COMMUTATIVITY AND ITS ROLE IN THE PROCESSING OF LINEAR RECURSION*

YANNIS E. IOANNIDIS†

▷ We investigate the role of commutativity in query processing of linear recursion. We give a sufficient condition for two linear, function-free, and constant-free rules to commute. The condition depends on the form of the rules themselves. For a restricted class of rules, we show that the condition is necessary and sufficient and can be tested in polynomial time in the size of the rules. Using the algebraic structure of such rules, we study the relationship of commutativity with several other properties of linear recursive rules and show that it is closely related to the important special classes of separable recursion and recursion with recursively redundant predicates. ◁

1. INTRODUCTION

Several general algorithms have been proposed for the processing of recursive programs in data base management systems (DBMSs). Recursive query processing is recognized as an expensive operation, and all the proposed algorithms incur some significant cost. Thus, it is important to identify special cases of recursion on which specialized and more efficient algorithms are applicable. Such special cases of recursion include bounded recursion (uniform or not), transitive closure, separable recursion, and one-sided recursion. In this paper, we elaborate on another special case of recursion, where participating operators (or rules) commute with each other. When this happens, recursive queries can be decomposed into smaller queries, which are expected to have a lower total execution cost than the original query.

*An earlier version of this paper appeared in the *Proceedings of the 15th International VLDB Conference*, Amsterdam, The Netherlands, August 1989, pp. 155–164.

† Partially supported by the National Science Foundation under Grant IRI-8703592.

Address correspondence to Y. Ioannidis, Computer Sciences Department, University of Wisconsin, Madison, WI 53706. Email: YANNIS@CS.WISC.EDU.

Received November 1989; accepted June 1991.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/92/$5.00

Commutativity has already been identified as a significant special case of recursion [13]. Its effect on general algorithms for several types of recursive queries have been studied, as well as how it can be used in conjunction with constants to reduce the amount of data at which the system has to look to answer a query with selections. This earlier work on commutativity was done within the algebraic framework of linear recursive operators (rules) [13]. In this paper, we use the logic representation of rules to derive syntactic conditions for two linear, function-free, and constant-free recursive rules to commute with each other. These conditions are based on the form of the rules themselves and make no direct use of the definition of commutativity, which requires composing the two rules in both ways and examining the two composites for equivalence. For a class of rules for which the conditions are necessary and sufficient, they can be tested in time that is a polynomial in the size of the rules. We also use the algebraic formulation of recursion to compare commutativity with other special classes of recursion, in particular separable recursion and recursion with recursively redundant predicates, and discuss the effects of commutativity on the algorithms proposed for them.

The paper is organized as follows. Section 1 is an introduction. Section 2 is a summary of the algebraic model for linear recursion, which has been introduced elsewhere [13]. In Section 3, we define commutativity in the algebraic model, show its impact on the efficiency of processing recursive rules, and discuss some previous work. In Section 4, we compare the notion of commutativity with separability and recursive redundancy. In Section 5, we use the logic representation of rules and present sufficient conditions for commutativity, which for a restricted class of rules are necessary and sufficient. In Section 6, separability and recursive redundancy are reexamined for the restricted class of rules studied in Section 5. Finally, Section 7 contains our conclusions and some directions for future work.

2. ALGEBRAIC MODEL

In this section, we provide a summary of the algebraic model for linear recursion [13]. We use the terms relation and predicate indistinguishably. Consider the following pair of one linear recursive and one nonrecursive rule:

$$\boldsymbol{P}\left(\underline{\boldsymbol{x}}^{(k+1)}\right) :- \boldsymbol{P}\left(\underline{\boldsymbol{x}}^{(0)}\right) \wedge \boldsymbol{Q}_1\left(\underline{\boldsymbol{x}}^{(1)}\right) \wedge \cdots \wedge \boldsymbol{Q}_k\left(\underline{\boldsymbol{x}}^{(k)}\right), \tag{2.1}$$

$$\boldsymbol{P}\left(\underline{\boldsymbol{x}}^{(k+1)}\right) :- \boldsymbol{Q}\left(\underline{\boldsymbol{x}}^{(k+1)}\right), \tag{2.2}$$

where for each i, $\underline{\boldsymbol{x}}^{(i)}$ is a vector of variables. No restriction is imposed on the form of the rule or on the finiteness of the relations corresponding to the various predicate symbols in the rule. Thus, for example, rules that contain functions can be expressed in the above form. Each one of $\boldsymbol{P}(\underline{\boldsymbol{x}}^{(0)})$, $\boldsymbol{Q}(\underline{\boldsymbol{x}}^{(k+1)})$, and the $\boldsymbol{Q}_i(\underline{\boldsymbol{x}}^{(i)})$'s is a (*positive*) *literal*. Without loss of generality, we assume a typeless system, so that the *schema* of a relation is defined as the number of its argument positions.

Operationally, (2.1) can be represented by a function $f(\boldsymbol{P}, \{\boldsymbol{Q}_i\})$ that has $\{\boldsymbol{Q}_i\}$ as parameters and accepts as input and produces as output relations of the same schema as $\boldsymbol{P}$: $f(\boldsymbol{P}, \{\boldsymbol{Q}_i\}) \subseteq \boldsymbol{P}$. The function f can be thought of as a linear relational operator applied to the recursive relation $\boldsymbol{P}$ to produce another relation of the same schema. Let R be the set of all such operators. We can establish an algebraic framework in which we can define operations on relational operators as follows.

Multiplication of operators is defined by $(A*B)\boldsymbol{P} = A(B\boldsymbol{P})$ and *addition* by $(A + B)\boldsymbol{P} = A\boldsymbol{P} \cup B\boldsymbol{P}$.[1] For notational convenience we omit the operator $*$. Also, because $+$ and $*$ are associative, we often omit the parentheses around them, assuming right associativity. The multiplicative identity ($1\boldsymbol{P} = \boldsymbol{P}$) and the additive identity ($0\boldsymbol{P} = \varnothing$, where $\varnothing$ is the empty set) are defined in obvious ways. The nth power of an operator A is inductively defined as: $A^0 = 1$, $A^n = A*A^{n-1} = A^{n-1}*A$. Equality of operators in R is defined as $A = B \Leftrightarrow \forall \boldsymbol{P}$, $A\boldsymbol{P} = B\boldsymbol{P}$. Finally, a partial order $\leq$ is defined on R as $A \leq B \Leftrightarrow \forall \boldsymbol{P}$, $A\boldsymbol{P} \subseteq B\boldsymbol{P}$. The set R with the above defined operations forms a *closed semi-ring* [13].

The above embedding of the linear relational operators in a closed semi-ring allows the rewriting of the set of Horn clauses (2.1) and (2.2) (assuming that A is the operator that corresponds to (2.1)) as

$$A\boldsymbol{P} \subseteq \boldsymbol{P},$$

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

The minimal solution of the system is the minimal solution of the equation

$$\boldsymbol{P} = A\boldsymbol{P} \cup \boldsymbol{Q}. \tag{2.3}$$

The solution is a function of $\boldsymbol{Q}$. Hence, $\boldsymbol{P}$ can be written as $\boldsymbol{P} = B\boldsymbol{Q}$, and the problem becomes one of finding the operator B. Manipulation of (2.3) results in the elimination of $\boldsymbol{Q}$, 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 the following:

(a) $1 + AB = B$, (2.4)
(b) B is minimal with respect to (a), i.e., for all C, $1 + AC = C \Rightarrow B \leq C$.

Theorem 2.1. [13] *The solution of equation* (2.4*a*) *with restriction* (2.4*b*) *is* $A^* = \sum_{k=0}^{\infty} A^k$.

The operator A^* is called the *transitive closure* of A. In the data base context, theorems similar to Theorem 2.1 were first derived by Aho and Ullman [4]. The unique characteristic of Theorem 2.1 as described above is that the solution of (2.4) is expressed in an explicit algebraic form within an algebraic structure like the closed semi-ring of linear relational operators. The implications of the manipulative power thus afforded on the implementation of A^* are significant [11–13]. In this paper, we concentrate on the implications of commutativity of operators in the implementation of A^*.

Note that, although an operator A may be derived from a recursive rule, the operator itself is nonrecursive, i.e., it corresponds to a conjunctive query [8]. Also note that A^* represents an operator. The query answer is the result of applying A^* to a given relation $\boldsymbol{Q}$. This is only an abstraction, however, that allows us to study recursion within the closed semi-ring of relational operators. It poses no restriction

[1] The above definitions are valid only if the operators involved are appropriately compatible, e.g., for $+$, the operators have to agree on the schema of their input and the schema of their output. Although in the rest of the paper we only deal with appropriately compatible operators, the general algebraic theory incorporates all operators [13].

whatsoever in the processing order of the query, i.e., it does not enforce that A^* is computed first and then it is applied to Q. For example, assume that A^* can be decomposed into B^* and C^*, i.e., $A^* = B^*C^*$, so that the final computation is B^*C^*Q. The computation may proceed by first computing C^*, then applying it to Q, and then using *semi-naive* [5] with B as the basic operator and (C^*Q) as the initial relation. The significance of the algebraic formulation lies in the abstraction that it offers, within which the capability of the decomposition $A^* = B^*C^*$ can be exhibited.

3. COMMUTATIVITY

3.1. Definitions and Motivation

We say that two operators B and C *commute* if $BC = CB$. Consider computing A^*, the transitive closure of A, where $A = B + C$. It has been shown that if $CB \leq B^kC^l$, for some k, l with $k \in \{0, 1\}$ or $l \in \{0, 1\}$, then $A^* = B^*C^*$ [13]. Commutativity is a special case of this condition. The computation of A^* is decomposed into two smaller computations, those of B^* and C^* (plus an additional multiplication of them). The complexity of B and C is smaller than that of A. In general, this is expected to affect the total cost of the computation significantly. To see this observe that the following always holds:

$$(B + C)^* = B^*C^* + (B + C)^*CB(B + C)^*. \tag{3.1}$$

This formula expresses the fact that the terms of the series that corresponds to $(B + C)^*$ can be partitioned into those that do not have CB in them and those that do. In general, all such terms need to be computed. If the condition that was mentioned at the beginning of this section holds, however, then the second set of terms does not need to be computed, because it is known that it can only produce duplicates. Unfortunately, this is not enough to prove that computing B^*C^* is more efficient than computing $(B + C)^*$. In an actual implementation, several parameters affect performance, and their complex interactions can rarely be studied analytically, e.g., main memory size, buffer replacement strategies, and availability of indices. For example, the computation of B^*C^* is likely to be cheaper than that of $(B + C)^*$ because main memory can be used more efficiently when computing the transitive closure of smaller operators (recall that $B \leq B + C$ and $C \leq B + C$), but this is hard to quantify.

One aspect of performance that is tractable is the number of duplicate tuples produced by an algorithm. Quite often, especially in recursive computations, duplicate production and elimination has been shown to dominate the cost of an algorithm [1]. Comparing the computations of B^*C^* and $(B + C)^*$ with respect to duplicates, we derive the superiority of B^*C^* by the following general result.

Theorem 3.1. Let $\{A_i\}$, $\{B_i\}$, $\{C_i\}$ *be sets of linear operators such that every operator in* $\{A_i\}$ *and* $\{B_i\}$ *is a product of operators in* $\{C_i\}$, *and if* $(C_1 \cdots C_{k-1}C_k) \in \{A_i\}$ (*or* $\{B_i\}$) *then* $(C_1 \cdots C_{k-1}) \in \{A_i\}$ (*or* (B_i)) *as well. Consider two linear operators A and B, where* $A = B$, $A = \Sigma_i A_i$, *and* $B = \Sigma_i B_i$. *Let* Q *be an arbitrary relation and*

$\boldsymbol{T}$ be equal to $\boldsymbol{T} = A\boldsymbol{Q} = B\boldsymbol{Q}$. If $\{A_i\} \subseteq \{B_i\}$, then the evaluation of $\boldsymbol{T}$ based on A produces no more duplicates than its evaluation based on B.

PROOF. Let the *derivation graph* of a computation of $\boldsymbol{T}$ be a labeled directed graph $G = (V, E, L: E \to \{C_i\})$, where the set of nodes V, the set of arcs E, and the label function L from E to the set of operators $\{C_i\}$ are defined as follows:

$V = \boldsymbol{T}$, i.e., the nodes of G are the tuples in $\boldsymbol{T}$,

$E = \{(t_1 \to t_2) | t_2$ is produced by applying one operator from $\{C_i\}$ on $t_1\}$,

$L((t_1 \to t_2)) = C$, where $C \in \{C_i\}$ and t_2 is produced by applying C on t_1.

Since there is a 1 to 1 correspondence between nodes and tuples, we use the two terms indistinguishably. We assume a model of computation that starts at the tuples in $\boldsymbol{Q}$ and traverses the graph until all nodes are visited at least once. We also assume that the same tuple is not derived through the same arc more than once.[2] Such a computation can be achieved, for example, by employing the semi-naive evaluation [5].

A path in the graph from a tuple s in $\boldsymbol{Q}$ to a tuple t represents a derivation of t starting from s. The concatenation of the labels of the arcs along the path represent a product of the corresponding operators in $\{C_i\}$ that is equal to one of the operators in $\{A_i\}$ or $\{B_i\}$. No derived tuple has zero in-degree, i.e., every derived tuple is always connected to some tuple in $\boldsymbol{Q}$. Each tuple is derived as many times as there are arcs entering it. Thus, the number of tuple derivations during a computation, which is the sum of the number of tuples in $\boldsymbol{T}$ plus the number of duplicates produced, is equal to the sum of the in-degrees of the nodes in the graph that corresponds to the computation, i.e., it is equal to $|E|$. If $\{A_i\} \subseteq \{B_i\}$, the graph corresponding to A has the same set of nodes but is possibly missing some of the arcs of the graph corresponding to B. In that case, some nodes have lower in-degree in the graph of A than in the graph of B, which implies that computing $\boldsymbol{T}$ based on A will produce less duplicates than computing it based on B. □

We would like to elaborate on the result of Theorem 3.1 briefly. Consider the derivation graph for the computation of $\boldsymbol{T}$ based on B. If that computation is duplicate-free, then all nodes have in-degree equal to 1, and no improvement can be made. Only arcs that lead into nodes with in-degrees that are higher than 1 can be removed from the graph of B to construct the graph of A. In that case, i.e., when the terms in $\{B_i\} - \{A_i\}$ do produce tuples when applied to $\boldsymbol{Q}$, the computation based on A is more efficient than the computation based on B.

Theorem 3.1 shows that it is important to be able to identify when two operators commute, since commutativity allows decompositions of the form $(B + C)^* = B^*C^*$. It is applicable in this case with B^*C^*, $(B + C)^*$, and $\{B, C\}$ playing the roles of A, B, and $\{C_i\}$ respectively in its statement. Therefore, in several cases, using B^*C^* instead of $(B + C)^*$ decreases the number of produced duplicates. In Section 5, we present a sufficient condition for commutativity, which for rules of some restricted form is shown to be necessary and sufficient.

[2] We do not take into account any computation steps that fail to produce a tuple. Such computation steps are not represented in the graph and their cost is not captured.

3.2. Previous Work

Commutativity or properties related to it have been rarely addressed in the past. The earliest result that we are aware of that is related to commutativity is by Lassez and Maher [14]. Their interest in commutativity was mostly with respect to certain decompositions that can be achieved when computing the transitive closure of the sum of multiple operators. They obtained two main results that are related to commutativity. In algebraic form, they are expressed as follows:

$$B^*C^* = C^*B^* = B^* + C^* \Leftrightarrow (B + C)^* = B^* + C^*,$$

$$BC = CB = B + C \Rightarrow (B + C)^* = B^* + C^*.$$

The above results are easily generalized for an arbitrary number of operators.

A syntactic sufficient condition for commutativity has been presented by Ramakrishnan, Sagiv, Ullman, and Vardi [19]. Their condition is less general than the one presented in Section 5 and, therefore, fails to be necessary and sufficient for the class that ours is. It is always tested in polynomial time, however. Deriving this sufficient condition was part of a study of proof-tree transformations. (Commutativity can be seen as a proof-tree transformation if operators are represented as proof trees). Among other results, that study led to an independent discovery of the above mentioned fact that if $CB \leq B^kC^l$, for some k, l with $k \in \{0, 1\}$ or $l \in \{0, 1\}$, then $(B + C)^* = B^*C^*$.

Finally, Dong has examined several possible decompositions of the transitive closure of the sum of multiple operators [9]. The only result that involves commutativity in a significant way can be expressed as follows in algebraic form:

$$B^*C^* = C^*B^* \Leftrightarrow (B + C)^* = B^*C^* = C^*B^*.$$

4. COMMUTATIVITY VS. SEPARABILITY AND RECURSIVE REDUNDANCY

4.1. Commutativity vs. Separability

The separable algorithm has been introduced by Naughton as an efficient processing method for some special class of linear recursive rules [15]. For the sake of simplicity, it is presented below in its specialized form for two operators A_1 and A_2. Extensions of it to an arbitrary number of operators are straightforward. Consider an initial relation $\boldsymbol{q}$ and a selection σ on arguments of either the parameter relations of A_2 or its input. The separable algorithm corresponds to the algebraic formula $A_1^* (\sigma A_2^*)\boldsymbol{q}$ and is given below in pseudo-code form. The variables B and C contain operators, whereas the variables $\boldsymbol{R}$ and $\boldsymbol{S}$ contain relations. Multiplication of operators is shown explicitly for readability.

Algorithm 4.1. The separable algorithm:

$B := \sigma$;
$C := \sigma$;
repeat
    $B := B * A_2$;
    $C := B + C$;

until C does not change
$\boldsymbol{R} := C\boldsymbol{q}$;
$\boldsymbol{S} := \boldsymbol{R}$;
repeat
    $\boldsymbol{R} := A_1\boldsymbol{R}$;
    $\boldsymbol{S} := \boldsymbol{S} \cup \boldsymbol{R}$;
until $\boldsymbol{S}$ does not change.

The first loop actually involves manipulating relations that are parameters of the various operators. Moreover, in every application of an operator inside each loop, only the new tuples produced in the previous iteration are used. The following theorem shows that the efficient separable algorithm is applicable to the class of commutative recursions.

Theorem 4.1. Given two operators A_1 and A_2 that commute and a selection σ that commutes with one of them, the equality $\sigma(A_1 + A_2)^* = A_1^*(\sigma A_2^*)$ holds, i.e., the separable algorithm can be used for the computation of $\sigma(A_1 + A_2)^*$.

PROOF. Let $A_1A_2 = A_2A_1$. The transitive closure of the sum of A_1 and A_2 is given by $(A_1 + A_2)^* = A_1^*A_2^*$ [13]. Given a query with a selection σ that commutes with A_1, an easy induction on the power of A_1 yields the result:

$$\sigma(A_1 + A_2)^* = A_1^*(\sigma A_2^*). \qquad \square$$

The significance of the above theorem can be realized only in conjunction with additional results that are presented in Section 6. Essentially, its importance lies in the fact that it widens the class of recursive rules on which the separable algorithm is applicable.

We should also note that, although Theorem 4.1 deals with two operators, the result can be generalized. 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:

$$\sigma_0\sigma_1\sigma_2 \cdots \sigma_n(A_1 + A_2 + \cdots + A_n)^* = (\sigma_1 A_1^*)(\sigma_2 A_2^*) \cdots (\sigma_n A_n^*)\sigma_0.$$

Usually, most of the selections are not present. In the presence of multiple selections, it is an interesting optimization problem to choose the order in which the various operators will be computed and the time when an operator will be applied to the input relation.

4.2. Commutativity vs. Recursive Redundancy

The class of recursions that contain recursively redundant predicates has also been introduced by Naughton [16]. Consider an operator A that is equal to the product BCD, i.e., $A = BCD$. In general, every term in the series $A^* = \sum_{k=0}^{\infty} A^k$ is an arbitrary product involving B, C, and D. Operator C is *recursively redundant* in A^* if there is some N such that each term in the series of A^* is equal to a product containing C less than N times. The nonrecursive predicates appearing in C as parameters are also called recursively redundant. Before stating the main result of this subsection we need the following definitions. An operator B is *uniformly bounded*, if there exist K and N, $K < N$, such that $B^N \le B^K$. An operator B is *torsion*, if there exist K and N, $K < N$, such that $B^N = B^K$. Clearly, every torsion is

uniformly bounded, but the opposite is not true in general. The effect of the presence of recursively redundant operators on the query processing algorithm of an operator is given by the following result, which is actually a generalization of an earlier result on the subject [13]. (Without loss of generality, we assume that all operators have the same domain and range, so that the product of any pair of them is well defined.)

Theorem 4.2. Let $\boldsymbol{Q}$ *be a parameter relation of some operator* A. *If there exist* $L \geq 1$ *and operators* B *and* C *such that* $\boldsymbol{Q}$ *is a parameter of* C *but not of* B, C *is torsion,*

$$A^L = BC^L, \quad \text{and} \quad C^L(BC^L) = C^L(C^LB), \tag{4.1}$$

then $\boldsymbol{Q}$ *is recursively redundant in* A^*.

PROOF. Consider a relation $\boldsymbol{Q}$ that satisfies the premises of the theorem. Let $K > 0$ and $N > 0$, $K < N$, be the smallest numbers such that $C^N = C^K$. The above equality implies that $C^{KL} = C^{NL}$ holds as well. It takes an easy induction to show that

$$C^{mL} = C^{(m+i(N-K))L}, \text{ for all } K \leq m < N \text{ and all } i \geq 0. \tag{4.2}$$

The main result follows from the derivation below:

$$\begin{aligned}
A^* &= \sum_{m=0}^{KL-1} A^m + \sum_{m=KL}^{\infty} A^m \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{n=0}^{L-1} A^n\right)\left(\sum_{m=K}^{\infty} A^{mL}\right) \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{n=0}^{L-1} A^n\right)\left(\sum_{m=K}^{\infty} (BC^L)^m\right) \\
&\qquad\qquad [\text{From the first equality of (4.1)}] \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{n=0}^{L-1} A^n\right)\left(\sum_{m=K}^{\infty} BC^{mL}B^{m-1}\right) \\
&\qquad\qquad [\text{From the second equality of (4.1)}] \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{n=0}^{L-1} A^n\right)\left(\sum_{m=K}^{N-1} BC^{mL}\right)\left(\sum_{i=0}^{\infty} B^{m-1+i(N-K)}\right) \\
&\qquad\qquad [\text{From (4.2)}] \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{n=0}^{L-1} A^n\right)\left(\sum_{m=K}^{N-1} BC^{mL}B^{m-1}\right)\left(\sum_{i=0}^{\infty} B^{i(N-K)}\right) \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{n=0}^{L-1} A^n\right)\left(\sum_{m=K}^{N-1} A^{mL}\right)\left(\sum_{i=0}^{\infty} B^{i(N-K)}\right) \\
&\qquad\qquad [\text{From the first equality of (4.1)}] \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{m=KL}^{NL-1} A^m\right)\left(\sum_{i=0}^{\infty} B^{i(N-K)}\right) \\
&= \sum_{m=0}^{KL-1} A^m + \left(\sum_{m=KL}^{NL-1} A^m\right)(B^{(N-K)})^*.
\end{aligned}$$

Note that C^{NL-1} is the highest power of C used in any term of A^*: since $\boldsymbol{Q}$ is not a parameter of B, the latter cannot contain C as a factor either. Thus, C is recursively redundant, and so is $\boldsymbol{Q}$ as one of its parameter relations. Clearly, the above formula corresponds to a more efficient algorithm than processing A as a whole, since C is processed only for a fixed finite number of times, i.e., $NL-1$, beyond which only B is processed. □

5. CHARACTERIZATION OF COMMUTATIVITY

We now turn our attention to commutativity as expressed in a logic framework. We restrict ourselves to linear, function-free, and constant-free recursive rules. If a variable appears in the consequent of a rule, it is called *distinguished*, otherwise it is called *nondistinguished*. We assume that the rules have the same consequent and share no nondistinguished variables. Moreover, repeated variables in the consequent are replaced by distinct ones, while adding the appropriate equality predicates in the antecedent. Finally, although the original task is to compute the transitive closure of two recursive rules with the same consequent, we are interested in the commutativity of the underlying nonrecursive rules, i.e., conjunctive queries. Given a linear recursive rule whose recursive predicate is $\boldsymbol{P}$, its underlying nonrecursive one is constructed by replacing the instance of $\boldsymbol{P}$ in its consequent by $\boldsymbol{P}_O$ (output), and its instance of $\boldsymbol{P}$ in its antecedent by $\boldsymbol{P}_I$ (input). However, we are still referring to these two predicates as instances of the recursive predicate.

Given two nonrecursive rules r and s, a *homomorphism* f: $r \rightarrow s$ is a mapping from the variables of r into those of s, such that (i) if x is a distinguished variable then $f(x)=x$, and (ii) if $Q(x_1,\ldots,x_n)$ appears in the antecedent of r, then $Q(f(x_1),\ldots,f(x_n))$ appears in the antecedent of s. Homomorphisms are directly related to the partial order of rules defined in Section 2 (for the corresponding operators). In particular, s *is contained* in r (i.e., given any relations for the predicates in the antecedents of r and s, the output relation produced by s for the predicate in its consequent is a subset of the one produced by r), denoted by $s \leq r$, iff there exists a homomorphism f from r to s [3, 8]. Also, s is *equivalent* to r, denoted by $s=r$, iff $s \leq r$ and $r \leq s$.

Given two rules r_1 and r_2, the *composite* of r_1 with r_2, denoted by $r_1 r_2$ is defined as the result of resolving the consequent of r_2 with the literal of the recursive predicate in the antecedent of r_1.[3] We say that two rules r_1 and r_2 with the same consequent *commute*, if composing r_1 with r_2 and composing r_2 with r_1 yield equivalent rules. This, in turn, is equivalent to the existence of homomorphisms from each composite to the other. Clearly, the definition of commutativity suggests a straightforward algorithm to test it for two rules r_1 and r_2: form the two composites $r_1 r_2$ and $r_2 r_1$ and test their equivalence. Unfortunately, a polynomial time implementation of this algorithm is unlikely to exist, since equivalence of conjunctive queries is known to be an NP-complete problem [3, 8].

5.1. A Sufficient Condition

In this section, we give a sufficient condition for commutativity that avoids producing the two composites. The condition can be tested in exponential time,

[3] Analogously, the composite of a rule r with itself n times is denoted by r^n.

because it potentially involves testing for equivalence of conjunctive queries. The test, however, is still more efficient than the one based on the definition of commutativity, because its exponential part is only occasionally applied on parts of the original rules as opposed to always being applied on the composites of the two rules.

For a rule r, we define the function h from the set of distinguished variables in r to the set of all variables in r. For a distinguished variable x, $h(x)$ is the variable that appears in the recursive predicate in the antecedent in the same position as x appears in the consequent. Since distinguished variables are assumed to appear exactly once in the consequents of rules (with the potential of repeated variables being realized by equalities in the antecedent), h is a function. We may also define powers of h as

$$h^1(x) = h(x), \quad \text{and} \quad h^n(x) = h(h^{n-1}(x)) \quad \text{if} \quad h^{n-1}(x) \text{ is distinguished.}$$

For two rules r_1 and r_2, the corresponding h functions are denoted by h_1 and h_2 respectively. We also define two more functions, g_{12} on the variables of r_2 and g_{21} on the variables of r_1. Since the two rules are assumed to share no nondistinguished variable, the former is defined as

$$g_{12}(z) = \begin{cases} z & z \text{ is nondistinguished} \\ h_1(z) & z \text{ is distinguished,} \end{cases}$$

and the latter is defined similarly. By definition, when $r_1 r_2$ is formed, a variable z in a predicate of r_2 is always replaced by $g_{12}(z)$.

As a notation vehicle for the theorems to follow, we use a version of the α-graph of a rule (also called α-graph), which was introduced for the study of uniform boundedness [10]. The α-graph of a rule is defined as follows:

(i) There is a node in the graph for every variable in the rule.
(ii) If two variables x, y appear in two consecutive argument positions of some nonrecursive predicate $\boldsymbol{Q}$ in the rule, a *static* directed arc$(x \rightarrow y)$ is put in the graph between the corresponding two nodes x, y. Also, if x appears in a unary nonrecursive predicate $\boldsymbol{Q}$ in the rule, a static directed arc$(x \rightarrow x)$ is put in the graph. In both cases, the label of the edge is $\boldsymbol{Q}$. (Static arcs are shown as thin lines in all forthcoming figures.)
(iii) If two variables x, y appear in the same position of the recursive relation $\boldsymbol{P}$ in the antecedent and the consequent respectively, then a *dynamic* directed arc$(x \rightarrow y)$ is put in the graph from node x to node y. (Dynamic arcs are shown as thick lines in all forthcoming figures.)

Several characteristics of the underlying undirected graph of the α-graph of a rule are important, e.g., connected components. In the sequel, although these characteristics are defined for undirected graphs, we use them for directed ones as well, with the understanding that we always refer to the corresponding underlying undirected graphs.

It is also important to partition the distinguished variables of a rule in the following categories (in the sequel, due to part (i) of the definition of the α-graph of a rule, we use the terms variable and node indistinguishably). Consider a set of

variables $\{x_i\}$, $0 \le i \le n-1$, $n \ge 1$, such that x_i appears in the same argument position of the recursive predicate in the antecedent as $x_{(i+1) \bmod n}$ appears in the recursive predicate in the consequent (i.e., the positions of the variables in the antecedent is a permutation of their positions in the consequent). Any such variable is called *persistent* and in particular *n-persistent* (n is the cardinality of the set). More specifically, if no variable from the set appears anywhere else in the rule, every variable in the set is called *free n-persistent*. Otherwise, every variable in the set is called *link n-persistent*. All other variables are called *general*. Note that free n-persistent variables, $n \ge 1$, are the only variables in their connected component in the α-graph, connected only via dynamic arcs of the form $(x_i \to x_{(i+1) \bmod n})$.

Finally, we need to define some interesting subgraphs of the α-graph of a rule [7]. Consider an undirected graph G, a subset E' of its edges, and let G' be the subgraph of G induced by E'. Let V' be the node set of G'. Define a relation $\sim$ on the edges of $G - E'$ by the condition that, for two edges e_1 and e_2, $e_1 \sim e_2$, if $e_1 = e_2$ or there is a walk in G that contains e_1 and e_2 but contains no node from V' as an internal node (although the walk may start or end at nodes in V'). It is easy to verify that $\sim$ is an equivalence relation on the edges of G. The subgraph of G induced by the edges of an equivalence class under the relation $\sim$ is called a *bridge* of G with respect to G'. A bridge together with the part of G' that is connected to the bridge forms an *augmented bridge*. In the sequel, unless otherwise noted, whenever we refer to bridges in the α-graph of a rule, we mean its bridges with respect to its subgraph induced by the dynamic arcs connecting each link 1-persistent variable in the graph to itself. This is because they play a very important role in the study of commutativity and we refer to them continuously.

Let G' be a subgraph of the α-graph of a rule r and V' be its node set such that, for any distinguished variable $x, x \in V' \Rightarrow h(x) \in V'$. Then, any augmented bridge with respect to G' is an α-graph itself and corresponds to specific parts of the original rule. Thus, there is a unique *narrow* rule r_n that corresponds to such an augmented bridge. Its nonrecursive predicates in the antecedent are the ones of r that correspond to the static arcs of the augmented bridge. Its instances of the recursive predicate in the consequent and the antecedent are formed from the ones of r by projecting on the argument positions that contain in the consequent distinguished variables that appear in the augmented bridge. In addition, there is another unique *wide* rule r_w that corresponds to such an augmented bridge. Its only difference from r_n is that its recursive predicate has the same arity as in r, with the additional distinguished variables being free 1-persistent. Clearly, both rules constructed as above are unique, since they depend on a specific augmented bridge of the α-graph of a specific rule. Moreover, the α-graph of the narrow rule is the augmented bridge from which it was constructed. Thus, containment and equivalence of augmented bridges can be defined appropriately as containment and equivalence of the corresponding narrow rules.

Example 5.1. Figure 1 is the α-graph of the rule:

$$\boldsymbol{P}(u,v,w,x,y,z) :- \boldsymbol{P}(v,u,w,w,y,z) \wedge \boldsymbol{Q}(x,y).$$

Variable z is free 1-persistent, variables w and y are link 1-persistent, variables u and v are free 2-persistent, and variable x is general.

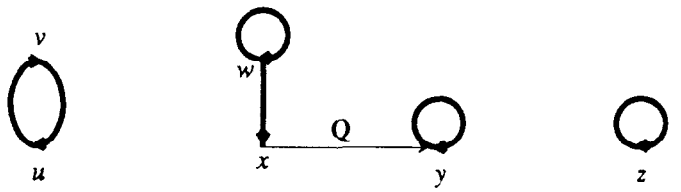

FIGURE 1. Example of an α-graph.

For another example, see Figure 2, the α-graph of the rule

$$P(v,w,x,y,z) :- P(v,v,u,y,y) \wedge Q(v,u,y) \wedge R(w) \wedge S(x) \wedge T(z).$$

Variables v and y are link 1-persistent. The augmented bridges of G with respect to the graph induced by the arcs $(v \rightarrow v)$ and $(y \rightarrow y)$ have been enclosed in dotted boundaries. Their corresponding narrow rules are

$$P(v,w) :- P(v,v) \wedge R(w),$$
$$P(v,x,y) :- P(v,u,y) \wedge Q(v,u,y) \wedge S(x),$$
$$P(y,z) :- P(y,y) \wedge T(z),$$

whereas their corresponding wide rules are

$$P(v,w,x,y,z) :- P(v,v,x,y,z) \wedge R(w),$$
$$P(v,w,x,y,z) :- P(v,w,u,y,z) \wedge Q(v,u,y) \wedge S(x),$$
$$P(v,w,x,y,z) :- P(v,w,x,y,y) \wedge T(z). \quad \square$$

The following theorem gives a sufficient condition for commutativity of rules of the form specified in the beginning of Section 5. Another, less general, sufficient condition for commutativity has been independently discovered and reported elsewhere [19].

Theorem 5.1. Two rules r_1 and r_2 with the same consequent commute if every distinguished variable x satisfies one of the following:

(a) *x is free 1-persistent in r_1 or r_2;*
(b) *x is link 1-persistent in both r_1 and r_2;*
(c) *x is free m_1-persistent, $m_1 > 1$, in r_1 and free m_2-persistent, $m_2 > 1$, in r_2 and $h_1(h_2(x)) = h_2(h_1(x))$;*
(d) *x is link m-persistent, $m > 1$, or general, and belongs to equivalent augmented bridges in both r_1 and r_2.*

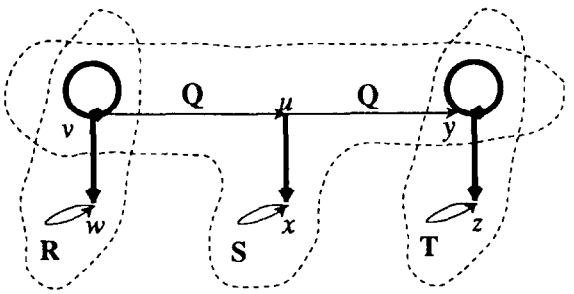

FIGURE 2. Augmented bridges in an α-graph.

PROOF. In the proof, we use the fact that commutativity of r_1 and r_2 is defined as the equivalence of r_1r_2 and r_2r_1, which in turn, is equivalent to the existence of homomorphisms from each composite to the other. Recall that we assume that the two rules have the same consequents and share no nondistinguished variables. Given that (a), (b), (c), and (d) hold for r_1 and r_2, we can partition their distinguished variables into the following vectors:

$\underline{p}_i$ vector of the free 1-persistent variables in r_i, $i = 1, 2$;

$\underline{s}$ vector of the common link 1-persistent variables in r_1 and r_2;

$\underline{c}$ vector of the common free persistent variables in the consequent of r_1 and r_2 that are free m_i-persistent, $m_i > 1$, in r_i, $i = 1, 2$;

$\underline{e}$ vector of the link m-persistent, $m > 1$, and general variables that belong to equivalent augmented bridges in r_1 and r_2.

Without loss of generality, the variables in the consequents of the two rules are grouped so that they can be written in the following form:

$$r_1\colon\ \boldsymbol{P}_O(\underline{p}_1, \underline{p}_2, \underline{s}, \underline{c}, \underline{e}) :- \boldsymbol{P}_I\big(\underline{p}_1, \underline{z}_1, \underline{s}, h_1(\underline{c}), \underline{v}_1\big) \wedge \boldsymbol{S}(\underline{u}_1) \wedge \boldsymbol{Q}_1(\underline{w}_1),$$

$$r_2\colon\ \boldsymbol{P}_O(\underline{p}_1, \underline{p}_2, \underline{s}, \underline{c}, \underline{e}) :- \boldsymbol{P}_I\big(\underline{z}_2, \underline{p}_2, \underline{s}, h_2(\underline{c}), \underline{v}_2\big) \wedge \boldsymbol{S}(\underline{u}_2) \wedge \boldsymbol{Q}_2(\underline{w}_2).$$

We have assumed that every rule seen as a conjunctive query is in its unique minimal form [8, 18]. This has the implication that the augmented bridges that are equivalent in the two rules are isomorphic (i.e., they are the same up to reordering of their nonrecursive predicates and renaming of their nondistinguished variables), so that their nonrecursive predicates can be represented by a common $\boldsymbol{S}$. $\boldsymbol{Q}_1, \boldsymbol{Q}_2$ represent the remaining nonrecursive predicates, i.e., those of bridges whose general and link m-persistent variables, $m > 1$, in one rule are free 1-persistent in the other. Finally, $\underline{z}_1$, $\underline{z}_2$, $\underline{v}_1$,$\underline{v}_2$, $\underline{w}_1$, $\underline{w}_2$ are vectors of variables. In particular,

$\underline{z}_1$ it contains nondistinguished variables and variables from $\underline{p}_2$;

$\underline{z}_2$ it contains nondistinguished variables and variables from $\underline{p}_1$;

$\underline{w}_1$ it contains nondistinguished variables and variables from $\underline{p}_2$ and $\underline{s}$;

$\underline{w}_2$ it contains nondistinguished variables and variables from $\underline{p}_1$ and $\underline{s}$;

$\underline{v}_1, \underline{v}_2$ they contain nondistinguished variables and variables from $\underline{e}$ and $\underline{s}$;

$\underline{u}_1, \underline{u}_2$ they contain nondistinguished variables and variables from $\underline{e}$ and $\underline{s}$.

Forming the two composites yields two equivalent rules:

$$r_1r_2\colon\ \boldsymbol{P}_O(\underline{p}_1, \underline{p}_2, \underline{s}, \underline{c}, \underline{e}) :- \boldsymbol{P}_I\big(\underline{z}_2, \underline{z}_1, \underline{s}, h_1(h_2(\underline{c})), g_{12}(\underline{v}_2)\big) \wedge \boldsymbol{S}(\underline{u}_1) \wedge \boldsymbol{S}\big(g_{12}(\underline{u}_2)\big) \wedge \boldsymbol{Q}_1(\underline{w}_1) \wedge \boldsymbol{Q}_2(\underline{w}_2),$$

$$r_2r_1\colon\ \boldsymbol{P}_O(\underline{p}_1, \underline{p}_2, \underline{s}, \underline{c}, \underline{e}) :- \boldsymbol{P}_I\big(\underline{z}_2, \underline{z}_1, \underline{s}, h_2(h_1(\underline{c})), g_{21}(\underline{v}_1)\big) \wedge \boldsymbol{S}\big(g_{21}(\underline{u}_1)\big) \wedge \boldsymbol{S}(\underline{u}_2) \wedge \boldsymbol{Q}_1(\underline{w}_1) \wedge \boldsymbol{Q}_2(\underline{w}_2).$$

We only explain the formation of r_1r_2, since r_2r_1 is formed similarly. $S(\underline{u}_1)$ remains as is from r_1. The variables of $\underline{u}_2$ in S change according to $g_{12}(\underline{u}_2)$ to produce $S(g_{12}(\underline{u}_2))$. $Q_1(\underline{w}_1)$ remains as is from r_1. The nondistinguished variables of $\underline{w}_2$ remain the same. (Since the two rules have distinct nondistinguished variable names there is no need for renaming.) The distinguished variables in $\underline{w}_2$ are all members of $\underline{p}_1$ and $\underline{s}$. Since all of them are 1-persistent in r_1, they remain the same in the composition. Hence, all the variables in $\underline{w}_2$ remain the same, and this produces $Q_2(\underline{w}_2)$. The variables in P_I from r_2 are formed as follows. The variables in $\underline{z}_2$ are either nondistinguished or they are free 1-persistent in r_1, i.e., they belong to $\underline{p}_1$, so they remain the same. The variables in $\underline{p}_2$ are free 1-persistent in r_2, hence they are replaced by $h_1(\underline{p}_2)$, i.e., by the corresponding variables in the antecedent of r_1, which are the variables in $\underline{z}_1$. The variables in $\underline{s}$ are 1-persistent, so they remain the same. The variables in $h_2(\underline{c})$ are permuted according to h_1 to give $h_1(h_2(\underline{c}))$. Finally, the variables in $\underline{v}_2$ are replaced by $g_{12}(\underline{v}_2)$.

Examining the two composites we observe the following. First, the parts of them that come from augmented bridges that are equivalent in the two rules are isomorphic. This is because when $s_1 = s_2 = s$, then $s_1s_2 = s_2s_1 = s^2$. More precisely, there is an isomorphism between the variables of $\underline{u}_1$, $g_{12}(\underline{u}_2)$, and $g_{12}(\underline{v}_2)$ and those of $\underline{u}_2$, $g_{21}(\underline{u}_1)$, and $g_{21}(\underline{v}_1)$ respectively. A straightforward renaming of their nondistinguished variables will make the two parts equal. Moreover, none of these variables appears anywhere else in the rules, since they belong to distinct bridges in the α-graph. Thus, this renaming does not affect the remaining parts of the rules. Second, by part (c) of the statement of the theorem, the equality $h_1(h_2(\underline{c})) = h_2(h_1(\underline{c}))$ holds. Third, the remaining parts of the two composites are the same. Hence, the two composites are isomorphic, i.e., equivalent. Therefore, the two original rules commute. □

Example 5.2. The canonical set of commuting rules involves the two linear forms of transitive closure:

$$P_O(x,y) :- Q(x,u) \wedge P_I(u,y),$$

$$P_O(x,y) :- P_I(x,v) \wedge R(v,y).$$

Both composites are equal to the rule below:

$$P_O(x,y) :- P_I(u,v) \wedge Q(x,u) \wedge R(v,y).$$

The α-graphs of the two rules are shown in Figure 3. Every distinguished variable is free 1-persistent in one of the two original rules, i.e., it satisfies condition (a) of Theorem 5.1. As a side comment, note that the product of the two original rules is

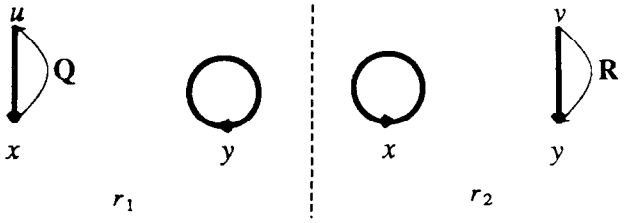

FIGURE 3. α-graphs of commuting rules satisfying the condition of Theorem 5.1.

the recursive rule of the "same-generation" program. Some implications of this fact have been examined elsewhere [13]. □

Example 5.3. The following is a more complex pair of rules that also commute with each other:

$$P_O(x,y,z) :- P_I(u,y,z) \wedge Q(x,y),$$

$$P_O(x,y,z) :- P_I(x,y,v) \wedge R(z,y).$$

Both composites are equal to the rule below:

$$P_O(x,y,z) :- P_I(u,y,v) \wedge Q(x,y) \wedge R(z,y).$$

The α-graphs of the two rules are shown in Figure 4. Note that the condition of Theorem 5.1 is satisfied by the corresponding α-graphs. □

Unfortunately, as the following counterexample shows, the condition of Theorem 5.1 is not necessary for commutativity.

Example 5.4. The following two rules also commute with each other:

$$P_O(x,y) :- P_I(y,w) \wedge Q(x)$$

$$P_O(x,y) :- P_I(u,v) \wedge Q(x) \wedge Q(y)$$

Both composites are isomorphic to the rule below:

$$P_O(x,y) :- P_I(u,v) \wedge Q(y) \wedge Q(w) \wedge Q(x).$$

The α-graphs of the two rules are shown in Figure 5. Note that, in this case, the condition of Theorem 5.1 is not satisfied. □

5.2. *A Necessary and Sufficient Condition*

We are not aware of any necessary and sufficient condition for commutativity of rules of unrestricted form that is computationally or aesthetically better than the condition of the definition of commutativity. In this section, we show that the condition of Theorem 5.1 is necessary and sufficient for commutativity if we restrict our attention to *range-restricted* rules, i.e., every variable in the consequent appears at least once in the antecedent as well, with *no repeated variables* in the consequent and *no repeated nonrecursive predicates* in the antecedant. The second restriction is enforced after all equalities have been eliminated from the antecedent. Before proceeding with the proof of the theorem, we need the following lemmas.

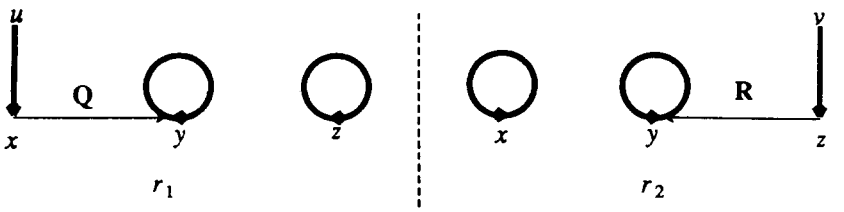

FIGURE 4. α-graphs of commuting rules satisfying the condition of Theorem 5.1.

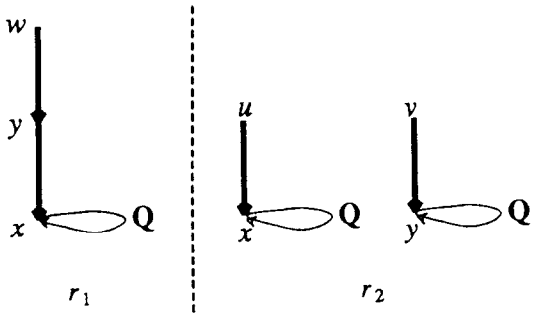

FIGURE 5. α-graphs of commuting rules not satisfying the condition of Theorem 5.1.

Lemma 5.1. Consider two rules, r_1 and r_2, with no repeated variables in the consequent that commute with each other. Let x be a distinguished variable, with $h_1(x) = x'$ and $h_2(x) = x''$, such that both x' and x'' are distinguished. Then, one of the following holds:

(a) *both $h_1(x'')$ and $h_2(x')$ are distinguished and $h_1(x'') = h_2(x')$, i.e., $h_1(h_2(x)) = h_2(h_1(x))$, or*
(b) *both $h_1(x'')$ and $h_2(x')$ are nondistinguished.*

PROOF. Assume that the two rules have the following form:

$$r_1\colon\ P_O(x,\cdots) :- P_I(x',\cdots) \wedge \cdots,$$

$$r_2\colon\ P_O(x,\cdots) :- P_I(x'',\cdots) \wedge \cdots.$$

The two composites are

$$r_1 r_2\colon\ P_O(x,\cdots) :- P_I(g_{12}(x''),\cdots) \wedge \cdots,$$

$$r_2 r_1\colon\ P_O(x,\cdots) :- P_I(g_{21}(x'),\cdots) \wedge \cdots.$$

Since x' and x'' are distinguished, by definition, $g_{12}(x'') = h_1(x'')$ and $g_{21}(x') = h_2(x')$. If $h_1(x'')$ is distinguished, due to the homomorphisms that have to exist between the two composites, it must be $h_1(x'') = h_2(x')$, which also implies that $h_2(x')$ is distinguished. On the other hand, if $h_1(x'')$ is nondistinguished, due to the homomorphisms between the two composites, $h_2(x')$ must be nondistinguished also. □

Lemma 5.2. Consider two rules, r_1 and r_2, with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent, that commute with each other. Let $\{x_k\}$, $0 \le k \le n+1$, be a set of distinguished variables such that $h_1(x_k) = x_{k+1}$, i.e., $h_1^{k+1}(x_0) = x_{k+1}$, for $0 \le k \le n$, and x_0 appears in a nonrecursive predicate $\boldsymbol{Q}$. Then, one of the following holds:

(a) *$h_2(x_k) = x_k$, $0 \le k \le n+1$, or*
(b) *$h_2(x_k) = x_{k+1}$, i.e., $h_2^{k+1}(x_0) = x_{k+1}$, for $0 \le k \le n$, and x_0 appears in a nonrecursive predicate $\mathbf{Q}$ in r_2.*

PROOF. Let $h_2(x_k) = y_k$, $0 \le k \le n+1$. The relevant parts of r_1 and r_2 are given below. We include a nonrecursive predicate $\boldsymbol{Q}$ in r_2, but we examine both cases,

when it exists and when it does not. The two different instances of Q will be distinguished by superscripts.

$$r_1\colon\ P_O(x_0,\ldots,x_k,\cdots)\text{:-}\ P_I(x_1,\ldots,x_{k+1},\cdots)\wedge Q^1(x_0,\cdots)\wedge\cdots,$$

$$r_2\colon\ P_O(x_0,\ldots,x_k,\cdots)\text{:-}\ P_I(y_0,\ldots,y_k,\cdots)\wedge Q^2(z,\cdots)\wedge\cdots.$$

Composing the two rules we have

$$r_1r_2\colon\ P_O(x_0,\ldots,x_k,\cdots)\text{:-}\ P_I(g_{12}(y_0),\ldots,g_{12}(y_k),\cdots)\wedge Q^1(x_0,\cdots)\wedge Q^2(g_{12}(z),\cdots)\wedge\cdots,$$

$$r_2r_1\colon\ P_O(x_0,\ldots,x_k,\cdots)\text{:-}\ P_I(y_1,\ldots,y_{k+1},\cdots)\wedge Q^1(y_0,\cdots)\wedge Q^2(z,\cdots)\wedge\cdots.$$

Examining the two composites we distinguish two cases. If Q does not appear in r_2 (i.e., if we ignore Q^2), in order for the two composites to be equivalent, it has to be $y_0=x_0$. An easy induction on k shows that $y_k=x_k$, i.e., $h_2(x_k)=x_k$, for all $0\le k\le n+1$.

Basis: For $k=0$, it was just shown that $y_0=x_0$.

Induction step: Assume that the claim is true for some $0\le k\le n$. We prove it for $k+1$. By the induction hypothesis, it is $y_k=x_k$. Hence, $g_{12}(y_k)=g_{12}(x_k)=h_1(x_k)=x_{k+1}$. (The second equality is due to x_k being a distinguished variable, whereas the last one is by the definition of $\{x_i\}$.) Due to the homomorphisms that must exist between the two composites in order for them to be equivalent, the instance of P_I in r_1r_2 must map to the instance of P_I in r_2r_1 and vice-versa. Comparing the two, we conclude that $y_{k+1}=x_{k+1}$ (because x_{k+1}, $k\le n$, is a distinguished variable).

If Q appears in r_2, then one of z or y_0 must be equal to x_0. If $y_0=x_0$, we have just shown that $h_2(x_k)=x_k$, for all $0\le k\le n+1$. If $z=x_0$, then since z is distinguished, by definition it must be $g_{12}(z)=g_{12}(x_0)=h_1(x_0)=x_1$. Since the two composites are equivalent, the necessary homomorphisms between them imply that $y_0=x_1$. Again, an easy induction on k shows that $y_k=x_{k+1}$, i.e., $h_2(x_k)=x_{k+1}$, for all $0\le k\le n$.

Basis: For $k=0$, it was just shown that $y_0=x_1$.

Induction step: Assume that the claim is true for some $0\le k\le n-1$. We prove it for $k+1$. By the induction hypothesis, it is $y_k=x_{k+1}$. Hence, $g_{12}(y_k)=g_{12}(x_{k+1})=h_1(x_{k+1})=x_{k+2}$. By definition, since $k+2\le n+1$, x_{k+2} is distinguished. Hence, comparing again the two instances of P_I in r_1r_2 and r_2r_1, we conclude that $y_{k+1}=x_{k+2}$.

In both cases, whether r_2 contains Q or not, we have shown that one of (a) or (b) holds. □

Theorem 5.2. Two range-restricted rules r_1 and r_2 with the same consequent and no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent commute if and only if every distinguished variable x satisfies one of the

following:

(a) *x is free 1-persistent in r_1 or r_2;*
(b) *x is link 1-persistent in both r_1 and r_2;*
(c) *x is free m_1-persistent, $m_1 > 1$, in r_1 and free m_2-persistent, $m_2 > 1$, in r_2 and $h_1(h_2(x)) = h_2(h_1(x))$;*
(d) *x is link m-persistent, $m > 1$, or general, and belongs to equivalent augmented bridges in both r_1 and r_2.*

PROOF. Recall that we assume that the two rules have the same consequents and share no nondistinguished variables. The "if" direction of the theorem follows from Theorem 5.1.

For the other direction of the theorem, assume that r_1 and r_2 commute. We show that for a distinguished variable x of r_1, one of (a), (b), (c), or (d) holds in r_2, depending on the type of x. Since the theorem is symmetric in r_1 and r_2, the variables in r_2 are not examined. We always consider x being the first distinguished variable in the consequent, and we only write down the parts of the rules that are relevant to the proof. Also, unimportant variables are denoted by _.

(i) *x is a free 1-persistent variable*: This simply states that (a) holds.
(ii) *x is a link 1-persistent variable*: In this case, x appears at least twice in the antecedent of r_1. Since the rules are range-restricted, this implies that there exists a set of distinguished variables $\{x_k\}$, $0 \le k \le n+1$, such that $h_1(x_k) = x_{k+1}$, $0 \le k \le n$, with $x = x_n = x_{n+1}$, and such that x_0 appears in a nonrecursive predicate $\boldsymbol{Q}$. If this is not true, then there must exist repeated variables in the consequent of r_1, which is a contradiction. Applying Lemma 5.2 for $x = x_n$ yields $h_2(x_n) = x_n$ or $h_2(x_n) = x_{n+1}$. Since $x = x_n = x_{n+1}$, this implies that in all cases $h_2(x) = x$, i.e., x is 1-persistent in r_2 ((a) or (b) holds).
(iii) *x is a free m_1-persistent variable, $m_1 > 1$*: Since the rules are range-restricted, if x is not a free m_2-persistent variable, $m_2 \ge 1$, in r_2, there must exist a set of distinguished variables $\{y_k\}$, $0 \le k \le n+1$, such that $h_2(y_k) = y_{k+1}$, $0 \le k \le n$, with $x = y_{n+1}$, and such that y_0 appears in a nonrecursive predicate $\boldsymbol{Q}$ in r_2. By Lemma 5.2, this implies that either $x = h_1(x)$ or $x = h_1^{n+1}(y_0)$ and y_0 appears in a nonrecursive predicate $\boldsymbol{Q}$ in r_1. In the first case, x is a 1-persistent variable in r_1, and in the second case, x is a link l-persistent, $l > 1$, or general variable in r_1. In both cases, this is a contradiction, since x is a free m_1-persistent variable, $m_1 > 1$, in r_1. Hence, x must be a free m_2-persistent, $m_2 \ge 1$, variable in r_2 also.

If $m_2 = 1$, i.e., it is free 1-persistent ((a) holds), then x satisfies the theorem. Otherwise, x is free m_2-persistent, $m_2 > 1$, in r_2, and we have to show that $h_1(h_2(x)) = h_2(h_1(x))$. Since x is a free persistent variable in r_1 and r_2, by definition, $h_1(x)$ and $h_2(x)$ must also be free persistent variables in r_1 and r_2 respectively ($h_i(x)$ is part of the same component as x in r_i). The argument in the previous paragraph can be applied in the case of $h_2(x)$ and yield that $h_2(x)$ is a free persistent variable in r_1 as well. Hence, $h_1(x)$, $h_2(x)$, and $h_1(h_2(x))$ are distinguished variables. By Lemma 5.1, $h_2(h_1(x))$ is also distinguished, and $h_2(h_1(x)) = h_1(h_2(x))$, i.e., (c) holds.
(iv) *x is a link m-persistent, $m > 1$,or general variable*: Again, since the rules are range-restricted, this implies that there exists a set of distinguished vari-

ables $\{x_k\}$, $0 \le k \le n+1$, such that $h_1(x_k) = x_{k+1}$, $0 \le k \le n$, with $x = x_{n+1}$, and such that x_0 appears in a nonrecursive predicate Q in r_1. By Lemma 5.2, this implies that either $x = h_2(x)$, i.e., that x is 1-persistent in r_2, or $x = h_2^{n+1}(x_0)$, and x_0 appears in a nonrecursive predicate Q in r_2, i.e., that x is link persistent or general in r_2. We examine the two cases separately.

If x is 1-persistent in r_2, we show that it cannot be link 1-persistent, i.e., it must be free 1-persistent. Assume to the contrary that x is link 1-persistent in r_2. From case (ii) for r_2, we conclude that x is 1-persistent in r_1, which is a contradiction. Hence, x must be free 1-persistent in r_2 ((a) holds).

If x is link persistent or general in r_2, we show that it belongs to an augmented bridge that is equivalent to the augmented bridge to which it belongs in r_1. Recall that we examine the case where $h_2(x_k) = x_{k+1}$, for all $0 \le k \le n$, which implies that $h_1(x_k) = h_2(x_k)$. Since $x = x_{n+1}$ is an arbitrary link m-persistent, $m > 1$,or general variable in its augmented bridge in r_1, we can conclude that for any such variable z in that augmented bridge, either both $h_1(z), h_2(z)$ are distinguished and $h_1(z) = h_2(z)$, or both $h_1(z), h_2(z)$ are nondistinguished, i.e., the structure of h for the augmented bridges of z in r_1 and r_2 is the same. Hence, if we assume that the two augmented bridges are not equivalent, there must be some nonrecursive predicate connected (through a series of nonrecursive predicates) to a link m-persistent, $m > 1$, or general variable in the bridge in r_1 that is not connected through the same series of nonrecursive predicates to the same distinguished variable in the bridge in r_2 (or vice-versa). Without loss of generality, assume that x is such a distinguished variable. Also without loss of generality, assume that $h_1(x) = h_2(x) = y$ is a distinguished variable, and that only nondistinguished variables appear in the nonrecursive predicates connected to x (except x). The other cases are treated similarly. This situation is depicted in the following two rules:

$$\begin{aligned} r_1\colon\ & \boldsymbol{P}_O(x,\cdots) :- \boldsymbol{P}_I(y,\cdots) \wedge \boldsymbol{R}_1(x,z_1) \wedge \boldsymbol{R}_2(z_1,z_2)\cdots \\ & \qquad \wedge \boldsymbol{R}_{m-1}(z_{m-2},z_{m-1}) \wedge \boldsymbol{R}_m(z_{m-1},z_m) \wedge \cdots, \\ r_2\colon\ & \boldsymbol{P}_O(x,\cdots) :- \boldsymbol{P}_I(y,\cdots) \wedge \boldsymbol{R}_1(x,z_1') \wedge \boldsymbol{R}_2(z_1',z_2')\cdots \\ & \qquad \wedge \boldsymbol{R}_{m-1}(z_{m-2}',z_{m-1}') \wedge \cdots. \end{aligned}$$

Composing the two rules we obtain the following:

$$\begin{aligned} r_1r_2\colon\ & \boldsymbol{P}_O(x,\cdots) :- \boldsymbol{P}_I(\_,\cdots) \wedge \boldsymbol{R}_1(y,z_1') \wedge \boldsymbol{R}_2(z_1',z_2')\cdots \\ & \qquad \wedge \boldsymbol{R}_{m-1}(z_{m-2}',z_{m-1}') \\ & \qquad \wedge \boldsymbol{R}_1(x,z_1) \wedge \boldsymbol{R}_2(z_1,z_2)\cdots \wedge \boldsymbol{R}_{m-1}(z_{m-2},z_{m-1}) \\ & \qquad \wedge \boldsymbol{R}_m(z_{m-1},z_m) \wedge \cdots, \\ r_2r_1\colon\ & \boldsymbol{P}_O(x,\cdots) :- \boldsymbol{P}_I(\_,\cdots) \wedge \boldsymbol{R}_1(y,z_1) \wedge \boldsymbol{R}_2(z_1,z_2)\cdots \\ & \qquad \wedge \boldsymbol{R}_{m-1}(z_{m-2},z_{m-1}) \wedge \boldsymbol{R}_m(z_{m-1},z_m) \\ & \qquad \wedge \boldsymbol{R}_1(x,z_1') \wedge \boldsymbol{R}_2(z_1',z_2')\cdots \wedge \boldsymbol{R}_{m-1}(z_{m-2}',z_{m-1}') \wedge \cdots. \end{aligned}$$

Clearly, since $y \neq x$ (x is not 1-persistent), the two composites are not equivalent, and r_1 and r_2 cannot commute, which is a contradiction. Hence,

the assumption that the two augmented bridges to which x belongs in r_1 and r_2 are not equivalent is wrong, i.e., (d) holds. □

5.3. *Complexity*

In order to show the complexity of testing the condition in Theorem 5.2, we first need to discuss the complexity of finding the bridges in a graph with respect to a subgraph and that of testing equivalence of two range-restricted rules with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent. We discuss the two problems separately.

Identifying the bridges of an undirected graph with respect to a subgraph is very similar to identifying biconnected components in the graph [2]. The two problems have the same complexity. In particular, the complexity of identifying bridges is given by the following lemma, which is provided without a proof.

Lemma 5.3. Identifying the bridges of an undirected graph with respect to a subgraph can be done in $O(n + e)$ time, where e is the number of edges and n is the number of nodes in the graph.

The complexity of testing the equivalence of two rules with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent is addressed in Lemma 5.4.

Lemma 5.4. Equivalence testing of two range-restricted rules with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedant can be done in $O(a \log a)$, where a is the total number of argument positions in the predicates in the antecedents of the two rules.

PROOF. Since the rules contain no repeated nonrecursive predicates, if they are equivalent, they have to be isomorphic. Moreover, every predicate in the one rule can map to only one predicate in the other. Thus, equivalence can be tested as follows:

(1) Test if the set of predicates in the antecedents of the rules are the same. This can be done by first sorting the two sets, and then examining the predicates pairwise, traversing the two sets in order; since the number of predicates is less than or equal to the number of argument positions in the predicates, this step takes $O(a \log a)$ time.

(2) Define f such that for any pair of literals $Q(x_1, \ldots, x_n)$ in the antecedent of r_1 and $Q(y_1, \ldots, y_n)$ in the antecedent of r_2, $f(x_i) = y_i$ and test if f is $1-1$ (and onto) and $x_i = y_i$ when x_i is distinguished. This can be done by first sorting the list of variables of the one rule and then scanning the antecedents of both rules in parallel and recording the value of $f(x)$ for every variable x in the sorted list. If at any point, two distinct values are assigned to $f(x)$ or $f(x) \neq x$ for a distinguished variable x, then the rules are not equivalent; otherwise, they are because f is an isomorphism between them. The cost of this step is dominated by the sorting cost of the variable list, i.e., it is $O(a \log a)$ time.

Adding the time complexities of Steps (1) and (2) yields that the total time

complexity of testing equivalence of rules that satisfy the restrictions stated in the lemma is $O(a \log a)$. □

Theorem 5.3. Commutativity of two range-restricted rules with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent can be tested in $O(a \log a)$ time, where a is the total number of argument positions in the recursive and the nonrecursive predicates of the antecedents of the rules.

PROOF. The algorithm has the following steps:

(1) Form the α-graphs of the two rules. The most complex operation in this step is sorting the lists of variables of the two rules, so that a unique node is assigned to each one of them in the appropriate graph, independent of the number of times it appears in the corresponding rule. Thus, this step can be done in $O(a \log a)$ time.
(2) Identify the type of every distinguished variable (i.e., free 1-persistent, link 1-persistent, free m-persistent, $m > 1$, link m-persistent, $m > 1$, or general), and then identify the bridges of the underlying undirected graphs of the α-graphs of the two rules. The number of argument positions a is an upper bound on both the nodes and the arcs in the graph. Hence, by Lemma 5.3, this step can be done in $O(a)$ time.
(3) For every link 1-persistent variable in the one rule, check if it is 1-persistent in the other. This step takes $O(1)$ for every link 1-persistent variable, for a total time of $O(a)$.
(4) For every free m_1-persistent variable, $m_1 > 1$, in the one rule, check if it is free m_2-persistent, $m_2 \geq 1$, in the other. In addition, for every such variable x, test whether $h_1(h_2(x)) = h_2(h_1(x))$ or not. This step takes $O(1)$ for every free m_1-persistent variable, $m_1 > 1$, for a total of $O(a)$ time.
(5) For every link m-persistent, $m > 1$, or general variable in the one rule, check if it is free 1-persistent in the other. If it is, do nothing. This step takes $O(1)$ for every such variable, for a total of $O(a)$ time. If it is not, check if it belongs in an equivalent augmented bridge in the other rule. Because the rules contain no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent, by Lemma 5.4, equivalence of all the relevant bridges can be tested in $O(a \log a)$ time. (In fact, its cost will be $O(a)$ if the sorted variable lists from Step (1) are being used.)

The total complexity is given by the sum of the total times for steps (1), (2), (3), (4), and (5), which is equal to $O(a \log a)$. □

6. SEPARABILITY AND RECURSIVE REDUNDANCY REVISITED

In Section 4, we examined commutativity vs. separability and recursive redundancy as expressed in the abstract form of the algebra to obtain results that hold for any linear rules. In this section, we restrict ourselves to function-free, constant-free, and range-restricted rules and use our results from Section 5 to obtain more relationships of commutativity with separability and recursive redundancy for this class of rules.

6.1. Commutativity vs. Separability

Based on the original definition [15],[4] two rules r_1 and r_2 with the same consequent are separable if

(1) For any distinguished variable x, either $h_i(x) = x$ or $h_i(x)$ is nondistinguished, $i = 1, 2$;
(2) For any distinguished variable x, either both x and $h_i(x)$ appear under nonrecursive predicates in r_i or none, $i = 1, 2$.
(3) The sets of distinguished variables that appear under nonrecursive predicates in r_1 and r_2 are either equal or disjoint.
(4) The subgraph of the α-graph of r_i induced by its static arcs is connected, $i = 1, 2$.

For the case of two rules, one can take advantage of the efficient features of the separable algorithm only if in Clause (3) the intersection of the sets of distinguished variables that appear under nonrecursive predicates in r_1 and r_2 is empty. With this assumption, Naughton proved the following theorem on the relationship of separable recursions and the separable algorithm (Algorithm 4.1).

Theorem 6.1 [15]. Given two operators A_1 and A_2 that are separable and a full selection σ,[5] then $\sigma(A_1 + A_2)^* = A_1^*(\sigma A_2^*)$, i.e., the separable algorithm can be used for the computation of $\sigma(A_1 + A_2)^*$.

With the same assumption on Condition (3) of the definition of separable rules as above, we can prove the following lemma.

Lemma 6.1. If two range-restricted rules r_1 and r_2 with the same consequent are separable, then they only contain 1-persistent and general variables. Moreover, any link 1-persistent or general variable in r_1 is free 1-persistent in r_2 (similarly for the variables of r_2).

PROOF. Condition (1) of the definition of separable rules states that for any variable x, either $h_i(x) = x$ or $h_i(x)$ is nondistinguished, $i = 1, 2$. In the first case, x is 1-persistent in r_i, whereas in the second one, x is general. If x is link 1-persistent or general in one of the rules, say r_1, x must appear under some nonrecursive predicate in r_1. Otherwise, there must exist another distinguished variable y, such that $h_1(y) = x$, which contradicts Condition (1) of the definition of separble rules. Hence, by Condition (3), x is free 1-persistent in r_2. □

Combining Lemma 6.1 with Theorem 5.1 yields the following theorem.

Theorem 6.2. If two rules are separable then they commute, but the opposite does not hold.

PROOF. If two rules r_1 and r_2 are separable, by Lemma 6.1, every variable is free 1-persistent in r_1 or r_2, i.e., it satisfies Condition (a) of Theorem 5.1. Thus, by Theorem 5.1, the two rules commute.

[4] The definition given in this paper can be easily extended to multiple rules (in accordance to the original definition [15]). For presentation clarity, however, we restrict ourselves to two rules.

[5] The precise definition of full selections is given by Naughton [15]. For our purposes, the key observation is that if $A_1A_2 = A_2A_1$ and $\sigma A_1 = A_1\sigma$, then σ is a full selection.

The rules of Example 5.3 serve as examples of commutative rules that are not separable. They violate both Condition (2) and Condition (3) of the separable definition. □

By Theorem 6.2, commutativity is a strictly more general notion than separability. Nevertheless, by Theorem 4.1, all the efficient processing algorithms for separable rules are applicable for commutative rules as well, i.e., Theorem 4.1 is a strict generalization of Theorem 6.1.

6.2. *Commutativity vs. Recursive Redundancy*

The following lemma on the relationship of the properties of being uniformly bounded and being torsion is instrumental in proving the necessity of the condition of Theorem 4.2 for recursive redundancy.

Lemma 6.2. [6]*Every uniformly bounded rule with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent is torsion.*

PROOF. Consider a rule r that satisfies the conditions of the lemma. By definition, this implies that there are $k > 0$ and $\tau > 0$ such that $r^{k+\tau} \leq r^k$, i.e., there is a homomorphism f from r^k to $r^{k+\tau}$. Moreover, as Naughton has pointed out [17], for the class of rules defined in the lemma, we can find $k > 0$ and $\tau > 0$ such that the antecedents of r^k and $r^{k+\tau}$ are of the form

r^k: ab,

$r^{k+\tau}$: aqb',

where a, b, b', and q are conjunctions of predicates that satisfy the following: (i) any two of them that appear in the same rule share no nondistinguished variables, (ii) the recursive predicate appears in b or b', and (iii) b and b' are isomorphic. Naughton showed that there is a homomorphism f: $r^k \to r^{k+\tau}$ such that $f(a) = a$ and $f(b) = b'$ [17]. Consider $r^{k+2\tau}$. Clearly, it can be written in the form

$r^{k+2\tau}$: $aqq'b''$,

where q is isomorphic to q' and b' is isomorphic to b''. Property (i) and the isomorphism of q and q' guarantee that no nondistinguished variable is shared between any two of a, q, q' and b''. Based on this and the existence of f, we can define two homomorphisms f_1: $r^{k+\tau} \to r^{k+2\tau}$ and f_2: $r^{k+2\tau} \to r^{k+\tau}$ as follows:

$$f_1(a) = a, \quad f_1(q) = q, \quad f_1(b') = b''$$

$$f_2(a) = a, \quad f_2(q) = q, \quad f_2(q') = q, \quad f_2(b'') = b'.$$

The existence of f_1 and f_2 imply that $r^{k+\tau} = r^{k+2\tau}$, i.e., that r is torsion. □

For the study of recursive redundancy, the following subset of the general variables in the α-graph of a rule plays an important role. Any general distinguished variable whose corresponding node in the α-graph of a rule is connected to some link-persistent variable through a path of dynamic arcs alone is a *ray* variable. If n is the length of the shortest such path, then the variable is called *n-ray*. Let I_l and I_r be the sets of the link-persistent and ray variables in some rule

[6] Similar results are easily provable for the class of recursions examined by Ioannidis [10].

r (corresponding to some operator A) respectively, and let I be the union of the two sets $I = I_l \cup I_r$. Let G_I denote the subgraph of the α-graph of r (A) induced by the dynamic arcs connecting the variables in I, and G_I^L denote the subgraph of the α-graph of r^L (A^L) induced by the dynamic arcs connecting the variables in I as well, for any $L \geq 1$. In the study of recursive redundancy, an important role is played by the bridges of the α-graph of r with respect to G_I. A necessary and sufficient condition for a nonrecursive predicate in a rule of some restricted form to be redundant was originally given by Naughton [16]. Using a different terminology, that condition is expressed in the following theorem.

Theorem 6.3. [16]. A nonrecursive predicate in a rule with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent is recursively redundant if and only if it appears in a uniformly bounded augmented bridge of the α-graph of the rule with respect to G_I.

Example 6.1. [16]. Consider the following rule, whose corresponding α-graph is shown in Figure 6:

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

Clearly, the component of the graph that contains the variable y is a uniformly bounded augmented bridge with respect to the subgraph induced by the dynamic arcs connecting its ray and link-persistent variables (of which there is only one: y). Thus, according to Theorem 6.3, the predicate **cheap** is recursively redundant. □

We present a different necessary and sufficient condition that shows the relationship between commutativity and recursive redundancy. For that, we need the following lemmas.

Lemma 6.3. Let A be an operator corresponding to a rule with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent. The following holds.

(a) *For all $L \geq 1$, I_l and I_r are the sets of link-persistent and ray variables of A^L respectively.*
(b) *There exists $L \geq 1$ such that all variables in I_l are link 1-persistent and all variables in I_r are 1-ray in A^L.*

PROOF. Part (a) is obvious. For Part (b), any link L_x-persistent variable x in A is link 1-persistent in A^{mL_x} for all $m \geq 1$. In addition, any L_y-ray variable y in A is 1-ray in A^m for all $m \geq L_y$. Let $cm(S)$ denote the set of common multiples of the members of a set S. Choose $L = \min\{M | M \in cm_{x \in I_l}\{L_x\}$ and $M \geq$

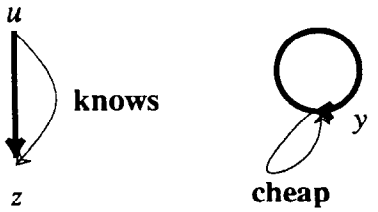

FIGURE 6. α-graph of rule with a recursively redundant predicate.

$\max_{y \in I_x}\{L_y\}\}$. That is, L is the least common multiple of $\{L_x\}$ that is greater than the maximum of $\{L_y\}$. Clearly, all link-persistent variables in A are link 1-persistent in A^L, all ray variables in A are 1-ray in A^L, and no other variable satisfies that (because of Part (a)). □

Consider a rule r with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent, and let $\boldsymbol{Q}$ be one of those predicates. Clearly, the Lth power of r contains L instances of $\boldsymbol{Q}$ in the antecedent. Every such instance is said to be *generated* by the instance of $\boldsymbol{Q}$ in the antecedent of r. Accordingly, the static arcs in the α-graph of r^L that correspond to those predicates are said to be *generated* by the arc in the α-graph of r that corresponds to the original instance of $\boldsymbol{Q}$.

Lemma 6.4. Let A be an operator corresponding to a rule with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent. The set of arcs generated by those of any bridge in the α-graph of A with respect to G_I forms one or more bridges in the α-graph of A^L with respect to G_I^L, for any $L \geq 1$.

PROOF. The lemma holds trivially for $L = 1$. Consider two arcs $(z_1 \rightarrow z_2)$ and $(w_1 \rightarrow w_2)$ in the α-graph of A that belong to different bridges with respect to G_I. Let $(z_1' \rightarrow z_2')$ and $(w_1' \rightarrow 2_2')$ be the arcs generated by $(z_1 \rightarrow z_2)$ and $(w_1 \rightarrow w_2)$ respectively in $A^L, L > 1$. If $(z_1' \rightarrow z_2')$ and $(w_1' \rightarrow w_2')$ are not connected in the α-graph of A^L, they trivially belong to different bridges with respect to G_I^L. If they are connected, the walk that connects them must correspond to a walk that connects $(z_1 \rightarrow z_2)$ and $(w_1 \rightarrow w_2)$ in the α-graph of A, which by definition passes through at least one link-persistent or ray variable x of A, since the two arcs belong to different bridges. Thus, the walk connecting $(z_1' \rightarrow z_2')$ and $(w_1' \rightarrow w_2')$ must pass through at least one of the variables that replace x in A^l, for some $l \leq L$. Since x is link persistent or ray, however, it is only replaced by link-persistent or ray variables as well. Thus, the walk connecting $(z_1' \rightarrow z_2')$ and $(w_1' \rightarrow w_2')$ must pass through one of those variables. By Lemma 6.3, this implies that the two arcs belong to different bridges with respect to G_I^L. □

Lemma 6.5. Consider the α-graph of some operator A. If C is the wide operator that corresponds to a set of augmented bridges of the graph with respect to G_I, then there exists an operator B such that $A = BC$.

PROOF. Let B be the operator that corresponds to the α-graph that is constructed as follows: in the α-graph of A, remove all arcs from the augmented bridges that correspond to C, introduce dynamic arcs so that their distinguished variables become (free or link) 1-persistent, and keep the rest of the graph unchanged. Because of the way B and C are defined, one can apply an argument similar to that in the proof of Theorem 5.1 and show that $A = BC$. Roughly, all nonrecursive predicates in C remain unchanged in BC, because all variables that they contain are unaffected by the composition: the distinguished ones are 1-persistent in B and the nondistinguished ones do not appear in B (nondistinguished variables in B and C come from different bridges in the α-graph of A), so by the definition of the g_{BC} function (Section 5), they remain unchanged. Thus, the identity function serves as an isomorphism between the nonrecursive predicates of A and BC. Moreover, every distinguished variable of A has been left unaffected in exactly one of B and C, whereas it has been transformed to a 1-persistent variable in the other. Thus,

the identity function serves as an isomorphism between the recursive predicates in the antecedents of A and BC as well. The above two facts yield the conclusion that $A = BC$. □

We can now proceed to the main result of this subsection. (Without loss of generality, we assume again that all operators have the same domain and range, so that the product of any pair of them is well defined.)

Theorem 6.4. Let $\boldsymbol{Q}$ be a parameter relation of some operator A that corresponds to a range-restricted rule with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent. $\boldsymbol{Q}$ is recursively redundant in A^* if and only if there exist $L \geq 1$ and operators B and C such that $\boldsymbol{Q}$ is a parameter of C but not B, C is uniformly bounded,

$$A^L = BC^L, \quad \text{and} \quad C^L(BC^L) = C^L(C^LB).$$

PROOF. By Lemma 6.2, if C is uniformly bounded and the corresponding rule contains no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent, C is also torsion. Thus, the *if* part of this theorem is given by Theorem 4.2.

For the *only if* part, assume that $\boldsymbol{Q}$ is recursively redundant. By Theorem 6.3, $\boldsymbol{Q}$ appears in a uniformly bounded augmented bridge in the α-graph of the rule with respect to G_I. Let C be the wide operator that corresponds to this augmented bridge. Clearly, C is uniformly bounded. By Lemma 6.4, the nonrecursive predicates of the augmented bridge that corresponds to C in the α-graph of A generate others in A^L, $L \geq 1$, that form a set of bridges in the α-graph of A^L with respect to G_I^L. It is straightforward to show that C^L is the wide rule related to the corresponding augmented bridges. Thus, by Lemma 6.5, for all $L \geq 1$, there exists an operator B such that $A^L = BC^L$. In addition, since all instances of $\boldsymbol{Q}$ are part of C^L, $\boldsymbol{Q}$ is not a parameter of B.

We choose $L \geq 1$ as defined by Lemma 6.3, Part (b), i.e., such that all link-persistent variables in A are link 1-persistent in A^L and all ray variables in A are 1-ray in A^L. Based on the construction in the proof of Lemma 6.5, we can partition the distinguished variables of A^L (as well as those of B and C^L) as follows:

- $\underline{x}$ vector of link 1-persistent and 1-ray variables that appear in the augmented bridges that correspond to C^L—they remain intact in C^L whereas they are 1-persistent in B;
- $\underline{x}_C$ vector of the remaining distinguished variables that appear in the augmented bridges that correspond to C^L—they remain intact in C^L whereas they are free 1-persistent in B;
- $\underline{x}_B$ vector of all other distinguished variables—they remain intact in B whereas they are free 1-persistent in C^L.

Without loss of generality, the variables in the consequents of the two rules are grouped so that the latter can be written in the following form:

$$B\colon\ \boldsymbol{P}_O(\underline{x}, \underline{x}_B, \underline{x}_C) :- \boldsymbol{P}_I(\underline{x}, \underline{z}_B, \underline{x}_C) \wedge \boldsymbol{Q}_B(\underline{x}', \underline{w}_B),$$

$$C^L\colon\ \boldsymbol{P}_O(\underline{x}, \underline{x}_B, \underline{x}_C) :- \boldsymbol{P}_I(h_{C^L}(\underline{x}), \underline{x}_B, \underline{z}_C) \wedge \boldsymbol{Q}_C(\underline{x}'', \underline{w}_C).$$

Predicates $\boldsymbol{Q}_B$ and $\boldsymbol{Q}_C$ represent the nonrecursive predicates in B and C^L

respectively. Also, $\underline{z}_B$, $\underline{z}_C$, $\underline{w}_B$, $\underline{w}_C$, $\underline{x}'$, $\underline{x}''$ are vectors of variables such that

$\underline{z}_B, \underline{w}_B$ they contain nondistinguished variables and variables from $\underline{x}_B$;

$\underline{z}_C, \underline{w}_C$ they contain nondistinguished variables and variables from $\underline{x}_C$;

$\underline{x}', \underline{x}''$ they contain variables from $\underline{x}$.

Forming the two composites, i.e., multiplying B and C^L in both possible ways, yields the following two rules:

$$BC^L\colon\ \boldsymbol{P}_O(\underline{x}, \underline{x}_B, \underline{x}_C) :- \boldsymbol{P}_I(h_{C^L}(\underline{x}), \underline{z}_B, \underline{z}_C) \wedge \boldsymbol{Q}_B(\underline{x}', \underline{w}_B) \wedge \boldsymbol{Q}_C(\underline{x}'', \underline{w}_C).$$

$$C^LB\colon\ \boldsymbol{P}_O(\underline{x}, \underline{x}_B, \underline{x}_C) :- \boldsymbol{P}_I(h_{C^L}(\underline{x}), \underline{z}_B, \underline{z}_C) \wedge \boldsymbol{Q}_B(h_{C^L}(\underline{x}'), \underline{w}_B) \wedge \boldsymbol{Q}_C(\underline{x}'', \underline{w}_C).$$

The formation of the above rules is straightforward and we do not explain it. Note that the two rules are isomorphic except for the first vector of arguments of $\boldsymbol{Q}_B$, which is equal to $\underline{x}'$ in BC^L, whereas it is equal to $h_{C^L}(\underline{x}')$ in C^LB. Thus, in order to prove that $C^L(BC^L) = C^L(C^LB)$, we only need to investigate how these variables are affected by the multiplication with C^L; all others behave equivalently in the two products. Recall that all variables in x' are link 1-persistent or 1-ray in C^L. For a variable x in x' that is link 1-persistent, $h_{C^L}(x) = x$. For a variable x in $\underline{x}'$ that is 1-ray, $h_{C^L}(x) = y$, where y is a link 1-persistent in C^L. Hence, independent of the type of x, $h_{C^L}(h_{C^L}(x)) = h_{C^L}(x)$. This implies that the first arguments of $\boldsymbol{Q}_B$ in $C^L(BC^L)$ and $C^L(C^LB)$ will be the same, like the remaining parts of the rules, i.e., it implies that the two products are equal. Thus, for rules in the class described in the statement of the theorem, the condition of Theorem 4.2 is necessary and sufficient for recursive redundancy. □

Example 6.2. Let A be the operator corresponding to the following rule, whose α-graph is shown in Figure 7:

$$\boldsymbol{P}(w, x, y, z) :- \boldsymbol{P}(x, w, x, u) \wedge \boldsymbol{Q}(x, u) \wedge \boldsymbol{R}(x, y) \wedge \boldsymbol{S}(u, z).$$

The role of C is played by the following rule:

$$C\colon\ \boldsymbol{P}(w, x, y, z) :- \boldsymbol{P}(x, w, x, z) \wedge \boldsymbol{R}(x, y).$$

Clearly, C is uniformly bounded (it has no nondistinguished variables). The nonrecursive predicate $\boldsymbol{R}$ is recursively redundant according to Theorem 6.2 and its augmented bridge with respect to G_I is enclosed in dotted boundaries in Figure 7. Theorem 6.4 is satisfied for this example for $L = 2$. The rules corresponding to

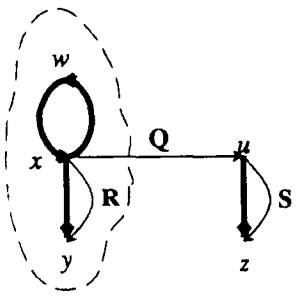

FIGURE 7. α-graph of rule with recursively redundant predicates.

operators A^2, B, and C^2 are

$$A^2\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,w,u') \wedge \boldsymbol{Q}(w,u') \wedge \boldsymbol{R}(w,x) \wedge \boldsymbol{S}(u',u) \wedge \boldsymbol{Q}(x,u) \wedge \boldsymbol{R}(x,y) \wedge \boldsymbol{S}(u,z),$$

$$B\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,y,u') \wedge \boldsymbol{Q}(w,u') \wedge \boldsymbol{S}(u',u) \wedge \boldsymbol{Q}(x,u) \wedge \boldsymbol{S}(u,z),$$

$$C^2\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,w,z) \wedge \boldsymbol{R}(w,x) \wedge \boldsymbol{R}(x,y).$$

One can verify that $A^2 = BC^2$. The α-graphs of B and C^2 are shown in Figure 8. Variables w and x are link 1-persistent in both B and C^2, whereas y is free 1-persistent in B and z is free 1-persistent in C^2. By Theorem 5.1, C^2 and B commute, and therefore, trivially $C^2(BC^2) = C^2(C^2B)$, i.e., Theorem 6.4 is satisfied. □

Example 6.3. Let A be slightly different from the previous example, i.e., having $\boldsymbol{Q}(y,u)$ instead of $\boldsymbol{Q}(x,u)$ in the antecedent:

$$\boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(x,w,x,u) \wedge \boldsymbol{Q}(y,u) \wedge \boldsymbol{R}(x,y) \wedge \boldsymbol{S}(u,z).$$

The α-graph of A is shown in Figure 9. Everything proceeds as in Example 6.2, except for the way $\boldsymbol{Q}$ behaves. The rules corresponding to operators A^2, B, and C^2 are

$$A^2\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,w,u') \wedge \boldsymbol{Q}(x,u') \wedge \boldsymbol{R}(w,x) \wedge \boldsymbol{S}(u',u) \wedge \boldsymbol{Q}(y,u) \wedge \boldsymbol{R}(x,y) \wedge \boldsymbol{S}(u,z),$$

$$B\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,y,u') \wedge \boldsymbol{Q}(x,u') \wedge \boldsymbol{S}(u',u) \wedge \boldsymbol{Q}(y,u) \wedge \boldsymbol{S}(u,z),$$

$$C^2\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,w,z) \wedge \boldsymbol{R}(w,x) \wedge \boldsymbol{R}(x,y).$$

One can verify that $A^2 = BC^2$ and that $BC^2 \neq C^2B$. Note that the latter cannot be derived by Theorem 5.2, since repeated nonrecursive predicates appear in the antecedents of the two rules. Instead it is derived by forming the two products. BC^2 is equal to A^2, which was given above. C^2B is shown below:

$$C^2B\colon\ \boldsymbol{P}(w,x,y,z) :- \boldsymbol{P}(w,x,w,u') \wedge \boldsymbol{Q}(x,u') \wedge \boldsymbol{R}(w,x) \wedge \boldsymbol{S}(u',u) \wedge \boldsymbol{Q}(w,u) \wedge \boldsymbol{R}(x,y) \wedge \boldsymbol{S}(u,z).$$

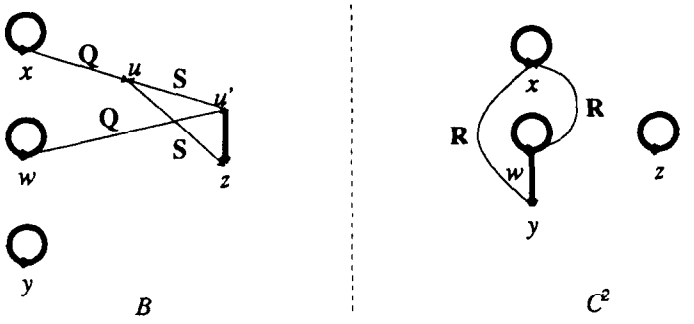

FIGURE 8. α-graphs of commuting rules B and C^2, where $A^2 = BC^2$ and C is recursively redundant.

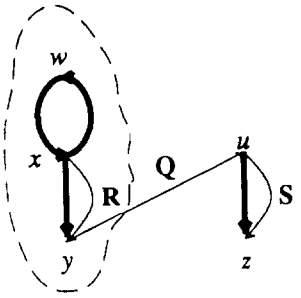

FIGURE 9. α-graph of rule with recursively redundant predicates.

Clearly, the two products are not equivalent, due to the presence of $Q(y,u)$ and $Q(w,u)$ in BC^2 and C^2B respectively. Nevertheless, multiplying either one with C^2 from the left yields the same rule, which after removing all repeated occurrences of literals is equal to

$$C^2(BC^2) = C^2(C^2B):$$

$$P(w,x,y,z) :- P(w,x,w,u') \wedge R(w,x) \wedge R(x,y) \wedge Q(x,u') \wedge \wedge S(u',u) \wedge Q(w,u) \wedge S(u,z).$$

Thus, Theorem 6.4 is satisfied. □

7. CONCLUSIONS

We have investigated the role of commutativity in query processing of linear recursive rules. Using the algebraic structure of such rules, we have identified commutativity as the essence of many properties that give rise to important classes of recursive rules, i.e., separable rules and rules with recursively redundant predicates. For separable rules, in particular, we have shown that commutativity is a strictly more general notion than separability, while it still allows the efficient separable algorithm to be applicable. Focusing on rules that contain no functions and no constants, we have given a sufficient condition for such rules to commute. We have also shown that the condition is necessary and sufficient when the rules are range-restricted and contain no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent. In that case, the condition can be tested in polynomial time in the size of the rules.

Commutativity emerges as a key property of linear recursive rules for which efficient algorithms can be applied. This paper is a first step in the investigation of its power. We believe that there is much more work to be done in this direction. Some problems we plan to study in the future are the following: characterize commutativity in more general classes of rules than the one studied in this paper; investigate the relationship of commutativity and one-sided recursion; investigate the relationship of commutativity and several optimizations proposed for the magic sets and counting algorithms (e.g., there seems to be a strong relationship between commutativity and the semi-join optimization [6]); examine ways to take advantage of partial commutativity, i.e., when the transitive closure of a product of operators is to be computed, only a subset of which are mutually commutative; and examine ways to take advantage of commutativity appearing in some higher power of an operator, as in the case of recursive redundancy.

REFERENCES

1. Agrawal, R., and Jagadish, H. V., Direct Algorithms for Computing the Transitive Closure of Database Relations, in: *Proceedings of the 13th International VLDB Conference*, Brighton, U.K., 1987, pp. 255–266.
2. Aho, A., Hopcroft, J., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.
3. Aho, A., Sagiv, Y., and Ullman, J. D., Equivalences among Relational Expressions, *SIAM J. Comput.* 8(2):218–246 (1979).
4. Aho, A., and Ullman, J. D., University of Data Retrieval Languages, in: *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, San Antonio, Tex., 1979, pp. 110–117.
5. Bancilhon, F., Naive Evaluation of Recursively Defined Relations, in: *Proceedings of the Islamorada Workshop on Large-Scale Knowledge Base and Reasoning Systems*, Islamorada, Fla., 1985.
6. Beeri, C., and Ramakrishnan, R., On the Power of Magic, in: *Proceedings of the 6th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, San Diego, Calif., 1987, pp. 269–283.
7. Bondy, J. A., and Murty, U. S. R., *Graph Theory with Applications*, North-Holland, New York, 1976.
8. Chandra, A. K., and Merlin, P. M., Optimal Implementation of Conjunctive Queries in Relational Data Bases, in: *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, Boulder, Colo., 1977, pp. 77–90.
9. Dong, G., *On the Composition of Datalog Program Mappings*, Unpublished Manuscript, University of Southern California, Los Angeles, Calif., 1988.
10. Ioannidis, Y. E., A Time Bound on the Materialization of Some Recursively Defined Views, *Algorithmica* 1(4):361–385 (1986).
11. Ioannidis, Y. E., On the Computation of the Transitive Closure of Relational Operators, in: *Proceedings of the 12th International VLDB Conference*, Kyoto, Japan, 1986, pp. 403–411.
12. Ioannidis, Y. E., and Wong, E., Query Optimization by Simulated Annealing, in: *Proceedings of the 1987 ACM-SIGMOD Conference on the Management of Data*, San Francisco, Calif., 1987, pp. 9–22.
13. Ioannidis, Y. E., and Wong, E., Towards an Algebraic Theory of Recursion, *J. ACM* 38(2):329–381 (1991).
14. Lassez, J. L., and Maher, M. J., Closures and Fairness in the Semantics of Programming Logic, *Theoret. Comput. Sci.* 29:167–184 (1984).
15. Naughton, J., Compiling Separable Recursions, in: *Proceedings of the 1988 ACM-SIGMOD Conference on the Management of Data*, Chicago, Ill., 1988, pp. 312–319.
16. Naughton, J., Minimizing Function-Free Recursive Inference Rules, *J. ACM* 36(1):69–91 (1989).
17. Naughton, J., Data Independent Recursion in Deductive Databases, *J. Comput. Syst. Sci.* 38(2):259–289 (1989).
18. Plotkin, G. D., A Note on Inductive Generalization, in: B. Meltzer and D. Michie (eds.), *Machine Intelligence 5*, Edinburgh University Press, U.K., 1969, pp. 153–163.
19. Ramakrishnan, R., Sagiv, Y., Ullman, J. D., and Vardi, M., Proof Tree Transformation Theorems and Their Applications, in: *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadelphia, Penn., 1989, pp. 172–181.