COMMUTATIVITY AND ITS ROLE IN THE PROCESSING OF LINEAR RECURSION

Yannis E. Ioannidis

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

(Revised Version)

COMMUTATIVITY AND ITS ROLE IN THE PROCESSING OF LINEAR RECURSION [†]

Yannis E. Ioannidis [‡]

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Abstract

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 database 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 and other), 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.

Commutativity has already been identified as a significant special case of recursion [Ioan88]. 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 the system has to look at to answer a query with selections. This earlier work on commutativity was done within the algebraic framework of linear recursive operators (rules) [Ioan86a, Ioan88]. In this paper, we use the logic representation of rules to give conditions for two linear 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.

[†] 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.

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 [Ioan86a, Ioan88]. 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. Section 5 uses the logic representation of rules and gives 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 presents our conclusions and gives some directions for future work.

2. ALGEBRAIC MODEL

In this section, we provide a summary of the algebraic model for linear recursion [Ioan86a, Ioan88]. We use the terms relation and predicate indistinguishably. Consider the following pair of one linear recursive and one nonrecursive rule:

$$\mathbf{P}\,(\underline{x}^{(0)}) \wedge \mathbf{Q}_1(\underline{x}^{(1)}) \wedge \cdots \wedge \mathbf{Q}_k(\underline{x}^{(k)}) \rightarrow \mathbf{P}\,(\underline{x}^{(k+1)}), \tag{2.1}$$

$$\mathbf{Q}\,(\underline{x}^{(k+1)}) \rightarrow \mathbf{P}\,(\underline{x}^{(k+1)}), \tag{2.2}$$

where for each i, $\underline{x}^{(i)}$ 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, the rules can contain functions. Each one of $\mathbf{P}\,(\underline{x}^{(0)})$, $\mathbf{Q}\,(\underline{x}^{(k+1)})$, and the $\mathbf{Q}_i(\underline{x}^{(i)})$'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 $f(\mathbf{P}, \{\mathbf{Q}_i\})$ that has $\{\mathbf{Q}_i\}$ as parameters and accepts as input and produces as output relations of the same schema as $\mathbf{P}$:

$$f\,(\mathbf{P}\,, \{\mathbf{Q}_i\}) \subseteq \mathbf{P}\,.$$

The function f can be thought of as a linear relational operator applied to the recursive relation $\mathbf{P}$ 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

$(A * B)\mathbf{P} = A(B\mathbf{P})$ and *addition* by $(A+B)\mathbf{P} = A\mathbf{P} \cup B\mathbf{P}$. [†] For notational convenience we omit the operator $*$. The multiplicative identity ($1\mathbf{P} = \mathbf{P}$) and the additive identity ($0\mathbf{P} = \emptyset$, $\emptyset$ the empty set) are defined in obvious ways. The n-th 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 $A = B \iff \forall \mathbf{P}, A\mathbf{P} = B\mathbf{P}$. Finally, a partial order $\leq$ is defined on R as $A \leq B \iff \forall \mathbf{P}, A\mathbf{P} \subseteq B\mathbf{P}$. The set R with the above defined operations forms a *closed semiring* [Ioan88].

The above embedding of the linear relational operators in a closed semiring 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

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

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

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

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

The solution is a function of $\mathbf{Q}$. Hence, $\mathbf{P}$ can be written as $\mathbf{P} = B\,\mathbf{Q}$, and the problem becomes one of finding the operator B. Manipulation of (2.3) results in the elimination of $\mathbf{Q}$, so that the equation contains operators only. In this pure operator form, the recursion problem can be restated as follows. Given operator A, find B satisfying the following:

(a) $1 + AB = B$,

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

Theorem 2.1: [Ioan88] The solution of equation (2.4a) with restriction (2.4b) is $A^* = \sum_{k=0}^{\infty} A^k$.

The operator A^* is called the *transitive closure* of A. Theorem 2.1 is originally due to Tarski [Tars55], and in the database context, it was first examined by Aho and Ullman [Aho79b]. It is the first time though that the solution of (2.4) is expressed in an explicit algebraic form within an algebraic structure like the closed semiring of linear relational operators. The implications of the manipulative power thus afforded on the implementation of A^* are significant [Ioan86a, Ioan86b, Ioan87, Ioan88]. In this paper, we concentrate on the implications of commutativity of operators in the implementation of A^*.

[†] 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 [Ioan88].

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 [Chan77]. Also note that A^* represents an operator. The query answer is the result of applying A^* to a given relation **Q**. This is only an abstraction, however, that allows us to study recursion within the closed semiring of relational operators. It poses no restriction 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 $B^*C^*\mathbf{Q}$. The computation may proceed by first computing C^*, then applying it to **Q**, and then using *seminaive* [Banc85] 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 $B\,C=C\,B$. Consider computing A^*, the transitive closure of A, where $A=B+C$. It has been shown that if $B\,C\le C^kB^l$, for some k,l with $k\in\{0,1\}$ or $l\in\{0,1\}$, then $A^*=B^*C^*$ [Ioan88]. 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:

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

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 $C\,B$ 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 $B\le B+C$ and $C\le B+C$), 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 [Agra87]. Comparing the computations of $B^* C^*$ and $(B+C)^*$ with respect to duplicates, we can easily derive the superiority of $B^* C^*$ in certain special cases. For instance, if the computation of $B^* C^*$ is duplicate-free, then applying the decomposition is always beneficial (unless CB=0, in which case the second set of terms in (3.1) would not produce anything anyway). If the computation of $B^* C^*$ is not duplicate-free, however, it is conceivable that the various terms in (3.1) may interact in a way so that more duplicates are produced by $B^* C^*$ than by the full expansion of $(B+C)^*$, despite the additional terms of the latter. The following general result shows that this impossible.

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 $C_1 \cdots C_{k-1}C_k \in \{A_i\}$ $(\{B_i\})$ then $C_1 \cdots C_{k-1} \in \{A_i\}$ $(\{B_i\})$ as well. Consider two linear operators A,B, where A=B, $A=\sum\limits_i A_i$, and $B=\sum\limits_i B_i$. Let **Q** be an arbitrary relation and **T** be equal to **T** = A**Q** = B**Q**. If $\{A_i\} \subseteq \{B_i\}$, then the evaluation of **T** based on A produces no more duplicates than its evaluation based on B.

Proof: Let the *derivation graph* of a computation of **T** be a labeled directed graph $G=(V,E,L:E \rightarrow \{C_i\})$, 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 = **T**, i.e., the nodes of G are the tuples in **T**

$E = \{(t_1 \rightarrow t_2) \mid t_2$ is produced by applying one operator from $\{C_i\}$ on $t_1\}$

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

Since there is a 1-1 correspondence between nodes and tuples, we shall use the two terms indistinguishably. We assume a model of computation that starts at the tuples in **Q** and traverses the graph until all nodes are visited at least once. We also assume computations that do not derive the same tuple through the same arc more than once. [†] Such a computation can be achieved, for example, by employing the semi-naive evaluation [Banc85].

A path in the graph from a tuple s in **Q** to a tuple t represents a derivation of t starting from s. The labels of the arcs along the path represent 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 **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 **T** plus the

[†] 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.

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 **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 **T** based on B. If the computation by B 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 **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 $(B+C)^* = B^* C^*$, which in several cases decrease 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.

3.2. Previous Work

Commutativity or properties related to it has been rarely addressed in the past. The earliest result of which we are aware that is related to commutativity is by Lassez and Maher [Lass84]. 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:

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

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

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 [Rama89]. Their condition is less general than the one we give 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 things, that study lead to an independent discovery of the above mentioned result that if $B\ C \le C^k B^l$, 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 [Dong88]. The only result that involves commutativity in a significant way can be expressed as follows in algebraic form:

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

4. COMMUTATIVITY VS. SEPARABILITY AND RECURSIVE REDUNDANCY

4.1. Commutativity vs. Separability

Separable recursions have been identified by Naughton as an important class of linear recursion where efficient algorithms can be applied [Naug88]. In this section, we shall show that the efficient separable algorithm is applicable to the class of commutative recursions. For the sake of simplicity, we shall concentrate on two operators A_1 and A_2. The extensions of the results to an arbitrary number of operators is straightforward.

Theorem 4.1: Given two operators A_1 and A_2 that commute, and a selection σ that commutes with one of them, 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 $(A_1+A_2)^* = A_1^*\ A_2^*$ [Ioan88]. Given an initial relation $\mathbf{q}$ and a query with a selection σ that commutes with A_1, we have

$$\sigma(A_1+A_2)^*\mathbf{q} = A_1^*\ (\sigma A_2^*)\mathbf{q}. \tag{4.1}$$

To take advantage of the selection, the following algorithm can be used to derive the query answer given in (4.1). The variables B and C contain operators, whereas the variables **R** and **S** contain relations. Multiplication of operators is shown explicitly for readability.

```
$B := \sigma$;
$C := \sigma$;
repeat
    $B := B * A_2$;
    $C := B + C$;
until $C$ doesn't change
R := $C$ q;
S := R;
repeat
    R := $A_1$ R;
    S := S $\cup$ R;
until S doesn't change
```

The first loop actually involves manipulating relations that are parameters of the various operators. If this is taken into account, and some small optimizations are incorporated so that, in every application of an operator inside each loop, only the new tuples produced in the previous iteration are used, the above algorithm is precisely the one proposed for separable recursions with full selections [†] [Naug88]. □

Theorem 4.1 establishes that commutativity implies the applicability of the separable algorithm for two operators. In general, 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 will not be present. In the presense 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 [Naug89a]. Consider an operator A that is the product of a set of operators $\{A_i\}$, i.e., $A = A_1 A_2 \cdots A_n$. In this case, every term in the series $A^* = \sum_{k=0}^{\infty} A^k$ is an arbitrary product of the A_i's. An operator A_i, $1 \le i \le n$, is *recursively redundant* if there is some N such that each term in the series of A^* is equal to a product containing A_i less than N times. The nonrecursive predicates appearing in A_i 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. Finally, for $A = BC$ and $A^L = DE$, the operator D is *independent* of B if no parameter relation of D is produced from a parameter relation of B. The effect of the presense 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 [Ioan88]. (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.)

[†] The precise definition of full selections is given by Naughton [Naug88]. The key observation is that if $A_1 A_2 = A_2 A_1$ and $\sigma A_1 = A_1 \sigma$, then σ is a full selection.

Theorem 4.2: Let $A{=}B\,C$. If B is torsion and there exist $L\geq 1$ and an operator D that is independent of B (i.e., it depends only on C) such that

$$A^L{=}B^L D{=}D\,B^L, \tag{4.1}$$

then B is recursively redundant in $A^*{=}(B\,C)^*$.

Proof: Consider an operator A that satisfies the premises of the theorem. Let, $K>0$ and $N>0$, $K<N$, be the smallest numbers such that $B^N=B^K$. Since $B^K=B^N$, clearly, the equality $B^{KL}{=}B^{NL}$ holds. It takes an easy induction to show that

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

The main result follows from the derivation below:

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

Note that, since D is independent of B, B^{NL-1} is the highest power of B used in any term of A^*, i.e., B is recursively redundant. Clearly, the above formula corresponds to a more efficient algorithm than processing A as a whole, since B is processed only for a fixed finite number of times, i.e., $N\,L-1$, beyond which only D 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 **P**, its underlying nonrecursive one is constructed by replacing the instance of **P** in its consequent by $\mathbf{P}_O$ (output), and its instance of **P** in its antecedent by $\mathbf{P}_I$ (input). However, we shall still be 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 $Q(x_1,...,x_n)$ appears in the antecedent of r, then $Q(f(x_1),...,f(x_n))$ 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 [Chan77, Aho79a]. 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. 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 [Chan77, Aho79a].

5.1. A Sufficient Condition

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

As a notation vehicle, we shall use a version of the α-graph of a rule (which we shall also call α-graph), which was introduced for the study of uniform boundedness [Ioan85]. 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 **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 **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 **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 **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 they are defined for undirected graphs, we shall use them for directed graphs as well with the understanding that we always refer to the 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 shall 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 $x_{(i+1) \, mod \, n}$ appears in the recursive predicate in the consequent (i.e., their positions 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 arcs of the form $(x_i \rightarrow x_{(i+1) \, mod \, n})$.

Finally, we need to define some interesting subgraphs of the α-graph of a rule [Bond76]. 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 ~ 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 ~ is an equivalence relation on the edges of G. A subgraph of G induced by the edges of an equivalence class under the relation ~ 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. Also note that an augmented bridge of the α-graph of a rule is an α-graph in itself and corresponds to a rule. Based on that, containment and equivalence of augmented bridges are defined appropriately as containment and equivalence of the corresponding rules.

Example 5.1: The following is the α-graph of the rule

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

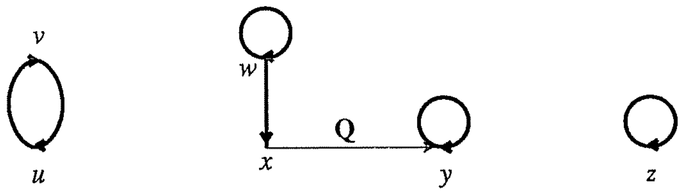


Figure 5.1: Example of an α-graph.

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.

For another example, the following is the α-graph of the rule

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

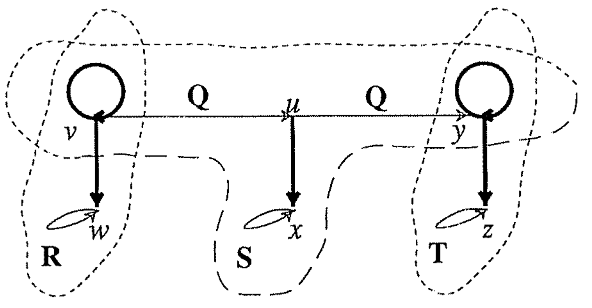


Figure 5.2: Augmented bridges in an α-graph.

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 rules are the following:

$$\mathbf{P}\,(v,w) :- \mathbf{P}\,(v,v) \wedge \mathbf{R}\,(w),$$

$$\mathbf{P}\,(v,x,y) :- \mathbf{P}\,(v,u,y) \wedge \mathbf{Q}\,(v,u,y) \wedge \mathbf{S}\,(x),$$

$$\mathbf{P}\,(y,z) :- \mathbf{P}\,(y,y) \wedge \mathbf{T}\,(z).$$

□

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. Note that, if $h(x)=y$, then there exists a dynamic arc $(y \rightarrow x)$ in the α-graph of the rule. We may also define powers of h as

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

For two rules r_1 and r_2, we 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

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

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

The following theorem gives a sufficient condition for commutativity of rules of the form specified in the beginning of Section 5. Another sufficient condition for commutativity has been independently discovered and reported elsewhere [Rama89].

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 n_1-persistent, $n_1>1$, in r_1 and free n_2-persistent, $n_2>1$, in r_2 and $h_1(h_2(x))=h_2(h_1(x))$,
(d) x is link n-persistent, $n>1$, or general and belongs to equivalent augmented bridges in both r_1 and r_2.

Proof: In the proof, we use the fact that commutativity of r_1 and r_2 is defined as $r_1 r_2$ and $r_2 r_1$ being equivalent, 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 n-persistent, $n>1$, variables in the consequent of r_1 and r_2,
$\underline{e}$	vector of the link n-persistent, $n>1$, 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 the latter can be written in the following form:

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

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

We have assumed that every rule seen as a conjunctive query is in its unique minimal form [Chan77]. 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 **S**. $\mathbf{Q}_1$, $\mathbf{Q}_2$ represent the remaining nonrecursive predicates, i.e., those of bridges whose general and link n-persistent variables, $n>1$, 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:

$$r_1 r_2\text{:}\quad \mathbf{P}_O(\underline{p}_1,\underline{p}_2,\underline{s},\underline{c},\underline{e}) :\!- \mathbf{P}_I(\underline{z}_2,\underline{z}_1,\underline{s},h_1(h_2(\underline{c})),g_{12}(\underline{v}_2)) \wedge \mathbf{S}(\underline{u}_1) \wedge \mathbf{S}(g_{12}(\underline{u}_2)) \wedge \mathbf{Q}_1(\underline{w}_1) \wedge \mathbf{Q}_2(\underline{w}_2),$$

$$r_2 r_1\text{:}\quad \mathbf{P}_O(\underline{p}_1,\underline{p}_2,\underline{s},\underline{c},\underline{e}) :\!- \mathbf{P}_I(\underline{z}_2,\underline{z}_1,\underline{s},h_2(h_1(\underline{c})),g_{21}(\underline{v}_1)) \wedge \mathbf{S}(g_{21}(\underline{u}_1)) \wedge \mathbf{S}(\underline{u}_2) \wedge \mathbf{Q}_1(\underline{w}_1) \wedge \mathbf{Q}_2(\underline{w}_2).$$

We only explain the formation of $r_1 r_2$, since $r_2 r_1$ is formed similarly. $\mathbf{S}(\underline{u}_1)$ remains as is from r_1. The variables of $\underline{u}_2$ in **S** change according to $g_{12}(\underline{u}_2)$ to produce $\mathbf{S}(g_{12}(\underline{u}_2))$. $\mathbf{Q}_1(\underline{w}_1)$ 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 $\mathbf{Q}_2(\underline{w}_2)$. The variables in $\mathbf{P}_I$ 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. 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, which implies that they are equivalent. Therefore, the two original rules commute. □

Example 5.2: The following two rules commute with each other.

$$\mathbf{P}_O(x,y,z) :- \mathbf{P}_I(u,y,z) \wedge \mathbf{Q}\ (x,y)$$

$$\mathbf{P}_O(x,y,z) :- \mathbf{P}_I(x,y,v) \wedge \mathbf{R}\ (z,y)$$

Both composites are equal to the rule below.

$$\mathbf{P}_O(x,y,z) :- \mathbf{P}_I(u,y,v) \wedge \mathbf{Q}\ (z,y) \wedge \mathbf{R}\ (x,y)$$

The α-graphs of the two rules are shown in Figure 5.3.

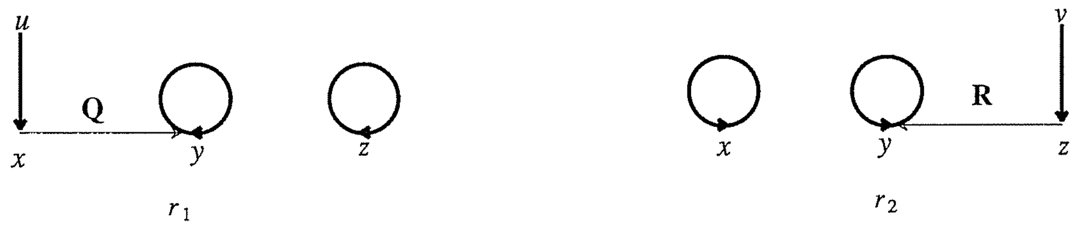


Figure 5.3: α-graphs of commuting rules satisfying the condition of Theorem 5.1.

Note that the condition of Theorem 5.1 is satisfied by the corresponding α-graphs. □

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

Example 5.3: The following two rules also commute with each other.

$$\mathbf{P}_O(x,y) :- \mathbf{P}_I(y,w) \wedge \mathbf{Q}(x)$$

$$\mathbf{P}_O(x,y) :- \mathbf{P}_I(u,v) \wedge \mathbf{Q}(x) \wedge \mathbf{Q}(y)$$

Both composites are isomorphic to the rule below.

$$\mathbf{P}_O(x,y) :- \mathbf{P}_I(u,v) \wedge \mathbf{Q}(y) \wedge \mathbf{Q}(w) \wedge \mathbf{Q}(x)$$

The α-graphs of the two rules are shown in Figure 5.4.

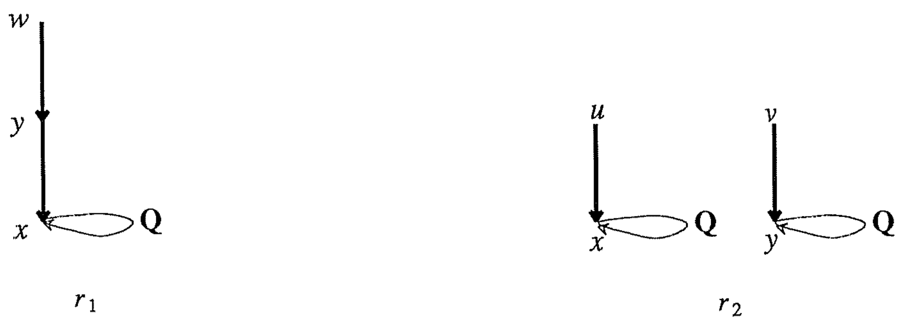


Figure 5.4: α-graphs of commuting rules not satisfying the condition of Theorem 5.1.

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 antecedent. 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.

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:

$$r_1: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(x', \cdots) \wedge \cdots,$$

$$r_2: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(x'', \cdots) \wedge \cdots.$$

The two composites are

$$r_1 r_2: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(g_{12}(x''), \cdots) \wedge \cdots,$$

$$r_2 r_1: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(g_{21}(x'), \cdots) \wedge \cdots.$$

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 **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 **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 **Q** in r_2, but we shall examine both cases, when it exists and when it does not. The two different instances of **Q** will be distinguished by superscripts.

$$r_1: \quad \mathbf{P}_O(x_0, \cdots, x_k, \cdots) :- \mathbf{P}_I(x_1, \cdots, x_{k+1}, \cdots) \wedge \mathbf{Q}^1(x_0, \cdots) \wedge \cdots,$$

$$r_2: \quad \mathbf{P}_O(x_0, \cdots, x_k, \cdots) :- \mathbf{P}_I(y_0, \cdots, y_k, \cdots) \wedge \mathbf{Q}^2(z, \cdots) \wedge \cdots.$$

Composing the two rules we have

$$r_1 r_2: \quad \mathbf{P}_O(x_0, \cdots, x_k, \cdots) :- \mathbf{P}_I(g_{12}(y_0), \cdots, g_{12}(y_k), \cdots) \wedge \mathbf{Q}^1(x_0, \cdots) \wedge \mathbf{Q}^2(g_{12}(z), \cdots) \wedge \cdots,$$

$$r_2 r_1: \quad \mathbf{P}_O(x_0, \cdots, x_k, \cdots) :- \mathbf{P}_I(y_1, \cdots, y_{k+1}, \cdots) \wedge \mathbf{Q}^1(y_0, \cdots) \wedge \mathbf{Q}^2(z, \cdots) \wedge \cdots.$$

Examining the two composites we distinguish two cases. If **Q** does not appear in r_2 (i.e., if we ignore $\mathbf{Q}^2$), 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 shall 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 $\mathbf{P}_I$ in r_1r_2 must map to the instance of $\mathbf{P}_I$ 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 shall 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 $\mathbf{P}_I$ 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 n_1-persistent, $n_1>1$, in r_1 and free n_2-persistent, $n_2>1$, in r_2 and $h_1(h_2(x))=h_2(h_1(x))$,
(d) x is link n-persistent, $n>1$, 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 will be 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 **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 n-persistent variable, $n>1$: Since the rules are range-restricted, if x is not a free m-persistent variable , $m \ge 1$, in r_2, then there exists 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 **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 **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 n-persistent variable, $n>1$, in r_1. Hence, x must be a free m-persistent, $m \ge 1$, variable in r_2 also.

If $m=1$, i.e., it is free 1-persistent ((a) holds), then x satisfies the theorem. Otherwise, x is free m-persistent, $m>1$, 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 $h_2(x)$ also 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 n-persistent, $n>1$, or general variable: Again, 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 $x=x_{n+1}$, 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 shall 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 shall 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 n-persistent, $n>1$, or general variable in its augmented bridge, we can conclude that for any such variable z in that 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.

$$r_1: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(y, \cdots) \wedge \mathbf{R}_1(x,z_1) \wedge \mathbf{R}_2(z_1,z_2) \cdots \wedge \mathbf{R}_{m-1}(z_{m-2},z_{m-1}) \wedge \mathbf{R}_m(z_{m-1},z_m) \wedge \cdots,$$

$$r_2: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(y, \cdots) \wedge \mathbf{R}_1(x,z'_1) \wedge \mathbf{R}_2(z'_1,z'_2) \cdots \wedge \mathbf{R}_{m-1}(z'_{m-2},z'_{m-1}) \wedge \cdots.$$

Composing the two rules results in the following:

$$r_1 r_2: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(\_, \cdots) \wedge \mathbf{R}_1(y,z'_1) \wedge \mathbf{R}_2(z'_1,z'_2) \cdots \wedge \mathbf{R}_{m-1}(z'_{m-2},z'_{m-1}) \wedge$$
$$\mathbf{R}_1(x,z_1) \wedge \mathbf{R}_2(z_1,z_2) \cdots \wedge \mathbf{R}_{m-1}(z_{m-2},z_{m-1}) \wedge \mathbf{R}_m(z_{m-1},z_m) \wedge \cdots,$$

$$r_2 r_1: \quad \mathbf{P}_O(x, \cdots) :- \mathbf{P}_I(\_, \cdots) \wedge \mathbf{R}_1(y,z_1) \wedge \mathbf{R}_2(z_1,z_2) \cdots \wedge \mathbf{R}_{m-1}(z_{m-2},z_{m-1}) \wedge \mathbf{R}_m(z_{m-1},z_m) \wedge$$
$$\mathbf{R}_1(x,z'_1) \wedge \mathbf{R}_2(z'_1,z'_2) \cdots \wedge \mathbf{R}_{m-1}(z'_{m-2},z'_{m-1}) \wedge \cdots.$$

Clearly, since $y \ne x$ (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 range-restricted two 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 [Aho74]. 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 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: Testing equivalence of two range-restricted rules with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent can be done in $O(p\log p+a)$, where p is the maximum number of predicates in the antecedents of the two rules and a is the total number of argument positions in those predicates.

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; this takes $O(p \log p)$ time.

(2) Define f such that, for any pair of literals $\mathbf{Q}(x_1, \cdots, x_n)$ in the antecedent of r_1 and $\mathbf{Q}(y_1, \cdots, y_n)$ in the antecedent of r_2, $f(x_i)=y_i$. If f is 1-1 (and onto) and $x_i=y_i$ when x_i is distinguished, then f is an isomorphism between the two rules, which are thus equivalent. Otherwise, they are not equivalent. This step takes $O(1)$ time for every variable instance, i.e., argument position, in the antecedent of the rules. Thus, this step takes $O(a)$ 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 $O(p \log p+a)$. □

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 $O(n \log n + a_r + a_n)$ time, where a_r is the number of argument positions of the recursive predicate, a_n is the total number of argument positions in the nonrecursive predicates, and n is the maximum number of nonrecursive predicates in the rules.

Proof: The algorithm has the following steps.

(1) Identify the type of every distinguished variable (i.e., free 1-persistent, link 1-persistent, free n-persistent, $n > 1$, link n-persistent, $n > 1$, or general), and then identify the bridges of the underlying undirected graphs of the α-graphs of the two rules, The quantity $a_r + a_n$ 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 $O(a_r + a_n)$ time.

(2) 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 $O(a_r)$.

(3) For every free n-persistent variable, $n > 1$, in the one rule, check if it is free m-persistent, $m \geq 1$, 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 n-persistent variable, $n > 1$, for a total of $O(a_r)$ time.

(4) For every link n-persistent, $n > 1$, 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 $O(a_r)$ 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 $O(n \log n + a_r + a_n)$ time.

The total complexity is given by the sum of the total times for steps (1), (2), (3), and (4), which is equal to $O(n \log n + a_r + a_n)$. $\square$

6. SEPARABILITY AND RECURSIVE REDUNDANCY REVISITED

In Section 3, 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 4 to obtain more relationships of commutativity with separability and recursive redundancy for this class of rules.

6.1. Commutativity vs. Separability

Separable rules were defined as follows [†] [Naug88]. Two rules r_1 and r_2 with the same consequent are separable if

[†] The definition given in this paper can be easily extended to multiple rules (in accordance to the original definition [Naug88]). For presentation clarity, however, we restrict ourselves to two rules.

(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.

We ignore a fourth clause that the original definition contained, since it is nonessential for the correctness of the separable algorithm. 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, 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 separable 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 6.1: 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.

The rules of Example 5.2 serve as examples of commutative rules that are not separable. They violate both condition (2) and condition (3) of the separable definition. □

By Theorem 6.1, 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.

6.2. Commutativity vs. Recursive Redundancy

Until now, we were exclusively dealing with the bridges of α-graphs with respect to the subgraph induced by the dynamic arcs connecting each link 1-persistent variable in the graph with itself. In the study of recursive redundancy, however, an important role is played by the bridges of the α-graph of a rule with respect to the subgraph induced by the dynamic arcs connecting any link persistent variables in the graph (not only 1-persistent ones). A necessary and sufficient condition for a nonrecursive predicate in a rule of some restricted form to be redundant was originally given by Naughton [Naug89a]. Using a different terminology, that condition is expressed in the following theorem.

Theorem 6.2: [Naug89a] 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 the subgraph induced by the arcs connecting its link persistent variables.

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

Lemma 6.2: † 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 [Naug89b], for the class of rules defined in the lemma, we can find $k>0$ and $\tau>0$ such that r^k and $r^{k+\tau}$ are of the form

$$\begin{aligned} r^k&: \quad b\,a, \\ r^{k+\tau}&: \quad b\,q\,a, \end{aligned}$$

where a, b, and q are conjunctions of predicates that mutually share no nondistinguished variables. Naughton showed that there is a homomorphism $f: r^k \rightarrow r^{k+\tau}$ such that $f(b)=b$ and $f(a)=a$ [Naug89b]. Consider $r^{k+2\tau}$. Clearly, it can be written in the form

$$r^{k+2\tau}: \quad b\,q\,q'\,a,$$

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

† Similar results are easily provable for the class of recursions examined by Ioannidis [Ioan85].

$f_1(a)=a, f_1(b)=b, f_1(q)=q$

$f_2(a)=a, f_2(b)=b, f_2(q)=q, f_2(q')=q$

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

In the following, for a given rule r (corresponding to some operator A) and $L \geq 1$, I is the set of the link persistent variables in r (A), G_I is the subgraph of the α-graph of r (A) induced by the dynamic arcs connecting the variables in I, and G_I^L is the subgraph of the α-graph of r^L (A^L) induced by the dynamic arcs connecting the variables in I as well.

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.

(a) For all $L \geq 1$, I is the set of link persistent variables of A^L.
(b) There exists $L \geq 1$ such that all variables in I are link 1-persistent in A^L.

Proof: Part (a) is obvious. For (b), clearly, any link L_x-persistent variable x in A is link 1-persistent in A^{L_x}. Choose $L = \underset{x \in I}{lcm}\{L_x\}$, the least common multiple of $\{L_x\}$. It is easy to see that all link persistent variables in A are link 1-persistent in A^L, and that no other variable satisfies that (because of (a)). □

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. Consider $L \geq 1$ as defined by Lemma 6.3 (b). A bridge in the α-graph of A with respect to G_I generates one or more bridges in the α-graph of A^L with respect to G_I^L.

Proof: 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 $(w_1' \rightarrow w_2')$ be the arcs generated by $(z_1 \rightarrow z_2)$ and $(w_1 \rightarrow w_2)$ respectively in A^k, $k>1$. If $(z_1' \rightarrow z_2')$ and $(w_1' \rightarrow w_2')$ are not connected in the α-graph of A^k, then vacuously they belong to different bridges with respect to G_I^k. 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 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 $l \leq k$. Since x is link persistent, however, it is only replaced by link persistent 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 as well. In the α-graph of A^L, all those variables are link 1-persistent. This implies that $(z_1' \rightarrow z_2')$ and $(w_1' \rightarrow w_2')$ belong to different bridges with respect to G_I^L. □

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 6.3: Let $A=B\,C$ be an operator corresponding to a range-restricted rule with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent. Then, B is recursively redundant in $A^*=(B\,C)^*$ if and only if B is uniformly bounded and there exist $L\geq 1$ and an operator D that is independent of B (i.e., it depends only on C) such that

$$A^L=B^L D=D\,B^L .$$

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

For the *only if* part, assume that B is recursively redundant. By Theorem 6.2, the nonrecursive predicates in B belong to a set of uniformly bounded augmented bridges in the α-graph of the rule with respect to G_I. Let I_B (I_C) be the distinguished variables in the bridges of B (C) excluding those in I. Thus, $I\cup I_B\cup I_C$ forms the set of all distinguished variables, whereas $I\cap I_B=I_B\cap I_C=I_C\cap I=\emptyset$. The α-graph of B can be constructed as follows: in the α-graph of A, keep the augmented bridges containing the nonrecursive predicates of B (i.e., its parameters) unchanged, remove all remaining arcs, and introduce dynamic arcs so that the remaining distinguished variables (I_C) become free 1-persistent. Clearly, B is uniformly bounded. By Lemma 6.3, there exists $L\geq 1$ such that all link persistent variables in A are link 1-persistent in A^L. Moreover, by Lemma 6.4, B^L (excluding the free 1-persistent variables of I_C) corresponds to some set of augmented bridges in the α-graph of A^L with respect to G_I^L. Let D be the operator that corresponds to the set of the remaining augmented bridges in the α-graph of A^L. The α-graph of D can be constructed as follows: in the α-graph of A^L, remove all arcs from the (nonaugmented) bridges that correspond to B^L, introduce dynamic arcs so that their distinguished variables (I_B) become free 1-persistent, and keep the rest of the graph unchanged. By the way D was defined, $A^L=B^L D$. Moreover, the distinguished variables in I_B or in I_C are free 1-persistent in at least one of B^L and D, whereas those in I are link 1-persistent in both B^L and D. Thus, by Theorem 5.2, B^L and D commute. This implies that for range-restricted rules with no repeated variables in the consequent and no repeated nonrecursive predicates in the antecedent, the condition of Theorem 4.2 is necessary and sufficient for recursive redundancy. □

Example 6.1: Let A be the operator corresponding to the rule

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

The α-graph of A is shown below:

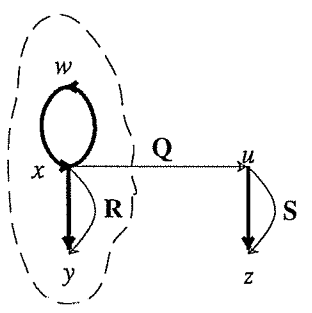


Figure 6.1: α-graph of rule with recursively redundant predicates.

The roles of B and C are played by the following rules:

B : **P** (w,x,y,z) :– **P** $(x,w,x,z) \wedge$ **R** (x,y),

C : **P** (w,x,y,z) :– **P** $(w,x,y,u) \wedge$ **Q** $(w,u) \wedge$ **S** (u,z).

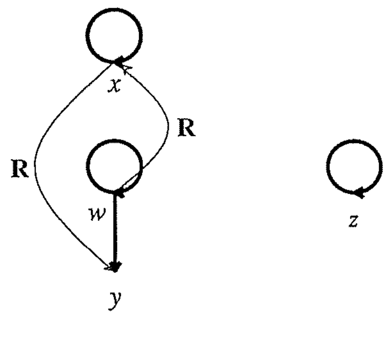
Clearly, $A=BC$ and B is uniformly bounded (it has no nondistinguished variables). The nonrecursive predicate **R** is recursively redundant according to Theorem 6.2 and its augmented bridge with respect to G_I is enclosed in dotted boundaries in Figure 6.1. Theorem 6.3 is satisfied for this example for $L=2$. The rules corresponding to operators A^2, B^2, and D are shown below:

A^2 : **P** (w,x,y,z) :– **P** $(w,x,w,u') \wedge$ **Q** $(w,u') \wedge$ **R** $(w,x) \wedge$ **S** $(u',u) \wedge$ **Q** $(x,u) \wedge$ **R** $(x,y) \wedge$ **S** (u,z),

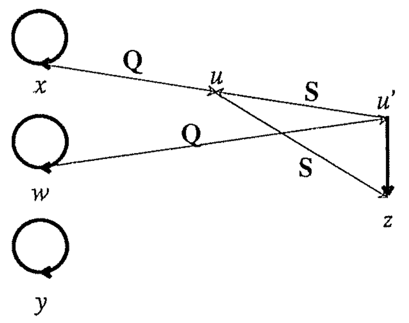
B^2 : **P** (w,x,y,z) :– **P** $(w,x,w,z) \wedge$ **R** $(w,x) \wedge$ **R** (x,y),

D : **P** (w,x,y,z) :– **P** $(w,x,y,u') \wedge$ **Q** $(w,u') \wedge$ **S** $(u',u) \wedge$ **Q** $(x,u) \wedge$ **S** (u,z).

Again, one can verify that $A^2=B^2D$. The α-graphs of B^2 and D are shown below:



B^2



D

Figure 6.2: α-graphs of commuting rules B^2 and D, where $A=BC$, $A^2=B^2D$, and B is recursively redundant.

Variables w and x are link 1-persistent in both B^2 and D, whereas y is free 1-persistent in D and z is free 1-persistent in

B^2. By theorem 5.2, B^2 and D commute, i.e., Theorem 6.3 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 semijoin optimization [Beer87]); 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.

8. REFERENCES

[Agra87]
Agrawal, R. and H. V. Jagadish, "Direct Algorithms for Computing the Transitive Closure of Database Relations", in *Proc. 13th International VLDB Conference*, Brighton, England, September 1987, pp. 255-266.

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

[Aho79a]
Aho, A., Y. Sagiv, and J. Ullman, "Equivalences Among Relational Expressions", *SIAM Computing Journal* **8**, 2

(May 1979), pp. 218-246.

[Aho79b]
Aho, A. and J. Ullman, "Universality of Data Retrieval Languages", in *Proc. of the 6th ACM Symposium on Principles of Programming Languages*, San Antonio, TX, January 1979, pp. 110-117.

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

[Beer87]
Beeri, C. and R. Ramakrishnan, "On the Power of Magic", in *Proc. of the 6th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, San Diego, CA, March 1987, pp. 269-283.

[Bond76]
Bondy, J. A. and U. S. R. Murty, *Graph Theory with Applications,* North Holland, 1976.

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

[Dong88]
Dong, G., *"On the Composition of Datalog Program Mappings"*, Unpublished Manuscript, University of Southern California, November 1988.

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

[Ioan86a]
Ioannidis, Y. E. and E. Wong, "An Algebraic Approach to Recursive Inference", in *Proc. of the 1st International Conference on Expert Database Systems*, Charleston, South Carolina, April 1986, pp. 209-223.

[Ioan86b]
Ioannidis, Y. E., "On the Computation of the Transitive Closure of Relational Operators", in *Proc. 12th International VLDB Conference*, Kyoto, Japan, August 1986, pp. 403-411.

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

[Ioan88]
Ioannidis, Y. E. and E. Wong, *"Towards an Algebraic Theory of Recursion"*, Technical Report No. 801, University of Wisconsin, Madison (submitted for publication), October 1988.

[Lass84]
Lassez, J. L. and M. J. Maher, "Closures and Fairness in the Semantics of Programming Logic", *Theoretical Computer Science* **29** (1984), pp. 167-184.

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

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

[Naug89b]
Naughton, J., "Data Independent Recursion in Deductive Databases", *Journal of Computer and System Sciences* **38**, 2 (April 1989), pp. 259-289.

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

base Systems, March 1989, pp. 172-181.

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

COMMUTATIVITY AND ITS ROLE IN THE PROCESSING OF LINEAR RECURSION

by

Yannis E. Ioannidis

Computer Sciences Technical Report #804

December 1989

(Revised Version)