ELSEVIER

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT

Information and Computation 188 (2004) 77–98

Information and Computation

www.elsevier.com/locate/ic

Generalization of ZYT-linearizability for bilinear datalog programs

Ji-Hoon Kang,[a,*] Ki-Hyung Hong,[b] Kyu-Young Whang,[c] and Jung-Wan Cho[d]

[a]*Department of Computer Science, Chungnam National University, 220 Gung-Dong, Yuseong-Gu, Daejeon 305-764, Republic of Korea*
[b]*School of Media and Information, Sungshin Women's University, Republic of Korea*
[c]*Department of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology, Republic of Korea*
[d]*Department of Computer Science, Korea Advanced Institute of Science and Technology, Republic of Korea*

Received 2 October 2000; revised 15 October 2002

Abstract

We propose a new type of linearizability, called right-linear-first (RLF) linearizability. The well-known ZYT-linearizability deals with only one bilinear rule. RLF-linearizability is a generalization of ZYT-linearizability since RLF-linearizability deals with general bilinear datalog programs consisting of multiple bilinear and linear rules. We identify sufficient conditions for RLF-linearizability. The test of the sufficient conditions is exponential in the size of the input datalog program, which is, however, usually very small compared with the size of the extensional database in deductive database applications.

Keywords: Deductive databases; Datalog programs; Bilinear rules; Linearizability

1. Introduction

If a nonlinear program can be linearized, it is possible to process queries on the program efficiently by using well-known cost-effective techniques [1,2,6,8,15,17,18] for linear programs. Since linearizability of general nonlinear datalog programs is undecidable [5], the researches on linearizability progress toward identifying the more and more larger class of linearizable programs.

[*] Corresponding author. Fax: +82-42-822-9959.
*E-mail address:*hkang@cs.chungnam.ac.kr (J.-H. Kang).

0890-5401/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0890-5401(03)00172-X

The first work on linearizability, called *ZYT-linearizability* [9,11,20], dealt with a very simple class of nonlinear programs that consist of two rules: a *bilinear* rule and an exit rule. The bilinear rule is a nonlinear rule with exactly two recursive subgoals in its body. The following program $\mathcal{Z}$, which consists of one exit rule r_e and one bilinear rule r_b, shows a typical form of such programs:

$$
\begin{array}{ll}
[\mathcal{Z}] & r_e : p(X_1, \ldots , X_t) \text{ :- } e(X_1, \ldots , X_t). \\
 & r_b : p(X_1, \ldots , X_t) \text{ :- } p(Y_1, \ldots , Y_t),\ p(Z_1, \ldots , Z_t),\ H.
\end{array}
$$

Here, H is a conjunction of extensional database (EDB) subgoals. ZYT-linearization transforms the program $\mathcal{Z}$ into a linear program $\mathcal{Z}^{zyt}$ by replacing a recursive subgoal of the bilinear rule r_b with the body of the exit rule r_e.

$$
\begin{array}{ll}
[\mathcal{Z}^{zyt}] & r_e : p(X_1, \ldots , X_t) \text{ :- } e(X_1, \ldots , X_t). \\
 & r'_b : p(X_1, \ldots , X_t) \text{ :- } p(Y_1, \ldots , Y_t), e(Z_1, \ldots , Z_t), H.
\end{array}
$$

The original program $\mathcal{Z}$ is called *ZYT-linearizable* if it is logically equivalent to $\mathcal{Z}^{zyt}$.

The ZYT-linearizability was originally proposed by Zhang et al. [19,20]. They found a necessary and sufficient condition for linearizing bilinear rules with at most one EDB subgoal in their bodies. Saraiya [11] extended the work of Zang et al. to the bilinear rules that have multiple EDB subgoals without repetition of the same EDB predicate. He also found a necessary and sufficient condition for linearizing the bilinear rules. He [13] further extended his work by allowing more than two recursive subgoals and the repetition of the same EDB predicates in the nonlinear rules. Ramakrishnan et al. [9] dealt with bilinear rules having more than one EDB subgoal possibly with the same predicate, and identified the largest class of bilinear rules that are linearizable by ZYT-linearization. Their approach is based on the concept of conjunctive-query containment [3,4,7,10]. They proposed a necessary and sufficient condition, but failed to find any way to check the condition. Instead, they proposed a testable but sufficient condition using the concept of uniform equivalence [10].

In [12], Saraiya defined *base-case linearizability* that is an extension of ZYT-linearizability for general nonlinear datalog programs including multiple nonlinear rules possibly with more than two recursive subgoals. He showed that base-case linearizability is undecidable. He never presented any sufficient condition for such nonlinear programs.

In this paper, we consider linearization of general bilinear programs that are nonlinear programs with multiple bilinear rules and together with multiple linear rules.

1. We propose a transformation method linearizing bilinear programs, called *right-linear-first linearization* (*RLF-linearization* for short). A bilinear program is called *RLF-linearizable* if it is logically equivalent to its RLF-linearized program.
2. We identify sufficient conditions for RLF-linearizability of the following two restricted types of general bilinear programs by utilizing the result [9] on ZYT-linearizability:
   (a) *MB-type (only multiple bilinear):* MB-type programs have no linear rule. That is, all the recursive rules are bilinear.
   (b) *SBSL-type (single bilinear and single linear):* SBSL-type programs have exactly two recursive rules. One is bilinear and the other linear.
3. Using the results on the above two types, we find a sufficient condition for RLF-linearizability of general bilinear programs. The condition is testable in exponential time. The test is only exponential

Table 1
Comparison of the results from the work on linearizability

Linearization method	ZYT				RLF	Base-case
	ZYT90 [20]	Sar89 [11]	RSUV93 [9]	Sar95 [13]	Our work	Sar90 [12]
No. of nonlinear rules	Only one	Only one	Only one	Only one	1 or more	1 or more
Degree of nonlinearity	2	2	2	2 or more	2	2 or more
No. of linear rules	0	0	0	0	0 or more	0 or more
No. of exit rules	1	1	1	1	1	1 or more
No. of EDB subgoals	0 or 1	0 or more	0 or more	0 or more	0 or more	0 or more
Duplication	No	No	Yes	Yes	Yes	Yes
Condition proved	iff	iff	iff	if	if	No
Testing algorithm	Yes	Yes	Sufficiency only	Yes	Yes	No
Time complexity	Polynomial	Polynomial	Exponential	Polynomial	Exponential	—

in the size of the input datalog program, which is, however, usually very small compared with the size of the extensional database in deductive database applications.

Table 1 summarizes the work on linearizability including this work, and compares the previous results with ours. In the table, the upper part shows the form of nonlinear programs being considered in each study: ‘no. of nonlinear rules’ denotes the number of nonlinear rules in a program, ‘degree of nonlinearity’ the number of recursive subgoals in a nonlinear rule, ‘no. of EDB subgoals’ the number of EDB subgoals in a recursive rule, and ‘duplication’ whether the EDB subgoals with the same predicate name is allowed. The lower part explains the results from each study: ‘condition proved’ denotes whether there are conditions that are proved and if they exist, whether they are necessary and sufficient (iff) conditions or sufficient (if) conditions, ‘testing algorithm’ whether there is an algorithm to test the proposed conditions, and ‘time complexity’ the time complexity of the algorithm.

This paper is organized as follows. The following section gives terms and definitions used in this paper. It also explains equivalence of logic programs and conjunctive query containment test. Section 3 defines RLF-linearization. Section 4 identifies sufficient conditions for RLR-linearizability of bilinear programs. First, it presents sufficient conditions for two restricted types of bilinear programs, MB-type and SBSL-type, and then gives a sufficient condition for general bilinear programs. Finally, Section 5 summarizes our results.

2. Preliminaries

2.1. Terminology

In this paper, we deal with function-free Horn clause logic programs, i.e., datalog programs [2,16]. In general, a logic program can be divided into an EDB being a set of facts and an intensional database (IDB) being a set of logical rules. We assume that no common predicate appears in both the EDB and the IDB. Since the EDB of a logic program does not affect whether the program is linear or nonlinear, we refer to the IDB of a program as a program unless otherwise specified.

Normally, a rule in the IDB has the form

$$p_0(\overline{X}_0) \text{ :- } p_1(\overline{X}_1), \ldots, p_n(\overline{X}_n).$$

Here, p_i is a predicate, $\overline{X}_i$ a sequence of variables, and $p_i(\overline{X}_i)$ a *literal*. The left part of ‘:-’, $p_0(\overline{X}_0)$, is the *head* of the rule, and the right part, the conjunction of literals, is the *body* of the rule. The predicate p_0 of the head is the *head predicate*. Each literal of the body is a *subgoal*. An *EDB subgoal* is a subgoal whose predicate belongs to the EDB. An *EDB conjunction* is a conjunction of EDB subgoals in the body. A *range-restricted* [1] rule is a rule in which every variable in the head appears in its body. In this paper, we consider only range-restricted rules.

In a program $\mathcal{P}$, if p is the head predicate of a rule r and q is the predicate of a subgoal in r, then p *directly depends on* q in r (or in $\mathcal{P}$). A predicate p *depends on* a predicate q if p directly depends on q or there is another predicate s such that p directly depends on s and s depends on q. A predicate p is *recursive* if p depends on itself. A subgoal in a rule r is *recursive* if the predicate of the subgoal depends on the head predicate of r. A rule is *recursive* if there is a recursive subgoal in its body. A rule is *non-recursive* otherwise. A rule is *directly recursive* if every recursive subgoal has the same predicate as the head. A program with at least one recursive rule is *recursive*.

A rule is *linear* if it has exactly one recursive subgoal. It is *nonlinear* if it has more than one recursive subgoal. Especially, it is *bilinear* if it has exactly two recursive subgoals. A program is *linear* if all the recursive rules in it are linear. It is *nonlinear* if it has at least one nonlinear rule. It is *bilinear* if it is nonlinear and all the nonlinear rules are bilinear.

A program is a *single-predicate program* if all the head predicates in the program are the same. Note that every recursive rule in such a program is directly recursive. An *exit rule* is a non-recursive rule in a single-predicate recursive program. In order for a single-predicate recursive program to be meaningful, at least one exit rule should exist in it. We assume that there is one and only one exit rule in a single-predicate recursive program.

2.2. *Equivalence of logic programs*

We represent all the facts that can be derived from a program $\mathcal{P}$ together with a database D as $M(\mathcal{P} \cup D)$, where D is not necessarily an EDB, i.e., it can contain facts for IDB predicates. A program $\mathcal{P}$ is *logically contained* into a program $\mathcal{P}'$ denoted as $\mathcal{P} \subseteq \mathcal{P}'$ if $M(\mathcal{P} \cup D) \subseteq M(\mathcal{P}' \cup D)$ for any EDB D. A program $\mathcal{P}$ is *logically equivalent* to a program $\mathcal{P}'$ denoted as $\mathcal{P} \equiv \mathcal{P}'$ if $\mathcal{P} \subseteq \mathcal{P}'$ and $\mathcal{P} \supseteq \mathcal{P}'$. We also use the concept of *uniform equivalence* proposed by Sagiv [10]. It is a more strengthened condition than logical equivalence. A program $\mathcal{P}$ is *uniformly contained* into a program $\mathcal{P}'$ denoted as $\mathcal{P} \subseteq^u \mathcal{P}'$ if $M(\mathcal{P} \cup D) \subseteq M(\mathcal{P}' \cup D)$ for any database D. A program $\mathcal{P}$ is *uniformly equivalent* to a program $\mathcal{P}'$ denoted as $\mathcal{P} \equiv^u \mathcal{P}'$ if $\mathcal{P} \subseteq^u \mathcal{P}'$ and $\mathcal{P} \supseteq^u \mathcal{P}'$. It is obvious that $\mathcal{P} \equiv^u \mathcal{P}'$ implies $\mathcal{P} \equiv \mathcal{P}'$. Note that logical equivalence is undecidable [14], but uniform equivalence is decidable [10].

We introduce the notion of a derivation tree that is used for definitions and proofs in the rest of the paper.

Definition 2.1. Let a program $\mathcal{P}$ and a database D be given. *Derivation trees* from $\mathcal{P} \cup D$ are defined as follows:
1. For each fact in D, there is a derivation tree of a single node that is the fact itself.
2. Consider a ground instance of a rule r in $\mathcal{P}$ given below.

$$p(\overline{c}_0) \text{ :- } q_1(\overline{c}_1), \ldots, q_n(\overline{c}_n).$$

If there are derivation trees T_i $(1 \leqslant i \leqslant n)$ such that the root of each T_i is $q_i(\overline{c}_i)$, then there is a derivation tree with $p(\overline{c}_0)$ as the root and with each T_i as a subtree.

3. All the derivation trees can be defined by the above rules only.

A fact is *derivable* from $\mathcal{P} \cup D$ if it belongs to $M(\mathcal{P} \cup D)$. It is obvious by Definition 2.1 that a fact α is derivable from $\mathcal{P} \cup D$ iff there is a derivation tree with the root α from $\mathcal{P} \cup D$. Therefore, we can prove that $\mathcal{P}$ is logically equivalent to $\mathcal{P}'$ by showing that for any EDB D, there is a derivation tree with the root α from $\mathcal{P} \cup D$ iff there is a derivation tree with the same root α from $\mathcal{P}' \cup D$.

Let $\mathcal{P}$ be a single-predicate program $\mathcal{P}$ for an IDB predicate p, and D an EDB. $M_p(\mathcal{P} \cup D)$ denotes all the p-facts in $M(\mathcal{P} \cup D)$. In this paper, we transform a single-predicate bilinear program $\mathcal{P}$ into a multiple-predicate linear program $\mathcal{P}'$. For linearization, a new IDB predicate is introduced into $\mathcal{P}'$. Generally, $M(\mathcal{P} \cup D)$ is not equal to $M(\mathcal{P}' \cup D)$ since $M(\mathcal{P} \cup D)$ does not have any fact for the newly introduced predicate. But, our interest is whether $M_p(\mathcal{P} \cup D)$ is equal to $M_p(\mathcal{P}' \cup D)$. A single-predicate program $\mathcal{P}$ for an IDB predicate p is *contained to a program $\mathcal{P}'$ with respect to the predicate p*, denoted as $\mathcal{P} \subseteq_p \mathcal{P}'$, if $M_p(\mathcal{P} \cup D) \subseteq M_p(\mathcal{P}' \cup D)$ for any EDB D. $\mathcal{P}$ is *equivalent to $\mathcal{P}'$ with respect to the predicate p*, denoted as $\mathcal{P} \equiv_p \mathcal{P}'$, if $\mathcal{P} \subseteq_p \mathcal{P}'$ and $\mathcal{P} \supseteq_p \mathcal{P}'$.

2.3. Conjunctive query containment test

For any rule, the rule body is a conjunction of literals and the rule head represents a form of answers that can be derived by the body. Therefore, a rule is just a *conjunctive query* [3]. We use the containment test between a conjunctive query Q and a program $\mathcal{P}$ with possibly multiple rules. The following algorithm tests whether Q is uniformly contained into $\mathcal{P}$ [10]:

Algorithm 2.1. *Testing $Q \subseteq^u \mathcal{P}$ for a conjunctive query Q and a program $\mathcal{P}$.*

1. Find a substitution σ such that it replaces each variable in Q with a unique new constant. Substitute each variable X in Q by $\sigma(X)$. We can obtain a fact α by applying σ to the head of Q.
2. Construct a database D such that for each subgoal s of Q, $\sigma(s)$ is in D.
3. Compute $M(\mathcal{P} \cup D)$. If $\alpha \in M(\mathcal{P} \cup D)$, then $Q \subseteq^u \mathcal{P}$. Otherwise, $Q \not\subseteq^u \mathcal{P}$.

Theorem 2.1 [10]**.** *In Algorithm 2.1, $\alpha \in M(\mathcal{P} \cup D)$ iff $Q \subseteq^u \mathcal{P}$.*

Algorithm 2.1 is complete for exponential time in the length of Q and $\mathcal{P}$ [9].

3. Right-linear-first linearization

Let $\mathcal{A}$ be a single-predicate bilinear program shown below. There are one exit rule r_e, m linear rules $r_{a_1}, \ldots, r_{a_m}$, and n bilinear rules $r_{b_1}, \ldots, r_{b_n}$.

$$\begin{array}{lll}
[\mathcal{A}] & r_e & : p(X_1, \ldots, X_t) \text{ :- } e(X_1, \ldots, X_t). \\
 & r_{a_1} & : p(X_1, \ldots, X_t) \text{ :- } p(U_{11}, \ldots, U_{1t}), G_1. \\
 & & \quad \ldots \\
 & r_{a_m} & : p(X_1, \ldots, X_t) \text{ :- } p(U_{m1}, \ldots, U_{mt}), G_m.
\end{array}$$

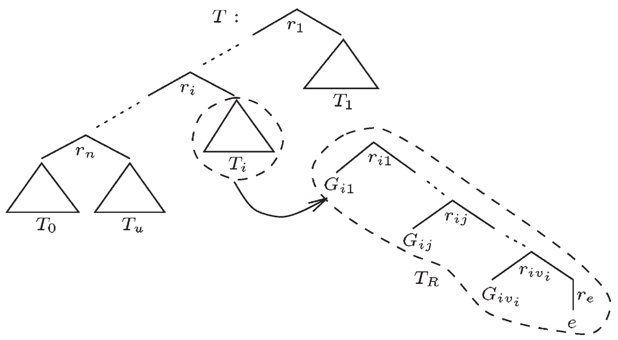

Fig. 1. A derivation tree of a special form derived from the program $\mathcal{A}$.

$$r_{b_1} : p(X_1, \ldots , X_t) \text{ :- } p(Y_{11}, \ldots , Y_{1t}), H_1, p(Z_{11}, \ldots , Z_{1t}).$$
$$\ldots$$
$$r_{b_n} : p(X_1, \ldots , X_t) \text{ :- } p(Y_{n1}, \ldots , Y_{nt}), H_n, p(Z_{n1}, \ldots , Z_{nt}).$$

Here, e is an EDB predicate, and $G_1, \ldots , G_m$, and $H_1, \ldots , H_n$ are conjunctions of EDB subgoals.

Consider a derivation tree derived from $\mathcal{A}$ and an EDB D that has a very special form T given in Fig. 1. For simplicity, EDB subgoals are omitted in T, and each node represents a recursive p subgoal. The followings are the characteristics of T:

- Each rule r_i $(1 \leqslant i \leqslant u)$ applied along the left side of the tree is either a bilinear rule or a linear rule. Let $\mathcal{L}$ be the set of the linear rules in the left side.
- If r_i is a linear rule, obviously the right subtree T_i is empty.
- If r_i is a bilinear rule, the right subtree T_i has a special form shown as T_R, in which the exit rule is applied only once or linear rules $(r_{ij},\ 1 \leqslant j \leqslant v_i)$ are applied repeatedly after an application of the exit rule. Note that each leaf in T_R represents an EDB subgoal. Let $\mathcal{R}$ be the set of the linear rules appearing in those right subtrees.
- T_0 also has the same form as T_R.
- The two sets of linear rules, $\mathcal{L}$ and $\mathcal{R}$, are disjoint.

If all the facts derivable from the trees of the form T_R are regarded as EDB facts, then each bilinear rule in T can be viewed as a linear rule since the right recursive subgoal of any bilinear rule is considered to be an EDB subgoal.

If, for each fact derivable from $\mathcal{A}$ and an arbitrary EDB D, there is a derivation tree of the form T, then $\mathcal{A}$ can be transformed into an equivalent linear program by the following procedure. We call the transformation RLF-linearization.

Right-linear-first linearization

Let $\mathcal{L}$ be the set $\{r_{a_{\pi(1)}}, \ldots , r_{a_{\pi(l)}}\}$ and $\mathcal{R}$ the set $\{r_{a_{\pi(l+1)}}, \ldots , r_{a_{\pi(m)}}\}$, where π is a permutation of the subscripts of the linear rules for representing their partition. For simplicity, we omit the arguments of rules in the procedure.

(1) Transform the exit rule r_e into r'_e, where q is a new predicate name not appearing in $\mathcal{A}$:

$$r_e : \quad p \text{ :- } e. \qquad \Rightarrow \qquad r'_e : \quad q \text{ :- } e.$$

(2) Transform each rule $r_{a_{\pi(h)}}$, $1 \leqslant h \leqslant l$, in $\mathcal{R}$ into $r'_{a_{\pi(h)}}$:

$$r_{a_{\pi(h)}} : \quad p \text{ :- } G_{\pi(h)}, p. \qquad \Rightarrow \qquad r'_{a_{\pi(h)}} : \quad q \text{ :- } G_{\pi(h)}, q.$$

(3) Add a new rule r'_q below:
$r'_q: \quad p \,\text{:-}\, q.$
(4) Transform each bilinear rule r_{b_k}, $1 \leqslant k \leqslant n$, into r'_{b_k}:
$r_{b_k}: \quad p \,\text{:-}\, p, H_k, p. \qquad \Rightarrow \qquad r'_{b_k}: \quad p \,\text{:-}\, p, H_k, q.$
(5) Use each linear rule $r_{a_{\pi(h)}}$, $l+1 \leqslant h \leqslant m$, in $\mathcal{L}$ as itself without modification.

The following program $\mathcal{A}^{rlf}$ is the result of RLF-linearization of the bilinear program $\mathcal{A}$:

$$
\begin{array}{lll}
[\mathcal{A}^{rlf}] & r'_e & : q(X_1, \ldots, X_t) \,\text{:-}\, e(X_1, \ldots, X_t). \\
& r'_{a_{\pi(1)}} & : q(X_1, \ldots, X_t) \,\text{:-}\, q(U_{\pi(1)1}, \ldots, U_{\pi(1)t}), G_{\pi(1)}. \\
& & \quad \ldots \\
& r'_{a_{\pi(l)}} & : q(X_1, \ldots, X_t) \,\text{:-}\, q(U_{\pi(l)1}, \ldots, U_{\pi(l)t}), G_{\pi(l)}. \\
& & \\
& r'_q & : p(X_1, \ldots, X_t) \,\text{:-}\, q(X_1, \ldots, X_t). \\
& r'_{a_{\pi(l+1)}} & : p(X_1, \ldots, X_t) \,\text{:-}\, p(U_{\pi(l+1)1}, \ldots, U_{\pi(l+1)t}), G_{\pi(l+1)}. \\
& & \quad \ldots \\
& r'_{a_{\pi(m)}} & : p(X_1, \ldots, X_t) \,\text{:-}\, p(U_{\pi(m)1}, \ldots, U_{\pi(m)t}), G_{\pi(m)}. \\
& r'_{b_1} & : p(X_1, \ldots, X_t) \,\text{:-}\, p(Y_{11}, \ldots, Y_{1t}), H_1, q(Z_{11}, \ldots, Z_{1t}). \\
& & \quad \ldots \\
& r'_{b_n} & : p(X_1, \ldots, X_t) \,\text{:-}\, p(Y_{n1}, \ldots, Y_{nt}), H_n, q(Z_{n1}, \ldots, Z_{nt}).
\end{array}
$$

The resulting linear program of RLF-linearization from a single-predicate bilinear program defines two IDB predicates. The program $\mathcal{A}^{rlf}$ consists of two single-predicate linear subprograms. One is for the predicate q and has rules r'_e and $r'_{a_{\pi(1)}}, \ldots, r'_{a_{\pi(l)}}$. The other is for the predicate p and has rules r'_q, $r'_{a_{\pi(l+1)}}, \ldots, r'_{a_{\pi(m)}}$, and $r'_{b_1}, \ldots, r'_{b_n}$. In order to compute all the p-facts in a bottom-up fashion, we must first compute the former program to obtain all the q-facts, and then do the latter to obtain all the p-facts. Here, the predicate q plays only a temporary role to accumulate the p-facts generated by the original exit rule and the linear rules in the set $\mathcal{R}$. Since each subprogram consists of only linear rules, $\mathcal{A}^{rlf}$ is a linear program.

Definition 3.1. Let $\mathcal{A}$ be a single-predicate bilinear program for an IDB predicate p. Let $\mathcal{A}^{rlf}$ be the RLF-linearized program of $\mathcal{A}$ under a partition, $\mathcal{L}$ and $\mathcal{R}$, of the linear rules in $\mathcal{A}$. $\mathcal{A}$ is *RLF-linearizable* under the partition if $M_p(\mathcal{A} \cup D) = M_p(\mathcal{A}^{rlf} \cup D)$ for any EDB D.

4. RLF-linearizability for bilinear datalog programs

4.1. MB-type bilinear programs

In this section, we consider RLF-linearizability of an MB-type program that has only bilinear rules as recursive rules. Since an MB-type program has no linear rules, we do not need to consider a partition of linear rules for RLF-linearization. Consider an MB-type program $\mathcal{B}$ and its RLF-linearized program $\mathcal{B}^{mb}$ as shown below. Our interest is whether $\mathcal{B}$ is logically equivalent to $\mathcal{B}^{mb}$.

$$\begin{array}{lll@{\qquad}lll}
[\mathcal{B}] & r_e & : p \text{ :- } e. & [\mathcal{B}^{mb}] & r_e & : p \text{ :- } e. \\
 & r_{b_1} & : p \text{ :- } p, H_1, p. & & r'_{b_1} & : p \text{ :- } p, H_1, e. \\
 & & \ldots & & & \ldots \\
 & r_{b_n} & : p \text{ :- } p, H_n, p. & & r'_{b_m} & : p \text{ :- } p, H_n, e.
\end{array}$$

Definition 4.1. A program $\mathcal{B}$ is *MB-Linearizable* if $\mathcal{B} \equiv \mathcal{B}^{mb}$.

MB-linearizability is a direct extension of ZYT-linearizability for multiple bilinear rules. Ramakrishnan et al. [9] give a very useful observation for ZYT-linearizability that a single bilinear rule program is ZYT-linearizable if for every nonlinear derivation tree with only multiple occurrences of minimal nonlinearity, there is a linear tree for the same fact. Nonlinearity occurs only if a subgoal of a bilinear rule is expanded by a bilinear rule. Minimal nonlinearity implies that such expansion occurs only once so that any recursive subgoal appeared from this expansion is replaced by the exit rule and is no more expanded by a bilinear rule. For MB-linearizability, we can directly adapt the observation of Ramakrishnan et al.'s. Any nonlinear derivation trees generated by the following program $\mathcal{B}^{nl}$ have only multiple occurrences of minimal nonlinearity:

$$\begin{array}{llll}
[\mathcal{B}^{nl}] & r_e & : & p \text{ :- } e. \\
 & r'_{b_1} & : & p \text{ :- } p, H_1, e. \\
 & & & \ldots \\
 & r'_{b_n} & : & p \text{ :- } p, H_n, e. \\
 & r'_{b_{11}} & : & p \text{ :- } p, H_1, \sigma_{11}(e, H_1, e). \\
 & & & \ldots \\
 & r'_{b_{ij}} & : & p \text{ :- } p, H_i, \sigma_{ij}(e, H_j, e). \\
 & & & \ldots \\
 & r'_{b_{nn}} & : & p \text{ :- } p, H_n, \sigma_{nn}(e, H_n, e).
\end{array}$$

Here, $r'_{b_{ij}} (1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n)$ represents minimal nonlinearity that appears from expanding the right recursive subgoal of the rule r_{b_i} by the rule r_{b_j}, and then by replacing the right two recursive predicates with the predicate in the body of the exit rule. The substitution σ_{ij} is for the subgoal expansion to obtain $r'_{b_{ij}}$.

Theorem 4.1. *$\mathcal{B}$ is MB-linearizable iff $\mathcal{B}^{nl} \subseteq \mathcal{B}^{mb}$.*

Proof. See Appendix. □

By the same argument as in [9], since we do not know any way to check the condition of Theorem 4.1, we strengthen the condition using uniform equivalence [10] to obtain a testable condition.

Corollary 4.2. *$\mathcal{B}$ is MB-linearizable if $\mathcal{B}^{nl} \subseteq^u \mathcal{B}^{mb}$.*

The condition of Corollary 4.2 can be tested by showing that each $r'_{b_{ij}}$ of $\mathcal{B}^{nl}$ is uniformly contained into $\mathcal{B}^{mb}$. Since the test for each rule has the complexity of exponential time as discussed in Section 2.3 and there are n^2 rules for such test, we obtain the following theorem:

Theorem 4.3. *The condition of Corollary 4.2 can be tested in exponential time.*

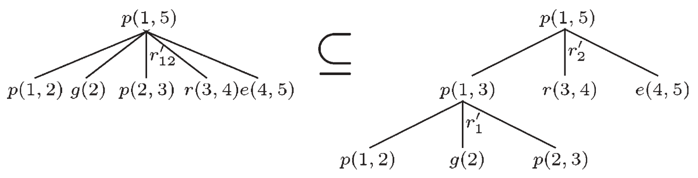

Fig. 2. A containment that shows $r'_{12} \subseteq^u \mathcal{P}1^{mb}$.

Example 4.1. Consider the following MB-type programs $\mathcal{P}1$ and its RLF-linearized program $\mathcal{P}1^{mb}$:

$[\mathcal{P}1]$	r_0 :	$p(X,Y)$:- $e(X,Y)$.	$[\mathcal{P}1^{mb}]$	r_0 :	$p(X,Y)$:- $e(X,Y)$.
	r_1 :	$p(X,Y)$:- $p(X,Z), g(Z), p(Z,Y)$.		r'_1 :	$p(X,Y)$:- $p(X,Z), g(Z), e(Z,Y)$.
	r_2 :	$p(X,Y)$:- $p(X,Y), r(Z,W), p(W,Y)$.		r'_2 :	$p(X,Y)$:- $p(X,Y), r(Z,W), e(W,Y)$.

The program $\mathcal{P}1^{nl}$ that generates nonlinear derivation trees with only multiple occurrences of minimal nonlinearity is as follows:

$[\mathcal{P}1^{nl}]$	r_0 :	$p(X,Y)$:- $e(X,Y)$.
	r'_1 :	$p(X,Y)$:- $p(X,Z), g(Z), e(Z,Y)$.
	r'_2 :	$p(X,Y)$:- $p(X,Y), r(Z,W), e(W,Y)$.
	r'_{11} :	$p(X,Y)$:- $p(X,Z), g(Z), e(Z,U_1), g(U_1), e(U_1,Y)$.
	r'_{12} :	$p(X,Y)$:- $p(X,Z), g(Z), e(Z,U_1), r(U_1,U_2), e(U_2,Y)$.
	r'_{21} :	$p(X,Y)$:- $p(X,Z), r(Z,W), e(W,U_1), g(U_1), e(U_1,Y)$.
	r'_{22} :	$p(X,Y)$:- $p(X,Z), r(Z,W), e(W,U_1), r(U_1,U_2), e(U_2,Y)$.

In order to show that $\mathcal{P}1^{nl}$ is uniformly contained into $\mathcal{P}1^{mb}$, we should test whether each rule r'_{ij} $(1 \leqslant i \leqslant 2, 1 \leqslant j \leqslant 2)$ is uniformly contained into $\mathcal{P}1^{mb}$. We can test these uniform containments using Algorithm 2.1. Here, we show only the case of rule r'_{12}.

For each variable in the body of r_{12}, we assign a unique new constant as follows:

$$X = 1, \quad Z = 2, \quad U_1 = 3, \quad U_2 = 4, \quad Y = 5.$$

Let D be a database, i.e., a set of facts, which is obtained from the body by assigning the constants. Then

$$D = \{p(1,2), g(2), e(2,3), r(3,4), e(4,5)\}.$$

Fig. 2 illustrates the remaining procedure. By applying r'_{12} to the data base D, we get a fact $p(1,5)$. It is shown by the left tree in Fig. 2. Now, we must prove that $\mathcal{P}1^{mb} \cup D$ also produces the fact $p(1,5)$. The right tree shows that the same fact $p(1,5)$ can also be produced from $\mathcal{P}1^{mb} \cup D$.

The above procedure shows that r'_{12} is uniformly contained into $\mathcal{P}1^{mb}$. By the same test procedure, we can show that the other three rules are also uniformly contained into $\mathcal{P}1^{mb}$. This implies $\mathcal{P}1^{nl} \subseteq^u \mathcal{P}1^{mb}$. By Corollary 4.2, $\mathcal{P}1 \equiv \mathcal{P}1^{mb}$.

4.2. SBSL-type bilinear programs

Consider an SBSL-type bilinear program $\mathcal{C}$ having one bilinear rule and one linear rule. It is of the form $\mathcal{C}$ that is obtained by adding a linear rule r_a into $\mathcal{Z}$.

$[\mathcal{C}]$ $r_e : p$:- e.
$r_a : p$:- p, G.
$r_b : p$:- p, H, p.

In order to RLF-linearize the program $\mathcal{C}$, we need a partition of linear rules in $\mathcal{C}$. Since $\mathcal{C}$ has only one linear rule, there are only two possible partitions: one is $\mathcal{L} = \{r_a\}$ and $\mathcal{R} = \{\}$, and the other is $\mathcal{L} = \{\}$ and $\mathcal{R} = \{r_a\}$. The linear rule r_a makes some trouble against linearizability. The right recursive subgoal of the bilinear rule can be expanded by the linear rule. The result of such expansion introduces a new type of nonlinearity, which we characterize for each partition. The SBSLU-linearizability corresponds to the former partition, and the SBSLD-linearizability to the latter.

4.2.1. SBSLU-linearizability

The new type of nonlinearity by a linear rule is depicted by a derivation tree T_u shown in Fig. 3. The superscripts α and β of the recursive subgoal p in T_u are for distinguishing the position of each subgoal. The corresponding rule expansion is

r_{ba}: p :- $p^\alpha, H, p^\beta, \sigma(G)$.

σ denotes a substitution for this expansion. (In fact, p^β denotes the recursive subgoal in the substituted rule $\sigma(r_a)$.) T'_u in Fig. 3 represents a possible transformed linear tree whose root is identical to that of T_u. All the leaves in T'_u exist in T_u. Hence, it is trivial that T_u is uniformly contained into T'_u. This containment is sufficient to eliminate the new type of nonlinearity. Note that while the application of the linear rule occurs at the lower part in T_u, it occurs at the relatively upper part in T'_u.

Fig. 4 shows more general containment of T_u into T''_u that is sufficient to eliminate the new nonlinearity. For each node in the left side of T''_u, either the linear rule r_a or the bilinear rule r_b is applied. When r_a is applied, e.g., $r_i = r_a$, the corresponding child G^i is an instance of the EDB conjunction G in r_a. When r_b is applied, e.g., $r_j = r_b$, the corresponding child H^j is an instance of H in r_b, and the right recursive subgoal is p^β. The leftmost leaf is either p^α or p^β. If T_u is contained into T''_u, it is sufficient to obtain linearity from the new nonlinearity. Note that in T_u, the application of the linear rule r_a occurs at the level lower than that of the leftmost leaf of T_u, but in T''_u, the application of r_a occurs at the upper

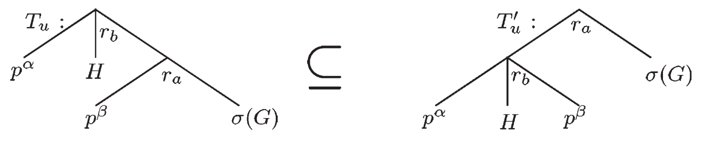

Fig. 3. Nonlinearity by a linear rule and a sufficient containment to eliminate such nonlinearity.

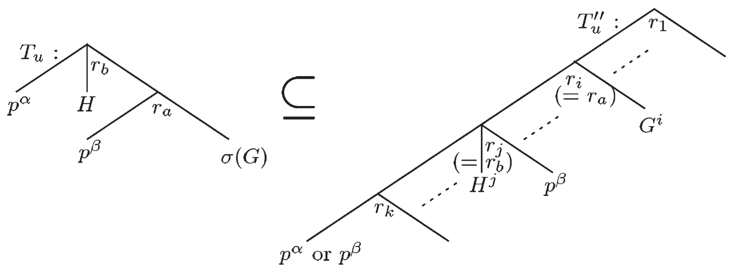

Fig. 4. A containment more general than the one in Fig. 3.

part of the leftmost leaf of T''_u. Hence, the application of the linear rule can be thought to be moved *up* in this transformation. (So, the letter *"U"* in the acronym *SBSLU* is to denote *"up"*.)

Note that Corollary 9 in [9] gives a condition for eliminating nonlinearity by the expansion of the right recursive subgoal of a bilinear rule by the rule itself. That is, the condition implies a transformation for eliminating such nonlinearity. Therefore, when this transformation together with the transformation of T_u into T''_u for eliminating the new nonlinearity is applied finite times to a derivation tree from a program $\mathcal{C}$ and an EDB D, we finally obtain a linear tree that can be generated from the following RLF-linearized program $\mathcal{C}^{slu}$ and the same EDB D:

$[\mathcal{C}^{slu}]$ $r_e : p$:- e.
$r_a : p$:- p, G.
$r'_b : p$:- p, H, e.

Definition 4.2. A program $\mathcal{C}$ is *SBSLU-linearizable* if $\mathcal{C} \equiv \mathcal{C}^{slu}$.

We can easily identify that the RLF-linearization for obtaining $\mathcal{C}^{slu}$ is based on the partition $\mathcal{L} = \{r_a\}$ and $\mathcal{R} = \{\}$. Therefore, SBSLU-linearizability is a special case of RLF-linearizability such that $\mathcal{R} = \{\}$ is given as a linear rule partition.

The following program $\mathcal{C}^{nl}$ generates only the derivation trees with multiple occurrences of the two types of minimal nonlinearity. They are represented by the rules r'_{ba} and r'_{bb}. σ_{ba} (respectively σ_{bb}) denotes the substitution for the expansion of r_b by r_a (respectively by r_b itself).

$[\mathcal{C}^{nl}]$ r_e : p :- e.
r_a : p :- p, G.
r'_b : p :- p, H, e.
r'_{ba} : p :- $p, H, \sigma_{ba}(e, G)$.
r'_{bb} : p :- $p, H, \sigma_{bb}(e, H, e)$.

Theorem 4.4. *$\mathcal{C}$ is SBSLU-linearizable iff $\mathcal{C}^{nl} \subseteq \mathcal{C}^{slu}$.*

Proof. See Appendix. □

Corollary 4.5. *$\mathcal{C}$ is SBSLU-linearizable if $\mathcal{C}^{nl} \subseteq^u \mathcal{C}^{slu}$.*

The condition of Corollary 4.5 can be tested by verifying whether each of the rules r'_{ba} and r'_{bb} is uniformly contained into the program $\mathcal{C}^{slu}$. Since each test for the rules requires exponential time, we have the following theorem:

Theorem 4.6. *The condition of Corollary 4.5 can be tested in exponential time.*

Example 4.2. Consider the following SBSL-type programs $\mathcal{P}2$ and its RLF-linearized program $\mathcal{P}2^{slu}$ for testing SBSLU-linerizability:

$[\mathcal{P}2]$ r_0 : $p(X, Y)$:- $e(X, Y)$.
r_1 : $p(X, Y)$:- $p(X, Z), g(Z, Y)$.
r_2 : $p(X, Y)$:- $p(X, Z), r(Z, W), p(W, Y)$.

$[\mathcal{P}2^{slu}]$ r_0 : $p(X, Y)$:- $e(X, Y)$.
r_1 : $p(X, Y)$:- $p(X, Z), g(Z, Y)$.
r'_2 : $p(X, Y)$:- $p(X, Y), r(Z, W), e(W, Y)$.

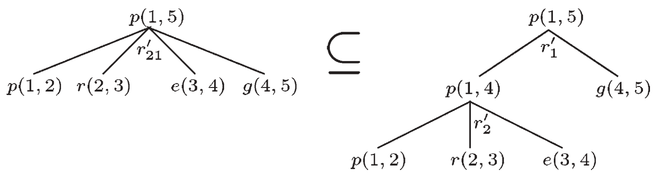

Fig. 5. A containment that shows $r'_{21} \subseteq^u \mathcal{P}2^{slu}$.

The program $\mathcal{P}2^{nl}$ that generates all the derivation trees with multiple occurrences of minimal non-linearity is as follows:

$[\mathcal{P}2^{nl}]$	r_0	:	$p(X, Y)$:- $e(X, Y)$.
	r_1	:	$p(X, Y)$:- $p(X, Z), g(Z, Y)$.
	r'_2	:	$p(X, Y)$:- $p(X, Z), r(Z, W), e(W, Y)$.
	r'_{21}	:	$p(X, Y)$:- $p(X, Z), r(Z, W), e(W, U_1), g(U_1, Y)$.
	r'_{22}	:	$p(X, Y)$:- $p(X, Z), r(Z, W), e(W, U_1), r(U_1, U_2), e(U_2, Y)$.

We can prove that $\mathcal{P}2^{nl}$ is uniformly contained into $\mathcal{P}2^{slu}$ by showing that both r'_{21} and r'_{22} are uniformly contained into $\mathcal{P}2^{slu}$. Algorithm 2.1 is applied as in the case of Example 4.1. Fig. 5 illustrates that r'_{21} is uniformly contained into $\mathcal{P}2^{slu}$. The left tree is constructed using the rule r'_{21} of $\mathcal{P}2^{nl}$ by instantiating each variable in the rule body to a unique constant, so that the set of facts, $D = \{p(1, 2), r(2, 3), e(3, 4), g(4, 5)\}$, is used to produce the fact $p(1, 5)$. The right tree shows that the same fact $p(1, 5)$ can be produced from $\mathcal{P}2^{slu} \cup D$.

Similarly, the rule r'_{22} is also uniformly contained into $\mathcal{P}2^{slu}$. Therefore, $\mathcal{P}2^{nl} \subseteq^u \mathcal{P}2^{slu}$, which implies $\mathcal{P}2 \equiv \mathcal{P}2^{slu}$ by Corollary 4.5.

4.2.2. *SBSLD-linearizability*

We can also think about the possibility that all the applications of the linear rule are moved *down*. (The letter "*D*" in *SBSLD* is to denote "*down*".) If such transformation can be applied to a derivation tree T_d from an SBSL-type bilinear program $\mathcal{C}$ and an EDB D, we can obtain a stratified derivation tree T'_d, as shown in Fig. 6, such that all the applications of the bilinear rule are in the upper part (labeled as p) and all the applications of the linear rule are in the lower part (labeled as q).

The following program $\mathcal{C}^{ud}$ generates only the derivation trees of the form T'_d. It consists of two subprograms. One is for the predicate q corresponding to the lower part of a derivation tree from $\mathcal{C}^{ud}$. The other is for the predicate p corresponding to the upper part of the tree. Let $\mathcal{C}^{ud}_p$ be the latter subprogram.

$[\mathcal{C}^{ud}]$	$r'_e : q$:- e.		
	$r'_a : q$:- q, G.		
	$r_q : p$:- q.	$[\mathcal{C}^{ud}_p]$	$r_q : p$:- q.
	$r_b : p$:- p, H, p.		$r_b : p$:- p, H, p.

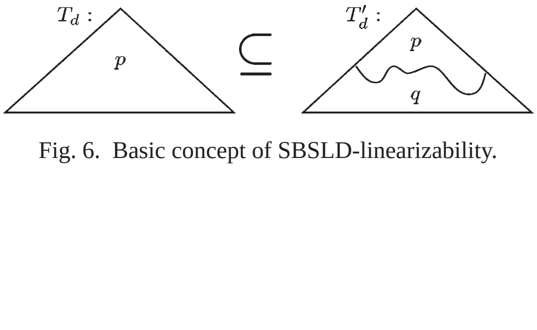

Fig. 6. Basic concept of SBSLD-linearizability.

If $\mathcal{C}^{ud}$ is equivalent to $\mathcal{C}$ with respect to the predicate p (i.e., $M_p(\mathcal{C}^{ud} \cup D) = M_p(\mathcal{C} \cup D)$ for any EDB D) and $\mathcal{C}^{ud}_p$ is ZYT-linearizable, we can see that $\mathcal{C}$ is linearizable, i.e., it is equivalent to the following linear program $\mathcal{C}^{sld}$ with respect to p.

$$
\begin{array}{lllll}
[\mathcal{C}^{sld}] & r'_e : q \text{ :- } e. & & & \\
 & r'_a : q \text{ :- } q, G. & & & \\
 & r'_q : p \text{ :- } q. & \qquad & [\mathcal{C}^{sld}_p] & r'_q : p \text{ :- } q. \\
 & r'_b : p \text{ :- } p, H, q. & & & r'_b : p \text{ :- } p, H, q.
\end{array}
$$

$\mathcal{C}^{sld}_p$ denotes the subprogram for the predicate p of $\mathcal{C}^{sld}$.

Definition 4.3. A program $\mathcal{C}$ is *SBSLD-linearizable* if $\mathcal{C} \equiv_p \mathcal{C}^{sld}$.

The RLF-linearization for obtaining $\mathcal{C}^{sld}$ is based on the partition $\mathcal{L} = \{\}$ and $\mathcal{R} = \{r_a\}$. As in the case of SBSLU-linearizability, the partition implies that SBSLD-linearizability is also a special case of RLF-linearizability.

Theorem 4.7. *A program $\mathcal{C}$ is SBSLD-linearizable iff $\mathcal{C} \equiv_p \mathcal{C}^{ud}$ and $\mathcal{C}^{ud}_p$ is ZYT-linearizable.*

Proof. See Appendix. □

Let $\mathcal{C}^{nl}_p$ denote the following program:

$$
\begin{array}{ll}
[\mathcal{C}^{nl}_p] & r'_q \ : p \text{ :- } q. \\
 & r'_b \ : p \text{ :- } p, H, q. \\
 & r'_{bb} : p \text{ :- } p, H, \sigma_{bb}(q, H, q).
\end{array}
$$

$\mathcal{C}^{nl}_p$ generates only the derivation trees with multiple occurrences of minimal nonlinearity among all the nonlinear trees from $\mathcal{C}^{ud}_p$. It is obvious by the reference [9] that $\mathcal{C}^{ud}_p$ is ZYT-linearizable if $\mathcal{C}^{sld}_p \equiv^u \mathcal{C}^{nl}_p$.

Now, we find how to check the condition $\mathcal{C} \equiv_p \mathcal{C}^{ud}$. Since $\mathcal{C} \supseteq_p \mathcal{C}^{ud}$ trivially holds, we want to know whether $\mathcal{C} \subseteq_p \mathcal{C}^{ud}$ holds. Consider the following two rules:

$$
r_a : p \text{ :- } p, G. \qquad\qquad r_s : p \text{ :- } p, H, s.
$$

The rule r_s is obtained from the rule r_b by replacing the predicate of the right recursive subgoal with a new predicate s. If s is treated as an EDB predicate, r_s can be considered as a linear rule. Then, both r_a and r_s are linear. If r_s commutes with r_a, we can move down all the applications of r_a in a derivation tree of the program $\mathcal{C}$ into the lower part and we obtain a new derivation tree with the same root where all the applications of r_a appear below the applications of r_s, i.e., r_b. Therefore, the equivalence of $\mathcal{C}$ and $\mathcal{C}^{ud}$ can be thought as a problem of commutativity [9] of two linear rules. This approach makes it possible to obtain a sufficient condition for SBSLD-linearizability.

Before we give the condition, it is necessary to explain *regular expressions for linear rules*. From the definition of derivation trees, every leaf of a derivation tree should be in the database. If we are interested mainly in the rule applications than in the derived facts themselves, it is useful to consider derivation trees whose leaves are not in the database. We will call such derivation trees as a *partial derivation trees*. By the observation of Ramakrishnan et al. [9], it is possible to represent a set of partial derivation trees for linear programs by regular expressions over the rule names. For instance, $r_a \cdot r_s$ represents all the trees such that r_a is applied immediately after r_s is applied. In other words, the recursive subgoal of r_a is expanded by r_s in such a tree. The power form r^m represents all the trees such that only r is applied m times consecutively. The closure form r^* represents all the trees such that only r is applied zero or more times consecutively. Therefore, a complex expression $r_s \cdot r_a^*$ represents all the trees such that r_s is applied after r_a is applied zero or more times. Ramakrishnan et al. [9] show that the two linear rules r_a and r_s are commutative if $r_a \cdot r_s \subseteq r_s \cdot r_a^*$. They also explain how such a subset relationship can be tested using uniform containment.

Now we give our condition for SBSLD-linearizability.

Lemma 4.8. *$\mathcal{C} \equiv_p \mathcal{C}^{ud}$ if $r_a \cdot r_s \subseteq r_s \cdot r_a^*$.*

Proof. See Appendix. □

Corollary 4.9. *A program $\mathcal{C}$ is SBSLD-linearizable if $r_a \cdot r_s \subseteq r_s \cdot r_a^*$ and $\mathcal{C}_p^{nl} \subseteq^u \mathcal{C}_p^{sld}$.*

Since each subcondition of Corollary 4.9 can be tested in exponential time, we obtain Theorem 4.10.

Theorem 4.10. *The condition of Corollary 4.9 can be tested in exponential time.*

Example 4.3. Consider the following SBSL-type program $\mathcal{P}3$.

$$\begin{array}{lll} [\mathcal{P}3] & r_e: & p(X,Y) \text{ :- } e(X,Y). \\ & r_a: & p(X,Y) \text{ :- } i(X,Z), p(Z,Y). \\ & r_b: & p(X,Y) \text{ :- } p(X,Z), g(Z), p(Z,Y). \end{array}$$

We want to test whether $\mathcal{P}3$ is SBSLD-linearizable, i.e., whether $\mathcal{P}3$ is equivalent to its RLF-linearized program $\mathcal{P}3^{sld}$.

$$\begin{array}{lll} [\mathcal{P}3^{sld}] & r'_e: & q(X,Y) \text{ :- } e(X,Y). \\ & r'_a: & q(X,Y) \text{ :- } i(X,Z), q(Z,Y). \\ & r_q: & p(X,Y) \text{ :- } q(X,Y). \\ & r'_b: & p(X,Y) \text{ :- } p(X,Z), g(Z), q(Z,Y). \end{array} \qquad \begin{array}{lll} [\mathcal{P}3_p^{sld}] & r_q: & q(X,Y) \text{ :- } e(X,Y). \\ & r'_b: & p(X,Y) \text{ :- } p(X,Z), g(Z), q(Z,Y). \end{array}$$

Consider the following program $\mathcal{P}3^{ud}$.

$$\begin{array}{lll} [\mathcal{P}3^{ud}] & r'_e: & q(X,Y) \text{ :- } e(X,Y). \\ & r'_a: & q(X,Y) \text{ :- } i(X,Z), q(Z,Y). \\ & r_q: & p(X,Y) \text{ :- } q(X,Y). \\ & r_b: & p(X,Y) \text{ :- } p(X,Z), g(Z), p(Z,Y). \end{array} \qquad \begin{array}{lll} [\mathcal{P}3_p^{ud}] & r_q: & p(X,Y) \text{ :- } q(X,Y). \\ & r_b: & p(X,Y) \text{ :- } p(X,Z), g(Z), p(Z,Y). \end{array}$$

$\mathcal{P}3_p^{ud}$ denotes the subprogram for the predicate p of $\mathcal{P}3^{ud}$. Theorem 4.7 says that $\mathcal{P}3$ is SBSLD-linearizable if $\mathcal{P}3 \equiv_p \mathcal{P}3^{ud}$ and $\mathcal{P}3_p^{ud}$ is ZYT-linearizable.

First, we should show that $\mathcal{P}3 \equiv_p \mathcal{P}3^{ud}$. This can be accomplished if the following two linear rules satisfy the condition $r_a \cdot r_s \subseteq r_s \cdot r_a^*$ of Lemma 4.8:

$$\begin{array}{ll} r_a: & p(X,Y) \text{ :- } i(X,Z), p(Z,Y). \\ r_s: & p(X,Y) \text{ :- } p(X,Z), g(Z), s(Z,Y). \end{array}$$

Both rules are from $\mathcal{P}3$, but r_s is obtained from the bilinear rule r_b by replacing the right recursive subgoal p with a new predicate s. We use the procedure given by the reference [9] as follows. The basic idea is to test containment between a rule $r_a \cdot r_s$ and a program $r_s \cdot r_a^*$.

The rule corresponding to $r_a \cdot r_s$ is

$$[r_a \cdot r_s] \quad r'_{as}: \; p(X,Y) \text{ :- } i(X,Z), p'(Z,W), g(W), s(W,Y).$$

The program corresponding to $r_s \cdot r_a^*$ is

$$\begin{array}{lll} [r_s \cdot r_a^*] & r'_s : & p(X,Y) \text{ :- } p'(X,Z), g(Z), s(Z,Y). \\ & r'_a : & p'(X,Y) \text{ :- } i(X,Z), p'(Z,Y). \end{array}$$

Fig. 7 illustrates that $r_a \cdot r_s \subseteq r_s \cdot r_a^*$ holds. Therefore, $\mathcal{P}3 \equiv_p \mathcal{P}3^{ud}$ holds.

Second, we must show that $\mathcal{P}3_p^{ud}$ is ZYT-linearizable. It is a special case of MB-linearizability, so that the same procedure as in Example 4.1 can be applied.

This can be done by showing that the following program $\mathcal{P}3_p^{nl}$ is uniformly contained into $\mathcal{P}3_p^{sld}$:

$$\begin{array}{lll} [\mathcal{P}3_p^{nl}] & r_q : & p(X,Y) \text{ :- } q(X,Y). \\ & r'_b : & p(X,Y) \text{ :- } p(X,Z), g(Z), q(Z,Y). \\ & r'_{bb} : & p(X,Y) \text{ :- } p(X,Z), g(Z), q(Z,U), g(U), q(U,Y). \end{array}$$

The program $\mathcal{P}3_p^{nl}$ generates only the derivation trees with minimal nonlinearity among all the nonlinear trees from $\mathcal{P}3_p^{ud}$. Fig. 8 illustrates that $r_{bb} \subseteq^u \mathcal{P}3_p^{sld}$. This fact implies $\mathcal{P}3_p^{nl} \subseteq^u \mathcal{P}3_p^{sld}$. Therefore, $\mathcal{P}3_p^{ud}$ is ZYT-linearizable.

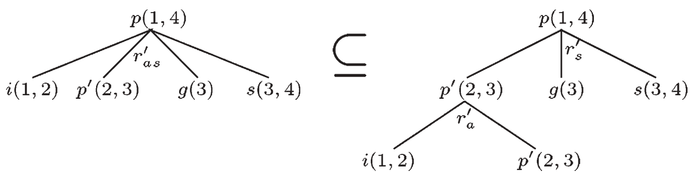

Fig. 7. A containment that shows $r_a \cdot r_s \subseteq r_s \cdot r_a^*$.

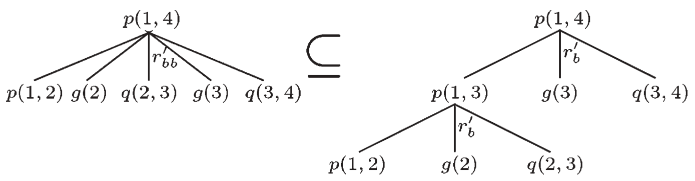

Fig. 8. A containment that shows $r_{bb} \subseteq^u \mathcal{P}3_p^{sld}$.

4.3. RLF-linearizability of general bilinear programs

Consider a derivation tree T from a bilinear program $\mathcal{A}$ and an EDB D. Let $\mathcal{A}^{rlf}$ be the RLF-linearized program of $\mathcal{A}$ under a given partition $\mathcal{L}$ and $\mathcal{R}$ of the linear rules. In order to obtain a derivation tree T' from $\mathcal{A}^{rlf} \cup D$ with the same root as T, we can consider two aspects: one is moving down all the applications of the linear rules in the group $\mathcal{R}$ (as in the case of SBSLD-linearizability), and the other is elimination of nonlinearity by the bilinear rules and the linear rules in the group $\mathcal{L}$ (as in the case of MB- and SBSLU-linearizability). The following conditions are sufficient to move down all the the applications of the linear rules in the group $\mathcal{R}$:

- Any linear rule $r_{a_{\pi(j)}}$ $(l+1 \leqslant j \leqslant m)$ in the group $\mathcal{R}$ commutes with any bilinear rule r_{b_k} $(1 \leqslant k \leqslant n)$ just like as in the case of SBSLD-linearizability. That is, $r_{a_{\pi(j)}} \cdot r_{s_k} \subseteq r_{s_k} \cdot r^*_{a_{\pi(j)}}$ where r_{s_k} is a modified rule of r_{b_k} obtained by replacing the predicate of the left recursive subgoal of r_{b_k} with a new predicate s_k.

  $r_{a_{\pi(j)}} : p$:- $p, G_{\pi(j)}$. $\qquad r_{s_k} : p$:- s_k, H_k, p.
- Any linear rule $r_{a_{\pi(j)}}$ $(l+1 \leqslant j \leqslant m)$ in the group $\mathcal{R}$ commutes with any linear rule $r_{a_{\pi(i)}}$ $(1 \leqslant i \leqslant l)$ in the group $\mathcal{L}$. That is, $r_{a_{\pi(j)}} \cdot r_{a_{\pi(i)}} \subseteq r_{a_{\pi(i)}} \cdot r^*_{a_{\pi(j)}}$.

If the above two conditions hold, any applications of the linear rules in $\mathcal{R}$ can be moved down in any derivation tree from $\mathcal{A} \cup D$. The following program $\mathcal{A}^{ud}$ generates the derivation trees in which the linear rules in $\mathcal{R}$ are applied first and then both the linear rules in $\mathcal{L}$ and the bilinear rules are applied. Any derivation tree obtained by moving down all the applications of the linear rules in $\mathcal{R}$ can be generated by $\mathcal{A}^{ud}$.

```
[$\mathcal{A}^{ud}$]  $r'_e$            : $q$ :- $e$.
        $r'_{a_{\pi(1)}}$       : $q$ :- $q, G_{\pi(1)}$.
                    ...
        $r'_{a_{\pi(l)}}$       : $q$ :- $q, G_{\pi(l)}$.
        $r'_q$            : $p$ :- $q$.
        $r_{a_{\pi(l+1)}}$      : $p$ :- $p, G_{\pi(l+1)}$.
                    ...
        $r_{a_{\pi(m)}}$       : $p$ :- $p, G_{\pi(m)}$.
        $r_{b_1}$           : $p$ :- $p, H_1, p$.
                    ...
        $r_{b_n}$           : $p$ :- $p, H_n, p$.
```

Lemma 4.11. *$\mathcal{A} \equiv_p \mathcal{A}^{ud}$ if $r_{a_{\pi(j)}} \cdot r_{s_k} \subseteq r_{s_k} \cdot r^*_{a_{\pi(j)}}$ for any j $(l+1 \leqslant j \leqslant m)$ and k $(1 \leqslant k \leqslant n)$, and $r_{a_{\pi(j)}} \cdot r_{a_{\pi(i)}} \subseteq r_{a_{\pi(i)}} \cdot r^*_{a_{\pi(j)}}$ for any j $(l+1 \leqslant j \leqslant m)$ and any i $(1 \leqslant i \leqslant l)$.*

After this transformation, we must further proceed to eliminate nonlinearity. Let $\mathcal{A}^{ud}_p$ denote the program as below, consisting of the rules with the head predicate p among the rules in $\mathcal{A}^{ud}$.

```
[$\mathcal{A}^{ud}_p$]  $r'_q$            : $p$ :- $q$.
        $r_{a_{\pi(l+1)}}$      : $p$ :- $p, G_{\pi(l+1)}$.
                    ...
        $r_{a_{\pi(m)}}$       : $p$ :- $p, G_{\pi(m)}$.
        $r_{b_1}$           : $p$ :- $p, H_1, p$.
                    ...
        $r_{b_n}$           : $p$ :- $p, H_n, p$.
```

Two types of nonlinearity can appear in the derivation trees from $\mathcal{A}_p^{ud} \cup D$.

- Any pair of bilinear rules generates a type of nonlinearity. There are n^2 kinds of such nonlinearity.
- Any pair of a bilinear rule and a linear rule in $\mathcal{L}$ also generates another type of nonlinearity. There are nl kinds of such nonlinearity.

Therefore, there are total $n^2 + nl$ kinds of nonlinearity. As we see in MB- and SBSLU-linearizability, if we eliminate all the occurrences of minimal nonlinearity, the given program $\mathcal{A}$ is RLF-linearizable. The following program $\mathcal{A}_p^{nl}$ generates only the derivation trees with multiple occurrences of each minimal nonlinearity:

$$
\begin{array}{lll}
[\mathcal{A}_p^{nl}] & r'_q & : p \text{ :- } q. \\
 & r_{a_{\pi(l+1)}} & : p \text{ :- } p, G_{\pi(l+1)}. \\
 & & \quad\ldots \\
 & r_{a_{\pi(m)}} & : p \text{ :- } p, G_{\pi(m)}. \\
 & r_{b_1} & : p \text{ :- } p, H_1, p. \\
 & & \quad\ldots \\
 & r_{b_n} & : p \text{ :- } p, H_n, p. \\
 & r'_{b_{11}} & : p \text{ :- } p, H_1, \sigma_{11}(e, H_1, e). \\
 & & \quad\ldots \\
 & r'_{b_{ij}} & : p \text{ :- } p, H_i, \sigma_{ij}(e, H_j, e). \\
 & & \quad\ldots \\
 & r'_{b_{nn}} & : p \text{ :- } p, H_n, \sigma_{nn}(e, H_n, e). \\
 & r'_{b_1 a_{\pi(l+1)}} & : p \text{ :- } p, H_1, \tau_{1\pi(l+1)}(e, G_{\pi(l+1)}). \\
 & & \quad\ldots \\
 & r'_{b_i a_{\pi(j)}} & : p \text{ :- } p, H_i, \tau_{i\pi(j)}(e, G_{\pi(j)}). \\
 & & \quad\ldots \\
 & r'_{b_n a_{\pi(m)}} & : p \text{ :- } p, H_n, \tau_{n\pi(m)}(e, G_{\pi(m)}).
\end{array}
$$

Here, $r'_{b_{ij}}$ $(1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant n)$ denotes the rule that is obtained by expanding the right recursive subgoal of the rule r_{b_i} by the rule r_{b_j}, and then by substituting the right two recursive predicates into the EDB predicate in the exit rule. σ_{ij} is a substitution corresponding to this expansion. $r'_{b_i a_{\pi(j)}}$ $(1 \leqslant i \leqslant n,\ l+1 \leqslant j \leqslant m)$ also denotes the rule obtained by expanding r_{b_i} by $r_{a_{\pi(j)}}$ and then substituting the right recursive predicate. $\tau_{i\pi(j)}$ is a corresponding substitution. Let $\mathcal{A}_p^{rlf}$ be the following program, consisting of the rules with the head predicate p among the rules in $\mathcal{A}^{rlf}$:

$$
\begin{array}{lll}
[\mathcal{A}_p^{rlf}] & r'_q & : p \text{ :- } q. \\
 & r'_{a_{\pi(l+1)}} & : p \text{ :- } p, G_{\pi(l+1)}. \\
 & & \quad\ldots \\
 & r'_{a_{\pi(m)}} & : p \text{ :- } p, G_{\pi(m)}. \\
 & r'_{b_1} & : p \text{ :- } p, H_1, q. \\
 & & \quad\ldots \\
 & r'_{b_n} & : p \text{ :- } p, H_n, q.
\end{array}
$$

Lemma 4.12. $\mathcal{A}^{ud} \equiv_p \mathcal{A}^{rlf}$ *iff* $\mathcal{A}_p^{nl} \subseteq \mathcal{A}_p^{rlf}$.

We obtain the following result from Lemma 4.11 and 4.12:

Theorem 4.13. *$\mathcal{A}$ is RLF-linearizable under a given partition, $\mathcal{L}$ and $\mathcal{R}$, of the linear rules in $\mathcal{A}$ if the conditions of Lemma 4.11 hold and $\mathcal{A}_p^{nl} \subseteq^u \mathcal{A}_p^{rlf}$.*

There are $(m - l) \cdot (n + l)$ cases for testing the commutativity conditions in Lemma 4.11. Each test is exponential in time. There are $n^2 + nl$ rules in $\mathcal{A}^{nl}$ corresponding to minimal nonlinearity. In order to test whether $\mathcal{A}_p^{nl} \subseteq^u \mathcal{A}_p^{rlf}$ holds, we must show that each of $n^2 + nl$ rules is uniformly contained into $\mathcal{A}_p^{rlf}$. Each test is also exponential. Therefore, we conclude the following theorem:

Theorem 4.14. *There is a sufficient condition to be tested in exponential time whether a bilinear program $\mathcal{A}$ is RLF-linearizable under a given partition of the linear rules in $\mathcal{A}$.*

5. Conclusions

Linearization of nonlinear recursive programs is very useful in deductive databases. It allows for the use of well-known cost-effective techniques for the evaluation of linear recursions. Unfortunately, the general problem of whether a bilinear program is equivalent to a linear program is undecidable, if $\mathcal{P} \neq \mathcal{NP}$ [5]. There is a well-known linearization, ZYT-linearization, for a limited class of bilinear programs. A bilinear program of the limited class consists of only one bilinear rule and one exit rule.

We have proposed a new transformation method, called *right-linear-first (RLF) linearization,* to linearize general bilinear datalog programs that have multiple bilinear and linear rules. In RLF-linearization, we first partition the set of linear rules in a bilinear program into two disjoint subsets. Note that bilinear programs having more than one linear rule have many partitions. Based on a partition, RLF-linearization transforms a bilinear program to a linear program that has two stratified linear recursions. If a bilinear program is equivalent to its RLF-linearized one, the program is said to be *RLF-linearizable*.

We have found sufficient conditions for RLF-linearizability of the two restricted types of bilinear programs, called *MB-type* and *SBSL-type*. An MB-type bilinear program has only bilinear rules. A SBSL-type program has exactly two recursive rules: one is linear and the other bilinear. Using the results on these two types, we have derived a testable sufficient condition for RLF-linearizability of general bilinear programs. This sufficient condition can be tested in exponential time.

Note that our results are obtained under the assumption that a partition of the linear rules is given when RLF-linearization is applied to a bilinear program. We can consider all possible partitions for RLF-linearizability. There are 2^m partitions for a bilinear program with m linear rules. Although we consider all the partitions, the time complexity of testing RLF-linearizability can still be exponential.

Acknowledgment

This work was partially supported by the Korea Science and Engineering Foundation (KOSEF) through the Software Research Center (SOREC) at Chungnam National University, and through the Advanced Information Technology Reseach Center (AITrc) at KAIST.

Appendix

Proof of Theorem 4.1

It is obvious that $\mathcal{B} \supseteq \mathcal{B}^{nl} \supseteq \mathcal{B}^{mb}$. Assume that $\mathcal{B}$ is MB-linearizable, i.e., $\mathcal{B} \equiv \mathcal{B}^{mb}$. It implies $\mathcal{B}^{nl} \equiv \mathcal{B}^{mb}$. Therefore $\mathcal{B}^{nl} \subseteq \mathcal{B}^{mb}$.

For the sufficiency, we prove by induction on the number of internal nodes of a derivation tree that for every derivation tree from $\mathcal{B}$ together with any EDB D, there is a derivation tree with the same root from $\mathcal{B}^{mb} \cup D$. The basis is trivial because a tree with only one internal node is generated only by the exit rule. Assume that the induction hypothesis holds for trees with less than k internal nodes. Let T of Fig. 9 be a derivation tree with k internal nodes from $\mathcal{B} \cup D$. We can find an internal node $p(\overline{c}_0)$ whose children correspond to the internal nodes derived by the exit rule, as shown in T. Let T_c be this subtree with the root $p(\overline{c}_0)$. Now, we change the tree T to T' by replacing T_c with T'_c as in Fig. 9. Let $D' = D \cup \{e(\overline{c}_0)\}$. Then, T' is a derivation tree with $k - 2$ internal nodes from $\mathcal{B} \cup D'$. By the induction hypothesis, there is a left-linear derivation tree T'' with the same root as T' as shown in Fig. 9. T'' uses zero or more $e(\overline{c}_0)$'s for leaves. Note that $p(\overline{c}_0)$ is derivable from $\mathcal{B} \cup D$, whose derivation tree is T_c. We restore the tree T'' by replacing each occurrence of T'_c in T'' with T_c. Then, we obtain T''' shown in Fig. 9, which is a derivation tree from $\mathcal{B}^{nll}$ together only with the original EDB D and has the same root as T. Since $\mathcal{B}^{nl} \subseteq \mathcal{B}^{mb}$, it is clear that there is a left-linear derivation tree from $\mathcal{B}^{mb} \cup D$ with the same root as T''', and thus, as T. $\square$

Proof of Theorem 4.4

It is obvious that $\mathcal{C} \supseteq \mathcal{C}^{nl} \supseteq \mathcal{C}^{slu}$. The necessity holds simply because $\mathcal{C} \equiv \mathcal{C}^{slu}$ implies $\mathcal{C}^{nl} \subseteq \mathcal{C}^{slu}$.

For the sufficiency, we prove by induction on the number of internal nodes of a derivation tree that every derivation tree from $\mathcal{C}$ is contained into a derivation tree from $\mathcal{C}^{slu}$. Let T be a derivation tree with $k(> 1)$ internal nodes from $\mathcal{C}$ and an EDB D. We can find an internal node $p(\overline{c}_0)$ whose children correspond to the internal nodes derived by the exit rule. There can be two types of such an internal node as shown in Fig. 10. One type is that the internal node is derived by the linear rule r_a. The other type is that the internal node is derived by the bilinear rule r_b. For the remainder of this proof, arguments similar to those of the proof for Theorem 4.1 can be applied to each case of the types. $\square$

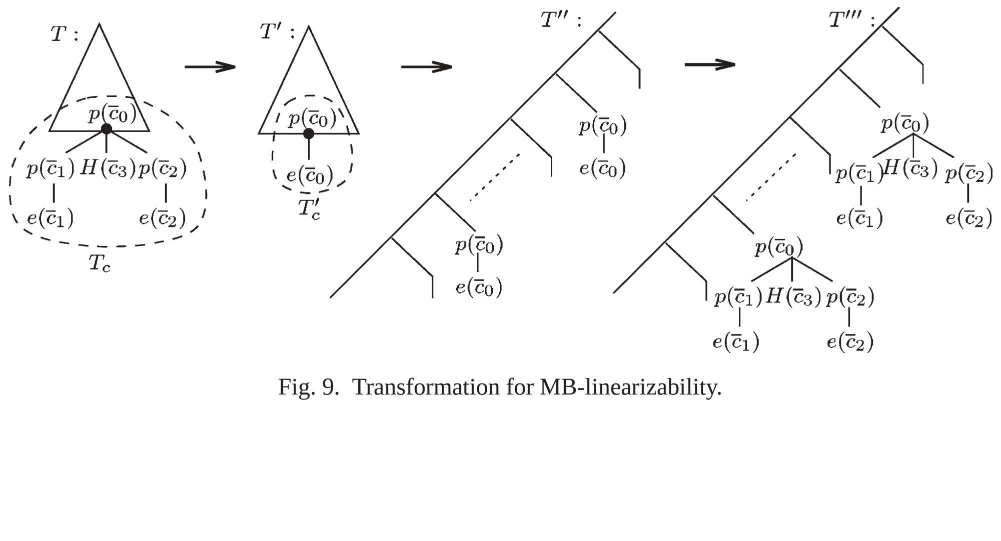
T :
$p(\overline{c}_0)$
$p(\overline{c}_1)$ $H(\overline{c}_3)$ $p(\overline{c}_2)$
$e(\overline{c}_1)$ $e(\overline{c}_2)$
T_c
T' :
$p(\overline{c}_0)$
$e(\overline{c}_0)$
T'_c
T'' :
$p(\overline{c}_0)$
$e(\overline{c}_0)$
T''' :
$p(\overline{c}_0)$
$p(\overline{c}_1)$ $H(\overline{c}_3)$ $p(\overline{c}_2)$
$e(\overline{c}_1)$ $e(\overline{c}_2)$

Fig. 9. Transformation for MB-linearizability.

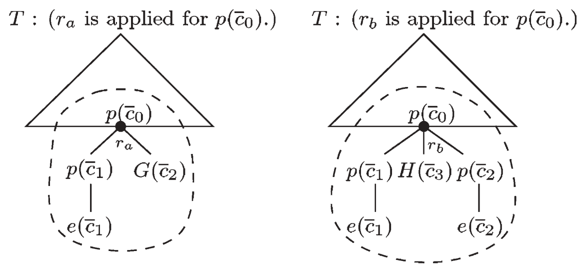
T : (r_a is applied for $p(\overline{c}_0)$.)
$p(\overline{c}_0)$
r_a
$p(\overline{c}_1)$
$G(\overline{c}_2)$
$e(\overline{c}_1)$
T : (r_b is applied for $p(\overline{c}_0)$.)
$p(\overline{c}_0)$
r_b
$p(\overline{c}_1)\, H(\overline{c}_3)\, p(\overline{c}_2)$
$e(\overline{c}_1)$
$e(\overline{c}_2)$

Fig. 10. Two types of internal nodes.

Proof of Theorem 4.7

It is trivial that $\mathcal{C} \supseteq_p \mathcal{C}^{ud} \supseteq_p \mathcal{C}^{sld}$. Also note that $\mathcal{C}^{ud}_p$ is ZYT-linearizable iff $\mathcal{C}^{sld} \equiv_p \mathcal{C}^{ud}$. Therefore, $\mathcal{C}$ is SBSLD-linearizable iff $\mathcal{C} \equiv_p \mathcal{C}^{sld}$ iff $\mathcal{C}^{ud} \equiv_p \mathcal{C}$ and $\mathcal{C}^{ud}_p$ is ZYT-linearizable. $\square$

Proof of Lemma 4.8

It is obvious that $\mathcal{C} \supseteq_p \mathcal{C}^{ud}$. We must prove the reverse containment. Since $r_a \cdot r_s \subseteq r_s \cdot r_a^*$, there exists $m(\geqslant 0)$ such that $r_a \cdot r_s \subseteq r_s \cdot r_a^m$. Therefore, the partial derivation tree T_{as} of Fig. 11 is uniformly contained into the tree T_{sa^*} of the same figure. That is, the two trees have the same root and the conjunction of all the leaves of T_{as} is uniformly contained into the conjunction of all the leaves of T_{sa^*}.

Let D be an EDB. Assume that T is an arbitrary derivation tree from $\mathcal{C} \cup D$. We prove the theorem by induction on the number of applications of the rule r_b in T, denoted by $\sharp_{r_b}(T)$. If $\sharp_{r_b}(T) = 0$, all the applications by recursive rules in T are by the linear rule r_a. Therefore, T is also a derivation tree from $\mathcal{C}^{ud} \cup D$. Assume that for every derivation tree from $\mathcal{C} \cup D$ with $\sharp_{r_b}(T) < n$, there is also a derivation tree from $\mathcal{C}^{ud} \cup D$ with the same root. Now, consider a derivation tree T such that $\sharp_{r_b}(T) = n$. There are two cases to be considered.

One case is that the root of T is derived by r_b. The number of applications of r_b in each of the two subtrees of the root is less than n. By the induction hypothesis, each subtree can be contained into a derivation tree from $\mathcal{C}^{ud} \cup D$. By replacing the original subtrees in T with these new trees from $\mathcal{C}^{ud} \cup D$, we can obtain a new derivation tree, which is from $\mathcal{C}^{ud} \cup D$, with the same root as T.

The other case is by r_a. As we go down the tree T from the root along through the recursive subgoals, we meet the p-facts $p(\overline{a}_0), p(\overline{a}_1), \ldots,$ that have been derived by r_a, and finally arrive at the node $p(\overline{a}_k)$ for some $k(\geqslant 1)$ such that the node $p(\overline{a}_k)$ has been derived by r_b for the first time as

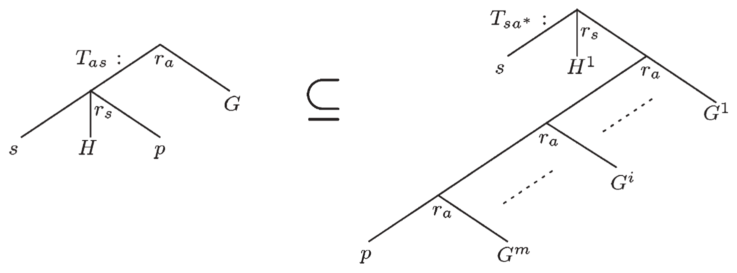

Fig. 11. Commutativity between a linear rule and a bilinear rule.

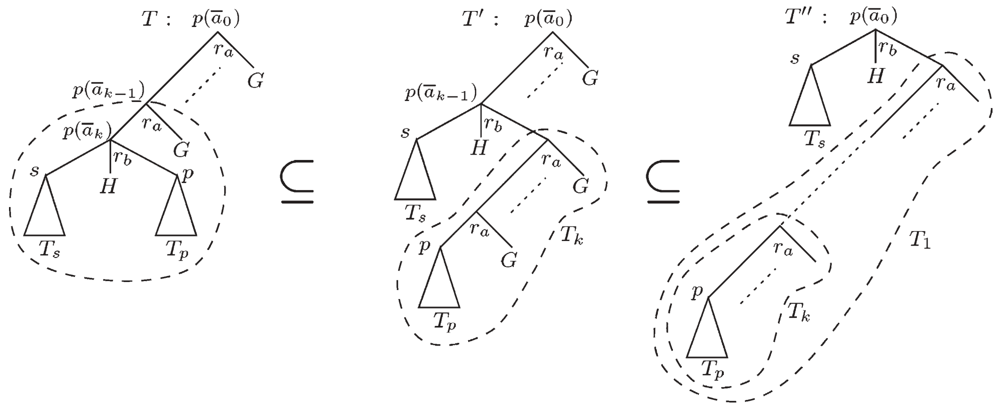

Fig. 12. Containment of derivation trees for SBSLD-linearizability.

shown in Fig. 12. That is, k is the number of applications of r_a above the node $p(\overline{a}_k)$. Such $p(\overline{a}_k)$ exists because $\sharp_{r_b}(T) > 0$. $p(\overline{a}_k)$ has been derived by r_b, and its parent by r_a. Since $T_{as} \subseteq T_{sa^*}$ holds as shown in Fig. 11, by applying this commutativity between the application of r_b for deriving $p(\overline{a}_k)$ and the application of r_a for its parent $p(\overline{a}_{k-1})$, we can obtain a tree T' that contains T as in Fig. 12. Note that there are only $k-1$ applications of r_a above the application of r_b for $p(\overline{a}_{k-1})$ in T'. The transformation by this commutativity can be repeated $k-1$ times for the remaining applications of r_a. After all, we obtain a tree T'' that contains T as in Fig. 12. Note that $\sharp_{r_b}(T_s) < n$. By induction hypothesis, there is a derivation tree T_s' from $\mathcal{C}^{ud} \cup D$ with the same root as T_s. And since $\sharp_{r_b}(T_1) < n$, there is a derivation tree T_1' from $\mathcal{C}^{ud} \cup D$ with the same root as T_1. In T'', by replacing T_s with T_s', and T_1 with T_1', we have a derivation tree from $\mathcal{C}^{ud} \cup D$ with the same root as T. □

References

[1] F. Bancilhon, R. Ramakrishnan, An Amateur's introduction to recursive query processing strategies, in: Proc. 1986 ACM SIGMOD Int'l Conf. on Management of Data, Washington, DC, May 1986, pp. 16–52.
[2] S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Springer, Berlin, 1990.
[3] A. Chandra, P.M. Merlin, Optimal implementation of conjunctive queries in relational databases, in: Proc. 9th ACM Symp. on the Theory of Computing, 1977, pp. 77–90.
[4] O.M. Duschka, M.R. Genesereth, Answering recursive queries using views, in: Proc. 16th ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Database Systems, Tucson, AZ, USA, May 1997, pp. 109–116.
[5] H. Gaifman, H. Mairson, Y. Sagiv, M. Vardi, Undecidable optimization problems for database logic programs, in: Proc. 2nd ACM Symp. on Logic in Computer Science, San Diego, CA, USA, March 1987, pp. 106–115.
[6] K.-H. Hong, Y.-J. Lee, K.-Y. Whang, Dynamically ordered semi-naive evaluation of recursive queries, Information Sciences 96 (3/4) (1997) 237–269.
[7] P.G. Kolaitis, M.Y. Vardi, Conjunctive-query containment and constraint satisfaction, in: Proc. 17th ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Database Systems, Seattle, WA, USA, June 1998, pp. 205–213.
[8] J.F. Naughton, R. Ramakrishnan, Y. Sagiv, J.D. Ullman, Argument reduction by factoring, Theoretical Computer Science 146 (1/2) (1995) 269–310.
[9] R. Ramakrishnan, Y. Sagiv, J.D. Ullman, M. Vardi, Proof tree transformation theorems and their applications, Journal of Computer and System Sciences (1993) 222–248.
[10] Y. Sagiv, Optimizing datalog programs, in: Proc. 6th ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Databases Systems, San Diego, CA, USA, March 1987, pp. 349–362.

[11] Y.P. Saraiya, Linearizing nonlinear recursions in polynomial time, in: Proc. Eighth ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Database Systems, Philadelphia, PA, USA, March 1989, pp. 182–189.
[12] Y.P. Saraiya, Hard problems for simple logic programs, in: Proc. 1990 ACM SIGMOD Int'l Conf. on Management of Data, Atlantic, NJ, USA, June 1990, pp. 64–73.
[13] Y.P. Saraiya, On the efficiency of transforming database logic programs, Jorunal of Computer and System Sciences 51 (1) (1995) 87–109.
[14] O. Shmueli, Decidability and expressiveness aspects of logic queries, in: Proc. 6th ACM SIGACT–SIGMOD–SIGART Symp. on Principles of Databases Systems, San Diego, CA, USA, March 1987, pp. 237–249.
[15] D. Srivastava, S. Sudarshan, R. Ramakrishnan, J. Naughton, Space optimazation in deductive databases, ACM Transactions on Database Systems 20 (4) (1995) 472–516.
[16] J.D. Ullman, Principles of Database and Knowledge–Base Systems, Computer Science Press, Rockville, MD, 1988.
[17] J.D. Ullman, Principles of Database and Knowledge–Base Systems, Computer Science Press, Rockville, MD, 1989.
[18] K.-Y. Whang, S.B. Navathe, An extended disjunctive normal form approach for optimizing recursive logic queries in loosely coupled environments, in: Proc. Thirteenth Int'l Conf. on Very Large Data Bases, Brighton, UK, September 1987, pp. 275–287.
[19] W. Zhang, C.T. Yu, A necessary condition for a doubly recursive rule to be equivalent to a linear recursive rule, in: Proc. 1987 ACM SIGMOD Int'l Conf. on Management of Data, San Francisco, CA, USA, May 1987, pp. 345–356.
[20] W. Zhang, C.T. Yu, D. Troy, A necessary and sufficient condition to linearize doubly recursive programs in logic databases, ACM Transactions on Database Systems 15 (3) (1990) 459–482.