Linearizability of $n$-linear Sirups

Héctor J. Hernández*  
Centro de Estudios Tecnológicos  
Instituto de Ingeniería y Tecnología  
Universidad Autónoma de Cd. Juárez  
Cd. Juárez, Chih., México CP 32320  
hector@uacj.mx

Dongxing Tang  
Lab for Logic and Databases  
Dept. of Computer Science  
New Mexico State University  
Las Cruces, NM, USA 88003  
dtang@cs.nmsu.edu

Article received on June 16, 1998; accepted on September 11, 1998

Abstract

A linear program is easier to evaluate than a nonlinear program. Hence, given a recursive program, it is desirable to find an equivalent linear program. However, not all nonlinear programs are linearizable. Theoretically, an $m$-linear program is easier to evaluate than an $n$-linear program when $m < n$, since the derivation tree of the former one is of smaller arity than the derivation tree of the latter. Thus, when an $n$-linear program is not linearizable, we would like to find another, equivalent $m$-linear program with $m < n$.

In this paper, we consider two possibilities of linearizing $n$-linear sirups. First, we consider the equivalence between an $n$-linear sirup and its derivative or its general ZYT-linearization, which are linear programs. We show that the problem of determining whether an $n$-linear sirup is equivalent to its derivative or to its general ZYT-linearization is NP-hard. We then give a tighter condition which is necessary and sufficient for testing those equivalences. The other possibility is to consider the equivalence between an $n$-linear sirup and another $m$-linear program, $m < n$, called its $k$-ZYT-linearization, where $k = n - m$. We also prove that the problem of determining whether an $n$-linear sirup is equivalent to its $k$-ZYT-linearization is NP-hard. Then, we present a tighter, exact condition for testing whether an $n$-linear sirup is equivalent to its $k$-ZYT-linearization. We do not know whether testing any of the above equivalences is decidable.

Keywords: deductive databases, optimization, data-logs, linearization, sirups (single recursive programs)

1 Introduction

A linear Datalog program is easier to evaluate than a nonlinear program [Ullman & Van Gelder, 1986] because there is one path in every proof tree. For example, in [Ullman, 1989] an efficient Kleene's Algorithm is proposed for computing the Generalized Transitive Closure program. Therefore, given any arbitrary nonlinear Datalog program, we would like to find another linear program which is equivalent to the original one. Unfortunately, this problem is known to be undecidable in general [Gaifman et al., 1993] [Saraiya, 1990].

The linearization of some bilinear Datalog programs has been studied. In particular, given a bilinear sirup [Kanellakis, 1987] $p$ of the form:

\[
p(X_1, \ldots, X_n) : - q(X_1, \ldots, X_n),
\]

where $G$ is a conjunction of extensional database (EDB) predicates, we want to know whether $p$ is equivalent to the following program $p'$:

\[
p'(X_1, \ldots, X_n) : - q(X_1, \ldots, X_n),
\]

If $p$ is equivalent to $p'$, it is said that $p$ is ZYT-linearizable [Zhang et al., 1990]. It is shown in [Saraiya, 1990] that the problem of testing whether a bilinear sirup is ZYT-linearizable is undecidable; if $G$ does not have repeated predicate names and some other conditions are satisfied, that problem becomes decidable [Zhang et al., 1990]. Ramakrishnan et al. [Ramakrishnan et al., 1993] considered a case of ZYT-linearization in which they allow repeated predicates in $G$. They proved that it is NP-hard to determine whether a bilinear sirup such as $p$ is ZYT-linearizable.

Notice that a bilinear sirup has two ZYT-linearizations, which are obtained by expanding the two recursive predicates in the recursive rule respectively. In this paper, we consider the derivative, which is a linear
program that includes these two linearizations of the original program. The derivative of the above program \( p \) is the following program.

\[
p(\overline{x}) = \\
p(x_1, x_2) : - e(x_1, x_2).
\]

If \( p \) is equivalent to \( p(\overline{x}) \), \( p \) is called differentiable.

Clearly, the derivative of a program is more general than its ZYT-linearization, in the sense that the derivative of a bilinear program contains its ZYT-linearization: when a program is not ZYT-linearizable, the derivative of a bilinear program can compute more facts than its ZYT-linearization. The following example shows this.

**Example 1.1:** Given the following bilinear sirup
\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- p(X, Z), p(Z, Y), a(X, Z), b(Z, Y).
\]

Its ZYT-linearization and derivative are as follows:

\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- e(X, Y), e(Z, Y), a(X, Z), b(Z, Y).
\]
\[
p(X, Y) :- e(X, Y), e(Z, Y), a(X, Z), b(Z, Y).
\]

Let us now consider the output of the above two linear programs with the following input:

\[
\{e(1, 2), e(2, 3), e(3, 4), a(1, 2), a(3, 3), b(2, 3), b(3, 4)\}.
\]

It is not difficult to check that the derivative computes the fact \( p(1, 4) \), which is not in the output of the ZYT-linearization. That is, the derivative can produce more facts than the ZYT-linearization when the original program is not ZYT-linearizable. Thus, the derivative of a given bilinear program approximates the original program better than its ZYT-linearization does when the given bilinear program is not ZYT-linearizable. Later, we show that this bilinear sirup is not differentiable.

Similar to the derivative of bilinear sirups, the derivative of an \( n \)-linear sirup consists of \( n \) linear rules which are obtained by expanding all but one of the recursive atoms in the recursive rule by the basis rule. However, when \( n \) is large, the number of rules in the derivative is large. Thus, the derivate is still not efficient to evaluate. Hence we consider other ways to linearize \( n \)-linear sirups.

One way is to consider the equivalence of the original program and a program, called its general ZYT-linearization, which is composed of the basis rule and the recursive rule with all the recursive predicates except one (arbitrarily chosen) substituted by the EDB predicate in the basis rule. The following example illustrates this.

**Example 1.2:** Consider the following program that finds all paths of odd length in \( e \):
\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- p(X, U), p(U, V), p(V, Y).
\]

In Section 5, we show that this program is general ZYT-linearizable. That is, it is equivalent to the following linear program:
\[
p(X, Y) :- e(X, Y).
\]
\[
p(X, Y) :- e(U, V), e(V, Y).
\]

However, not every \( n \)-linear sirup is general ZYT-linearizable. Let us show such a program.

**Example 1.3:** Let \( p \) be the following program:

\[
e(S) :- e_0(S).
\]
\[
e(S) :- e(T), e(U), e(V), e_1(S, T, U, V).
\]

We want to test whether this program is general ZYT-linearizable, that is, we want to check whether \( p \) is equivalent to the following program:

\[
p_1 : \\
e(S) :- e_0(S).
\]
\[
e(S) :- e(T), e(U), e(V), e_1(S, T, U, V).
\]

Let us now consider the following facts as input to \( p \) and \( p_1 \):
\[
\{e_0(1), e_0(2), e_0(3), e_0(4), e_0(5), e_0(6), e_0(7), e_1(2, 3, 4), e_1(9, 5, 6, 7), e_1(10, 1, 8, 9)\}.
\]

It is not difficult to check that the fact \( e(10) \) is produced from \( p \) but not from \( p_1 \). Hence, the original program is not general ZYT-linearizable. Note that this program is shown not to be linearizable in [Afrati & Cosmadakis, 1989].

If an \( n \)-linear sirup is not general ZYT-linearizable, theoretically, it may be interesting to consider whether it might be equivalent to its \( k \)-ZYT-linearization, which consists of the basis rule and one recursive rule obtained by replacing the first \( k \) (\( 1 \leq k < n \)) recursive predicates by the EDB predicate in the basis rule. If they are equivalent, we say that the original program is \( k \)-ZYT-linearizable. Thus, we might check whether \( p \) in the previous example is 1-ZYT-linearizable, that is to check whether it is equivalent to the following program.

\[
p_2 : \\
e(S) :- e_0(S).
\]
\[
e(S) :- e(T), e(U), e(V), e_1(S, T, U, V).
\]

It is clear that given a bilinear sirup, its 1-ZYT-linearization is its ZYT-linearization [Zhang et al., 1990]. Moreover, when \( k = n - 1 \), then the \( k \)-ZYT-linearization is the general ZYT-linearization. Hence, both derivative and general ZYT-linearization generalize ZYT-linearization, while the \( k \)-ZYT-linearization is the generalization of general ZYT-linearization.

The rest of the paper is organized as follows: Section 2 gives relevant definitions and some simple facts on Cartesian products of relations used in the paper. Section 3 considers the differentiability of a bilinear sirup. Section 4 generalizes the results in Section 3 and considers the derivative of an \( n \)-linear program with \( n \geq 3 \). Section 5 generalizes ZYT-linearization to the case of an \( n \)-linear sirup for arbitrary \( n \). Section 6 considers \( k \)-ZYT-linearization. Our conclusion is given in the last section.
2 Definitions and Basic Facts

This paper considers only Datalog programs, which in the sequel shall be referred to as programs. We assume the reader is familiar with standard terminology (see Ullman, 1988), and only define some of the relevant, non-standard notation used in the paper.

A sirup is a program which consists of one recursive rule and one base, nonrecursive rule [Kanellakis, 1987], called basis (or initialization) rule, such that both rules define the same intensional database (IDB) predicate. A sirup is simple if its basis rule has only one atom in its body, i.e., its basis rule is of the form \( p(X_1, X_2, \ldots, X_n) \). In this paper, we restrict ourselves to such simple programs. A sirup is \( n \)-linear if the predicate in the head of the recursive rule appears \( n \) times in its body.

A proof tree is a tree description for the derivation of an intensional fact by the application of some rules to extensional facts and the set of intensional facts generated earlier. The leaves of a proof tree must be EDB literals. The root, as well as any nonleaf node, must be an IDB literal. Every nonleaf node represents an application of a rule. The children of such a node are the subgoals in the body of the rule, instantiated to suitable EDB and IDB facts, of which the node is the head.

Given a program \( P \) and a database \( I \), \( P(I) \) denotes the output of \( P \) when \( I \) is its input; that is, \( P(I) \) is the least fixed point of \( P \) with respect to \( I \) [Ullman, 1988]. If \( r \) is a rule, \( r(I) \) shall denote the output of program \( \{ r \} \) when \( I \) is its input. Given two programs \( P \) and \( Q \), \( P \) is contained in \( Q \), denoted \( \mathcal{P} \subseteq \mathcal{Q} \), if over any database \( I \), defined only on EDB predicates, \( \mathcal{P}(I) \subseteq \mathcal{Q}(I) \); \( P \) is contained in \( Q \) w.r.t. predicate \( q \), denoted \( \mathcal{P} \subseteq_q \mathcal{Q} \), if over any database \( I \), defined only on EDB predicates, every fact \( t \) defined over predicate \( q \) in \( \mathcal{P}(I) \) is also in \( \mathcal{Q}(I) \). \( P \) is equivalent to \( Q \), written \( P \equiv Q \), if \( \mathcal{P} \subseteq \mathcal{Q} \) and \( \mathcal{Q} \subseteq \mathcal{P} \). \( P \) is equivalent to \( Q \) w.r.t. predicate \( q \), written \( P \equiv_q Q \), if \( \mathcal{P} \subseteq_q \mathcal{Q} \) and \( \mathcal{Q} \subseteq_q \mathcal{P} \). Note that the above definition of equivalence of two programs is over extensional databases. If we allow both extensional and intensional databases as input, the equivalence is called uniform equivalence; \( P \equiv_u Q \) denotes that \( P \) is uniformly equivalent to \( Q \). The problem of testing equivalence of two programs is undecidable [Shmueli, 1987], while the problem of testing uniform equivalence of two programs is decidable [Sagiv, 1987].

2.1 Graph \( G \) and rule \( r_G \)

In order to prove \( NP \)-hardness of the linearization problems studied here, we first show an important result from [Kanellakis, 1987]. Let \( G \) be a graph. Kanellakis has constructed the following rule from the graph \( G \).

\[
r_G: q(X, Y, Z, V) \iff q(X, Y, Z, W), \psi_G(W, V)
\]

where \( r_G \) contains a number of literals representing the graph; in particular, \( \psi_G \), which is a conjunction of EDB literals with occurrences of \( V \) and \( W \), is used for that purpose; in this paper, without loss of generality, we assume that \( \psi_G \) is an EDB atom.

We say that \( r_G \) is \( 1 \)-bounded if for any \( I \), where \( I \) is a set of atoms defined on \( q \) and \( \psi_G \), \( r_G(I) = r_G^1(I) \), where the equality is restricted on \( q \)-facts, and \( r_G^1(I) \), the nonrecursive application of \( r_G \) to \( I \), is \( r_G^1(I) = \{ q(a_1, a_2, a_3, a_4) \mid q(a_1, a_2, a_3, a_4) \in I \} \).

Kanellakis proved the following result (Theorem 2 in [Kanellakis, 1987]) that we shall use later in our proofs.

**Theorem 2.1**: \( G \) is three-colorable iff \( r_G \) is \( 1 \)-bounded.

2.2 Some Results on Cartesian Products

We now state some simple results about Cartesian products of relations that are needed later on in the paper. Before doing that, we need to define the following. Let \( r \) be a relation. Then, we define:

- \( r^0 = \{ \epsilon \} \), where \( \epsilon \) is the empty tuple such that \( r^0 \times \{ \epsilon \} = \{ \epsilon \} \times r^0 \) for any relation \( r \); and
- \( r^n = r \times r^{n-1} \), if \( n \geq 1 \).

**Lemma 2.1**: Let \( \mathcal{P} \) be a relation, let \( r_1 \) be a subset of \( r \), and assume that \( r \) and \( r_1 \) are not empty. Then for any \( n, n \geq 2 \), \( r_1 \times r^{n-1} \cup (r_1 \times r^{n-2}) \cup \cdots \cup (r_1 \times r) = r^n \) if \( r_1 \subset r \).

**Proof**: The if-part is obvious. For the other direction, let us suppose that \( r \neq r_1 \). Then let \( t \) be a tuple in \( r - r_1 \). It is not difficult to see that \( t' \), the tuple in \( \{(t)\} \), is not a member of \( r^n \). Because each of the terms in (2) has \( n - 1 \) occurrences of \( r_1 \), and \( t \) does not belong in \( r_1 \).

3 Differentiability of bilinear sirups

It is shown in [Ramakrishnan et al., 1993] that the problem of deciding whether a bilinear sirup is \( ZYT \)-linearizable is \( NP \)-hard. In this section, we prove that the problem of deciding if a bilinear sirup is differentiable is also \( NP \)-hard; we do not know if it is decidable. In addition, we show a necessary and sufficient condition of differentiability of bilinear sirups.
\( q(Z_1, \ldots, Z_n), G', p(Z_1, \ldots, Z_n), G. \)

\( r''_4: p(X_1, \ldots, X_n) > q(Y_1, \ldots, Y_n), q(Y_1', \ldots, Y_n'), q(Z_1', \ldots, Z_n'), G'', G. \)

\( \rho_3 \) is obtained from \( \rho_2 \) by substituting predicate \( p \) for predicate \( s \). \( \rho_2 \) consists of \( \rho_1 \) with the recursive predicate \( s \) replaced by \( s \) submitted to \( \rho_2 \) by rule \( r''_4 \). The rules \( r''_4 \) and \( r''_3 \) are obtained by expanding the first recursive atom in the body of \( r_1 \) by \( r_1 \) itself and then substituting the first two recursive predicates \( p \) in the expanded rule with the exit predicate \( q \) and the last recursive predicate \( s \); \( G' \) is \( G \) after the expansion of the first recursive atom of \( r_1 \); \( r''_3 \) is obtained similarly by expanding the second recursive atom in the body of rule\( r_1 \) and then substituting the last two recursive predicates \( p \) with the exit EDB predicate \( q \) and the first recursive predicate \( s \); \( G'' \) denotes \( G \) after the expansion in which we substitute the second recursive atom of \( r_1 \).

In the following theorem, we show that the problem of differentiability of \( p \) may be reduced to the problem of testing the equivalence of two linear programs.

**Theorem 3.2:** Let programs \( p, \rho_1, \) and \( \rho_2 \) be as defined above. Then, \( p \) is differentiable if and only if \( \rho_1 \equiv \rho_2 \).

**Proof:** Clearly, \( \rho_1 \subseteq p \subseteq \rho_2 \subseteq p \). Thus, if \( p \) is differentiable, we have \( p \subseteq \rho_1 \). Hence, \( \rho_1 \equiv \rho_2 \).

Now we prove the opposite direction. Assume that \( \rho_1 \equiv \rho_2 \). We shall prove by induction on the height of a proof tree that \( p \) has a proof tree \( \rho_1 \) which also derives \( q(v) \), which we marked above to distinguish them, in the proof tree and return to the original database as shown in Figure 2(b); let us use \( T' \) to denote the proof tree obtained in this step. We now prove that for each sub-proof tree like the one rooted at \( p_1 \) on Figure 2(b) there is a proof tree in \( \rho_2 \) that computes \( p_1 \), and hence a proof tree in \( \rho_1 \) (since \( \rho_2 \subseteq p \)). This shall prove that we can transform a tree like \( T' \) into a proof tree from \( \rho_1 \).

We walk up \( T' \), beginning at the bottom until we find the interior node \( p_1 \) with either a left or right branch that came from a compact node. From the proof tree rooted at \( p_1 \), we can obtain a proof tree in \( \rho_2 \) as shown in Figure 2(c) that derives the same fact \( p_1 \); the leftmost subtree is the counterpart of the leftmost subtree in the proof tree shown in the box of Figure 2(b), where we replaced the predicate \( p \) by \( s \); the \( q \)-facts and \( G \) and \( G' \) are the same as in the proof tree shown in the box; the proof tree rooted at \( p_1 \) in Figure 2(c) represents an application of rule \( r''_3 \) at the top. Since \( \rho_2 \subseteq p \), we have a proof tree from \( p_1 \), which produces the same fact \( p_1 \). The other kind of sub-proof tree transformation requires the application of rule \( r''_4 \) instead of \( r''_3 \) as above. This completes the proof of this part of the theorem and the proof of the theorem itself.

Note that in fact when we test if a bilinear program is differentiable, we can directly test if its derivative is equivalent to the original one. However, Theorem 3.2 shows us a tighter condition since both \( \rho_1 \) and \( \rho_2 \) are linear. We next use \( \rho_3 \) to give a condition to test the differentiability of \( p \).

**Theorem 3.3:** Let \( p, \rho_1, \) and \( \rho_3 \) be as defined previously. Then \( p \) is differentiable if and only if \( \rho_1 \) is equivalent to \( \rho_3 \).

![Figure 1: Reducing the height of a proof tree](image-url)
That fact. For the proof tree with height 1, it is trivial (because proof trees for facts produced by \( P_0 \) are compacted and hence a proof tree of the leftmost subtree in the proof tree shown in the box of Figure 2(b), where we replaced the predicate \( p \) by \( s \); the \( q \)-facts and \( G \) and \( G' \) are the same as in the proof tree shown in the box; the proof tree rooted at \( p_1 \) in Figure 2(c) represents an application of rule \( r_{s}^{G'} \) at the top. Since \( p_2 \subseteq p_1 \), we have a proof tree from \( p_1 \), which produces the same fact \( p_1 \). The other kind of sub-proof tree transformation requires the application of rule \( r_{s}^{G} \) instead of \( r_{s}^{G'} \) as above. This completes the proof of this part of the theorem and the proof of the theorem itself. □

Note that in fact when we test if a bilinear program is differentiable, we can directly test if its derivative is equivalent to the original one. However, Theorem 3.2 shows us a tighter condition since both \( p_1 \) and \( p_2 \) are linear. We next use \( p_3 \) to give a condition to test the differentiability of \( p \).

Theorem 3.3: Let \( p, p_1, \) and \( p_3 \) be as defined previously. Then \( p \) is differentiable if and only if \( p_1 \) is equivalent to \( p_3 \).

By the inductive hypothesis, there exists a proof tree as shown in Figure 2(a), which derives the same tuple in \( p_1 \) from the new relation \( Q'' \); the proof tree of the derivative of a bilinear program looks like a zigzag (one in which either left-hand or right-hand recursive nodes are expanded by the basis rule). We next uncompact all the tuples \( q(v) \), which we marked above to distinguish them, in the proof tree and return to the original database as shown in Figure 2(b); let us use \( T' \) to denote the proof tree obtained in this step. We now prove that for each sub-proof tree like the one rooted at \( p_1 \) on Figure 2(b) there is a proof tree in \( p_2 \) that computes \( p_1 \), and hence a proof-tree in \( p_1 \) (since \( p_2 \subseteq p_1 \)). This shall prove that we can transform a tree like \( T' \) into a proof tree from \( p_1 \).

We walk up \( T' \), beginning at the bottom until we find the first interior node \( p_1 \) with either a left or right branch that came from a compact node. From the proof tree rooted at \( p_1 \), we can obtain a proof tree in \( p_2 \) as shown in Figure 2(c) that derives the same fact \( p_1 \); the leftmost subtree is the counterpart of the leftmost subtree in the proof tree shown in the box of Figure 2(b), where we replaced the predicate \( p \) by \( s \); the \( q \)-facts and \( G \) and \( G' \) are the same as in the proof tree shown in the box; the proof tree rooted at \( p_1 \) in Figure 2(c) represents an application of rule \( r_{s}^{G'} \) instead of \( r_{s}^{G} \) as above. This completes the proof of this part of the theorem and the proof of the theorem itself. □

Note that in fact when we test if a bilinear program is differentiable, we can directly test if its derivative is equivalent to the original one. However, Theorem 3.2 shows us a tighter condition since both \( p_1 \) and \( p_2 \) are linear. We next use \( p_3 \) to give a condition to test the differentiability of \( p \).

Theorem 3.3: Let \( p, p_1, \) and \( p_3 \) be as defined previously. Then \( p \) is differentiable if and only if \( p_1 \) is equivalent to \( p_3 \).
Proof: It is easy to see that $p_1 \subseteq_p p_2 \subseteq_p p_3 \subseteq_p p$. Thus, the only-if part is trivial. The if part follows from Theorem 3.2. □

The problem of testing whether two Datalog programs are equivalent is undecidable [Shmueli, 1987]. Hence, the tests given by both Theorem 3.2 and Theorem 3.3 are noneffective. Fortunately, uniform equivalence is decidable [Sagiv, 1987] and uniform equivalence implies equivalence. Moreover, by Sagiv [Sagiv, 1987], to check if $p_3$ is uniformly contained in $p_1$ suffices to check if $r_3''$ and $r_4''$ are contained in $p_1$ individually. Since $p_1$ is a linear program, when $r_3''$ or $r_4''$ is considered as a conjunctive query and its body is viewed as a database input to $p_1$, we can search for the derivations of $p_3$ using a nondeterministic polynomial space algorithm [Chandra et al., 1981]. Following the above discussion, we have the following theorem.

**Theorem 3.4:** Let $p$, $p_1$, and $p_2$ be as defined previously. Then, $p$ is differentiable if $p_2$ is uniformly equivalent to $p_3$. This condition can be tested in polynomial space.

**Example 3.5:** Consider the following program:

\[
\begin{align*}
r_0 & : p(X, Y) :- e(X, Y), \\
r_1 & : p(X, Y) :- p(X, Z), p(Z, Y), a(X, Z), b(Z, Y).
\end{align*}
\]

We want to test whether this program is differentiable. It is well-known that the bilinear version of the transitive closure problem (i.e., the recursive rule $r_1$ in $p$ without $a(X, Z)$ and $b(Z, Y)$) is equivalent to its $ZYT$-linearization, hence it is differentiable. However, we now show that the above program $p$ is not differentiable. Note that the derivative of $p$ is as follows:

\[
\begin{align*}
r_0' & : p(X, Y) :- e(X, Y), \\
r_1' & : p(X, Y) :- p(X, Z), e(Z, Y), a(X, Z), b(Z, Y), \\
r_2' & : p(X, Y) :- e(X, Z), p(Z, Y), a(X, Z), b(Z, Y). \\
r_3' & : p(X, Y) :- e(X, Z), e(Z, Z), e(Z, Y), a(X, Z), b(Z, Y), \\
r_4' & : p(X, Y) :- e(X, Z), e(Z, Y), a(X, Z), b(Z, Y).
\end{align*}
\]

To test whether $p$ is differentiable, by Theorem 3.3, we need to check whether $p_1$ is equivalent to $p_3$ which consists of $p_1$ and rules $r_3'$ and $r_4'$:

\[
\begin{align*}
r_0' & : p(X, Y) :- e(X, Y), \\
r_1' & : p(X, Y) :- p(X, Z), e(Z, Y), a(X, Z), b(Z, Y), \\
r_2' & : p(X, Y) :- e(X, Z), p(Z, Y), a(X, Z), b(Z, Y). \\
r_3' & : p(X, Y) :- e(X, Z), e(Z, Z), e(Z, Y), a(X, Z), b(Z, Y), \\
r_4' & : p(X, Y) :- e(X, Z), e(Z, Y), a(X, Z), b(Z, Y).
\end{align*}
\]

Let us now consider the following set of facts as input:

\[
\{e(1, 2), e(2, 3), e(3, 4), e(4, 5), a(1, 2), a(3, 4), a(1, 3), b(2, 3), b(4, 5), b(3, 5)\}
\]

We can check that $p_1$ computes the following $p$-facts:

\[
\{p(1, 2), p(2, 3), p(3, 4), p(4, 5), p(1, 3), p(3, 5)\}
\]

while $p_3$ computes $p$-facts as follows:

\[
\{p(1, 2), p(2, 3), p(3, 4), p(4, 5), p(1, 3), p(3, 5), p(1, 5)\}
\]

Note the difference $p(1, 5)$ between the two outputs. Thus, $p_3$ is not contained in $p_1$. Therefore, by Theorem 3.3, we conclude that the original program $p$ is not differentiable. □
4 Differentiability of \( n \)-linear sirups

In the preceding section, we concentrated on bilinear programs. We have shown that it is \( \text{NP} \)-hard to determine whether a bilinear program is differentiable and we have also given a necessary and sufficient condition of its differentiability. In this section, we consider \( n \)-linear sirups.

**Theorem 4.1:** It is \( \text{NP} \)-hard to decide whether an \( n \)-linear sirup is differentiable.

**Proof:** The proof idea is identical to the one in Theorem 3.1, except that we use the following \( n \)-linear program \( p \) in which the recursive predicate has 4n arguments; its derivative \( p_i \) is also shown below.

\[
p(X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, X_{23}, X_{24}, \ldots, X_{n1}, X_{n2}, X_{n3}, X_{n4}) \rightarrow p(X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, X_{23}, X_{24}, \ldots, X_{n1}, X_{n2}, X_{n3}, X_{n4}).
\]

\[
p(X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, X_{23}, X_{24}, \ldots, X_{n1}, X_{n2}, X_{n3}, X_{n4}) \rightarrow p(X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, X_{23}, X_{24}, \ldots, X_{n1}, X_{n2}, X_{n3}, X_{n4}).
\]

The interested reader is referred to [Tang, 1996] for the details of the proof. □

### 4.1 Additional Conditions for Differentiability of trilinear sirups

Theorem 4.1 establishes a lower bound in the complexity of differentiability of an \( n \)-linear sirup. As in the case for bilinear sirups, given an \( n \)-linear program \( p \), there are linear programs \( p_2 \) and \( p_3 \) such that \( p \) is differentiable, i.e., equivalent to its derivative \( p_1 \), if and only if \( p_1 \approx p_2 \) or \( p_1 \equiv p_3 \).

Those programs are shown next; \( G \) is a conjunction of EDB literals. Theorems 3.2, 3.3, and 3.4 hold for these programs [Tang, 1996].
5 General ZYT-linearizability of an n-linear sirup

So far we have considered the problem of whether a given n-linear program is equivalent to its derivative. The derivative of an n-linear program essentially consists of n linear rules. However, when the number of linear rules in a program is large, the computation could be very inefficient. Therefore, it is still interesting to determine whether an n-linear program is equivalent to a linear sirup when n is large. For example, given the following n-linear program,

\[ p : p(X_1, X_2, \ldots, X_n) :- q(X_1, X_2, \ldots, X_n). \]

we want to find whether it is equivalent to the following linear sirup:

\[ p' : p(X_1, X_2, \ldots, X_n) :- \psi(X_1, X_2, \ldots, X_n). \]

where the linear recursive rule is obtained from \( r_1 \) by substituting predicate \( q \) for all predicates except the last one (in fact, any predicate \( p \) can be chosen). Recall that \( \psi' \) is called the general ZYT-linearization of \( \psi \). Moreover, when \( p \equiv p' \), we say that \( p \) is general ZYT-linearizable. R. Ramakrishnan et al. [1993] considered the case of \( n = 2 \). We now generalize their result to the case of n-linear programs with \( n \geq 3 \). In Section 5.1 we shall show a tighter condition which is necessary and sufficient to determine this equivalence.

**Theorem 5.1:** It is NP-hard to determine whether a sirup is general ZYT-linearizable.

**Proof:** We first construct an n-linear program \( p \) as follows:

\[ p : p(X_1, X_2, X_3, \ldots, X_n). \]

where \( \psi \) is a program obtained from \( p \) by substituting predicate \( p \) for predicate \( s \); \( r_1^n \) is obtained by first expanding the second recursive atom \( p \) once in the body of the recursive rule in \( p \) and then substituting the last four recursive predicates \( p \) in the expanded rule with the exit predicate \( q \) and the first \( p \) predicate with \( s \); \( r_1^n \) is obtained by expanding the last recursive predicate \( p \) in \( r_1 \) by itself and then substituting the last four recursive predicates \( p \) by the exit predicate \( q \) and the \( p \) predicate with \( s \) in the body, while rule \( r_1^n \) is obtained by expanding both the second and last recursive predicates \( p \) in the body of rule \( r_1 \) by \( r_1 \) itself and then substituting all recursive predicates except the first one (which is substituted by \( s \)) by the exit predicate \( q \). \( r_1^n \) and \( r_1^n \) are obtained in the same way.

5.1 Additional Conditions for general ZYT-linearizability of trilinear sirups

As in the previous sections, given an n-linear sirup \( p \) there are linear programs \( p_2 \) and \( p_3 \) such that \( p \) is general ZYT-linearizable if and only if \( p_2 \equiv p \) or \( p_1 \equiv p_3 \). However, when \( p_1 \) is the general ZYT-linearization of \( p \). We also have a sufficient condition for testing the general ZYT-linearizability using \( p_3 \), as the one given in Theorem 3.3 for differentiability. For the case \( n = 3 \), if \( p \) is the following program, \( p_2 \) and \( p_3 \) are shown next.

\[ p : \]

\[ r_0 : p(Y_1, Y_2, \ldots, Y_3). \]

\[ r_1 : p(Y_1, Y_2, \ldots, Y_3) :- \psi(Y_1, Y_2, \ldots, Y_3). \]

where \( \psi \) is the EDB predicate in \( r_1 \), the rule defined in Section 2.1. We also have the following program \( p' \), the general ZYT-linearization of \( p \):

\[ p' : \]

\[ r_0' : p(X_1, X_2, X_3, \ldots, X_n) :- p(Y_1, Y_2, \ldots, Y_3). \]

\[ r_1' : p(X_1, X_2, X_3, \ldots, X_n) :- \psi(Y_1, Y_2, \ldots, Y_3). \]

\[ \psi : \]

\[ \psi'' : \]

\[ \psi''' : \]

Then we can prove that \( p \) is general ZYT-linearizable iff \( r_0 \) is 1-bounded using the same technique as in the proof of Theorem 3.1.
6 k-ZYT-linearizability of an n-linear sirup

In this section, we consider the k-ZYT linearization of an n-linear sirup p which is obtained from p by substituting the exit predicate in the nonrecursive rule for the first k IDB predicates in the recursive rule (1 ≤ k < n). This linearization is more general than both the ZYT-linearization and the general ZYT-linearization. In the following theorem, we prove that the problem of determining whether an n-linear sirup is k-ZYT-linearizable is NP-hard. Furthermore, we shall show a tighter sufficient and necessary condition to test k-ZYT-linearizability of an n-linear sirup.

Theorem 6.1: It is NP-hard to determine whether an n-linear sirup is k-ZYT-linearizable.

Proof: Let rG be the rule defined in Section 2.1. We first construct the following n-linear program, where φG is the EDB predicate in rG's body:

\[ p: \]
\[ r_0: p(X, Y) \iff e(X, Y), \]
\[ r_1: p(X, Y) \iff p(X, Z_1), p(Z_1, Z_2), \ldots, p(Z_{n-1}, Y), \]

According to the results in this section, we can use the results in Section 2.1 to show that it is equivalent to its general ZYT-linearization:

\[ p': \]
\[ r'_0: p(X, Y) \iff e(X, Y), \]
\[ r'_1: p(X, Y) \iff e(X, Z_1), e(Z_1, Z_2), \ldots, e(Z_{n-2}, Z_{n-1}), p(Z_{n-1}, Y). \]

It is well known that the bilinear program to find the transitive closure is ZYT-linearizable [Ullman, 1989]. From the above discussion, therefore, all transitive programs to find all paths composed of either unit paths or paths composed of n subpaths can be expressed by a linear sirup.
testing whether the given n-linear sirup is equivalent to one of the above specific linearizations. Moreover, this tighter condition gives us the possibility to test linearizability by just testing the uniform equivalence between two linear programs.

Acknowledgements: This research was partially supported by NSF grant HRD-9353271 (while both authors were at New Mexico State University). H. J. Hernández thanks Dr. L. Cabibbo (of Università di Roma "La Sapienza") and Professor Domenico Sacca (of Università della Calabria) for their insightful comments on a previous version of this paper.

References


Shmueli, O., "Decidability and Expressiveness Aspects of Logic Queries." In Proceedings of Sixth ACM

Tang, D., "Linearization Based Query Optimization in Datalog." Ph.D. Dissertation, Department of Computer Science, New Mexico State University, Nov. 1996.


Héctor J. Hernández received his Bachelor, Master, and Ph.D. degrees all in Computer Science from Monterrey's Institute of Technology (ITESM Campus Monterrey), University of Waterloo, and University of Alberta, respectively. He was an Assistant Professor in the departments of Computer Science of Texas A&M University and New Mexico State University. He is a Professor at Universidad Autónoma de Cd. Juárez. His research interests are in database management, particularly in query processing.

Dongxing Tang received his Ph.D. in Computer Science from New Mexico State University. He worked as a professional computer consultant for VLSI design analysis tools in IBM and network management in AT&T from 1995 to 1997. He is currently working in UTStarcom, Inc. as a computer specialist in the area of telecommunication. He is a member of the Laboratory for Logic, Databases, and Advanced Programming. His research interests include database system, logic programming and parallel processing.