b]
Beeri, C. and R. Ramakrishnan, "On the Power of Magic", in *Proc. of the 6th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, San Diego, CA, March 1987, pp. 269-283.

[Carr79]
Carre, B., *Graphs and Networks*, Oxford University Press, Oxford, England, 1979.

[Ceri86]
Ceri, S., G. Gottlob, and L. Lavazza, "Translation and Optimization of Logic Queries: The Algebraic Approach", in *Proc. 12th International VLDB Conference*, Kyoto, Japan, August 1986, pp. 395-402.

[Ceri87]
Ceri, S. and L. Tanca, "Optimization of Systems of Algebraic Equations for Evaluating Datalog Queries, in *Proc. 13th International VLDB Conference*, Brighton, England, September 1987, pp. 31-41.

[Chan77]
Chandra, A. K. and P. M. Merlin, "Optimal Implementation of Conjunctive Queries in Relational Data Bases", in *Proc. 9th Annual ACM Symposium on Theory of Computing*, Boulder, CO, May 1977, pp. 77-90.

[Cloc81]
Clocksin, W. F. and C. S. Mellish, *Programming in Prolog*, Springer Verlag, New York, N.Y., 1981.

[Codd70]
Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", *CACM* **13**, 6 (1970), pp. 377-387.

[Cosm86]
Cosmadakis, S. and P. Kanellakis, "Parallel Evaluation of Recursive Rule Queries", in *Proc. of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, Cambridge, MA, March 1986, pp. 280-293.

[Deva86]
Devanbu, P. and R. Agrawal, *"Some Considerations on Moving Selections into Fixpoint Queries"*, unpublished manuscript, AT&T Bell Laboratories, 1986.

[Eile74]
Eilenberg, S., *Automata, Languages, and Machines*, Academic Press, New York, NY, 1974.

[Gaif87]
Gaifman, H., H. Mairson, Y. Sagiv, and M. Vardi, *"Undecidable Optimization Problems for Database Logic Programs"*, Ithaca, NY, Proc. of the 2nd ACM Symposium on Logic in Computer Science, June 1987, pp. 106-115.

[Gard86]
Gardarin, G. and C. de Maindreville, "Evaluation of Database Recursive Logic Programs as Recurrent Function Series", in *Proc. of the 1986 ACM-SIGMOD Conference on the Management of Data*, Washington, DC, May 1986, pp. 177-186.

[Gard87]
Gardarin, G., "Magic Functions: A Technique to Optimize Extended Datalog Recursive Programs", in *Proc. 13th*

International VLDB Conference, Brighton, England, September 1987, pp. 21-30.

[Han86]
Han, J. and H. Lu, " Some Performance Results on Recursive Query Processing in Relational Database Systems", in *Proc. 2nd International Conference on Data Engineering*, Los Angeles, CA, January 1986, pp. 533-539.

[Hens84]
Henschen, L. and S. Naqvi, "On Compiling Queries in Recursive First-Order Databases", *JACM* **31**, 1 (January 1984), pp. 47-85.

[Hers75]
Herstein, I. N., *Topics in Algebra*, Wiley, New York, N.Y., 1975.

[Ioan85]
Ioannidis, Y. E., "A Time Bound on the Materialization of Some Recursively Defined Views", in *Proc. 11th International VLDB Conference*, Stockholm, Sweden, August 1985, pp. 219-226.

[Ioan86a]
Ioannidis, Y. E., "On the Computation of the Transitive Closure of Relational Operators", in *Proc. 12th International VLDB Conference*, Kyoto, Japan, August 1986, pp. 403-411.

[Ioan86b]
Ioannidis, Y. E. and E. Wong, "An Algebraic Approach to Recursive Inference", in *Proc. of the 1st International Conference on Expert Database Systems*, Charleston, South Carolina, April 1986, pp. 209-223.

[Ioan87]
Ioannidis, Y. E. and E. Wong, "Query Optimization by Simulated Annealing", in *Proc. of the 1987 ACM-SIGMOD Conference on the Management of Data*, San Francisco, CA, May 1987, pp. 9-22.

[Ioan89]
Ioannidis, Y. E., "Commutativity and its Role in the Processing of Linear Recursion", in *Proc. 15th International VLDB Conference (to appear)*, Amsterdam, The Netherlands, August 1989.

[Kife85]
Kifer, M. and E. Lozinskii, *"Query Optimization in Logic Databases"*, Technical report, SUNY, Stonybrook, June 1985.

[Lehm77]
Lehmann, D. J., "Algebraic Structures for Transitive Closure", *Theoretical Computer Science* 4 (1977), pp. 59-76.

[Naug86]
Naughton, J., "Data Independent Recursion in Deductive Databases", in *Proc. of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, Cambridge, MA, March 1986, pp. 267-279.

[Naug88]
Naughton, J., "Compiling Separable Recursions", in *Proc. of the 1988 ACM-SIGMOD Conference on the Management of Data*, Chicago, IL, June 1988, pp. 312-319.

[Naug89a]
Naughton, J., "Minimizing Function-Free Recursive Inference Rules", *JACM* **36**, 1 (January 1989), pp. 69-91.

[Naug89b]
Naughton, J. F., R. Ramakrishnan, Y. Sagiv, and J. D. Ullman, "Factoring Can Reduce Arguments", in *Proc. 15th International VLDB Conference (to appear)*, Amsterdam, The Netherlands, August 1989.

[Rama88]
Ramakrishnan, R., C. Beeri, and R. Krishnamurthy, "Optimizing Existential Datalog Queries", in *Proc. of the 7th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Austin, TX, March 1988, pp. 89-102.

[Rama89]
Ramakrishnan, R., Y. Sagiv, J. D. Ullman, and M. Vardi, *"Proof Tree Transformation Theorems and their Applications"*, Philadelphia, PA, Proc. of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database

Systems, March 1989, pp. 172-181.

[Rose86]
Rosenthal, A., S. Heiler, U. Dayal, and F. Manola, "Traversal Recursion: A Practical Approach to Supporting Recursive Applications", in *Proc. of the 1986 ACM-SIGMOD Conference on the Management of Data*, Washington, DC, May 1986, pp. 166-176.

[Sacc86a]
Sacca, D. and C. Zaniolo, "On the Implementation of a Simple Class of Logic Queries for Databases", in *Proc. of the 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, Cambridge, MA, March 1986, pp. 16-23.

[Sacc86b]
Sacca, D. and C. Zaniolo, "The Generalized Counting Method for Recursive Logic Queries", in *Proc. of the International Conference on Database Theory*, Rome, Italy, October 1986.

[Sagi80]
Sagiv, Y. and M. Yannakakis, "Equivalences Among Relational Expressions with the Union and Difference Operators", *JACM* **27**, 4 (October 1980), pp. 633-655.

[Sara89]
Saraiya, Y. P., *"Linearizing Nonlinear Recursions in Polynomial Time"*, Philadelphia, PA, Proc. of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 1989, pp. 182-189.

[Scha66]
Schafer, R. D., *An Introduction to Nonassociative Algebras*, Academic Press, New York, N.Y., 1966.

[Shap80]
Shapiro, S. and D. McKay, "Inference with Recursive Rules", in *Proc. 1st Annual National Conference on Artificial Intelligence*, Palo Alto, CA, August 1980.

[Tars55]
Tarski, A., "A Lattice Theoretical Fixpoint Theorem and its Applications", *Pacific Journal of Mathematics* **5** (1955), pp. 285-309.

[Vald86]
Valduriez, P. and H. Boral, "Evaluation of Recursive Queries Using Join Indices", in *Proc. of the 1st International Conference on Expert Database Systems*, Charleston, SC, April 1986, pp. 197-208.

[Vald88]
Valduriez, P. and S. Khoshafian, "Transitive Closure of Transitively Closed Relations", in *Proc. of the 2st International Conference on Expert Database Systems*, Tysons Corner, VA, April 1988, pp. 177-185.

[VanE76]
VanEmden, M. H. and R. A. Kowalski, "The Semantics of Predicate Logic as a Programming Language", *JACM* **23**, 4 (January 1976), pp. 733-742.

[Viei86]
Vieille, L., "Recursive Axioms in Deductive Databases: The Query / Subquery Approach", in *Proc. of the 1st International Conference on Expert Database Systems*, Charleston, SC, April 1986, pp. 179-193.

[Zani85]
Zaniolo, C., "The Representation and Deductive Retrieval of Complex Objects", in *Proc. 11th International VLDB Conference*, Stockholm, Sweden, August 1985, pp. 458-469.

[Zhan87]
Zhang, W. and C. T. Yu, "A Necessary Condition for a Doubly Recursive Rule to be Equivalent to a Linear Recursive Rule", in *Proc. of the 1987 ACM-SIGMOD Conference on the Management of Data*, San Francisco, CA, May 1987, pp. 345-356.