Hard problems for simple logic programs

Yatin P. Saraiya *
Department of Computer Science
Stanford University

Abstract

A number of optimizations have been proposed for Datalog programs involving a single intensional predicate ("single-IDB programs") Examples include the detection of *commutativity* and *separability* ([Naug88],[RSUV89], [Ioan89a]) in linear logic programs, and the detection of *ZYT-linearizability* ([ZYT88], [RSUV89], [Sara89], [Sara90]) in nonlinear programs We show that the natural generalizations of the commutativity and ZYT-linearizability problems (respectively, the *sequencability* and *base-case linearizability* problems) are undecidable Our constructions involve the simulation of context-free grammars using single-IDB programs that have a bounded number of initialisation rules The constructions may be used to show that containment (or equivalence) is undecidable for such programs, even if the programs are linear, or if each program contains a single recursive rule These results tighten those of [Shmu87] and [Abit89]

*Work supported by NSF grant IST-84-12791, Air Force grant AFOSR-88-0266 and a grant of IBM Corp

1. Introduction

A deductive database consists of a set of ground atoms (comprising the *extensional* database or EDB) and a set of Horn clauses that define relations from the EDB The relations thus constructed are *intensional* relations, and occupy the intensional database (or IDB) The predicates that correspond to the EDB are extensional (or EDB) predicates, and those that correspond to the IDB are intensional (or IDB) predicates We will ignore the distinction between relations and predicates, and assume that no predicate is both intensional and extensional A set of rules is termed a *program*, and we assume that the rules are safe and function-free (i e we consider safe, Datalog programs)

The simplest recursive programs are those that involve a single intensional predicate ("single-IDB programs") A variety of powerful optimizations ([Naug88],[RSUV89],[ZYT88], [Sara89]) may be performed on such programs, often by using *proof-tree transformation techniques* ([RSUV89]) We illustrate two such optimizations in the following examples

Example 1. Consider the following linear logic program $\mathcal{P}$

$$\begin{array}{lll} r_1 & p(X,Y) & - p(X,B), a(B,Y) \\ r_2 & p(X,Y) & - p(U,V), a(V,Y), b(U,X) \\ b_1 & p(X,Y) & - c(X,Y) \end{array}$$

where a, b and c are EDB predicates It turns out that $\mathcal{P}$ is equivalent to the following program $\mathcal{Q}$ with respect to the predicate p, that is, both programs compute the same relation for p from every extensional database

$$\begin{array}{lll} r_1' & p(X,Y) & - p(X,B), a(B,Y) \\ s_1 & p(X,Y) & - q(X,Y) \\ r_2' & q(X,Y) & - q(U,V), a(V,Y), b(U,X) \\ s_2 & q(X,Y) & - c(X,Y)\ , \end{array}$$

The latter program can be thought of as representing a restricted bottom-up evaluation of the former, in which p is initialised using the basis rule b_1, then closed under r_2, and then closed under r_1 An alternative description

is that $\mathcal{Q}$ generates those facts that are generated by $\mathcal{P}$, using derivation trees of the form that is suggested in Figure 1 Hence, we say that the program $\mathcal{P}$ is *sequencable* Sequencability is formally defined in section 2

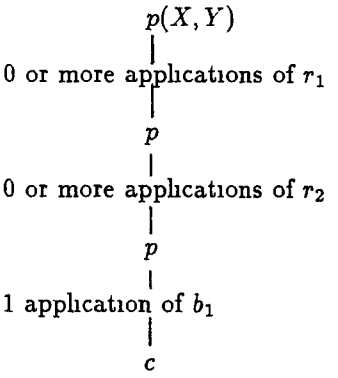


Figure 1

The conversion of $\mathcal{P}$ into $\mathcal{Q}$ in this way reduces the complexity of "counting" methods for query evaluation (such as that of [BMSU86]) from exponential to polynomial (see [RSUV89] for a proof) In addition, the transformation permits of a pipelined evaluation of the program

Sequencability has only been studied for linear logic programs, and no decision procedure is known even for such programs The focus has been on conditions that are sufficient (but not necessary) to detect sequencability The most common condition is "commutativity" among the linear rules, this condition may be treated as the containment of one *conjunctive query* ([CM77]) in another, or as the containment of a conjunctive query in a logic program ([RSUV89]) It is possible to show that the linear rules r_1 and r_2 in Example 1 commute (see [Sara90]), which justifies the transformation of $\mathcal{P}$ to $\mathcal{Q}$ Rule commutativity is also integral to the "separability" ([Naug88]) of linear programs

Example 2. Consider the following nonlinear program $\mathcal{P}$, which computes the transitive closure of the basis predicate b

$$\begin{array}{lll} r_1 & p(X,Y) & - p(X,Z), p(Z,Y) \\ b_1 & p(X,Y) & - b(X,Y) \end{array}$$

It is a well-known fact that we may replace $\mathcal{P}$ by the following linear logic program $\mathcal{Q}$, to obtain an equivalent program

$$\begin{array}{lll} r_1' & p(X,Y) & - b(X,Z), p(Z,Y) \\ b_1 & p(X,Y) & - b(X,Y) \end{array}$$

The linear program $\mathcal{Q}$ differs from $\mathcal{P}$ only in that the first recursive occurrence of p in the body of the recursive rule r_1 has been replaced by a corresponding occurrence of the basis predicate, b The program $\mathcal{Q}$ can be thought of as representing a restricted top-down evaluation of $\mathcal{P}$, in which the first recursive occurrence of p is never recursively expanded That is, $\mathcal{Q}$ generates those facts that would be generated by $\mathcal{P}$, using derivation trees of the form shown in Figure 2 If $\mathcal{P} \equiv \mathcal{Q}$, then we say that $\mathcal{P}$ is *base-case linearizable*, a precise definition of base-case linearizability is presented in section 2 □

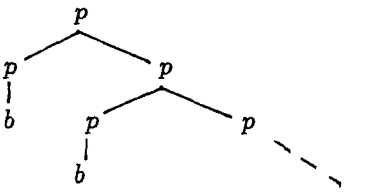


Figure 2

The advantage in transforming $\mathcal{P}$ to $\mathcal{Q}$ is that it permits the use of special query evaluators that are specific to linear programs In fact, the use of magic sets ([BMSU86], [BR87]) to evaluate the query $p(a, X)$ with respect to $\mathcal{P}$ takes quadratic time in the size of b, however, right-linear evaluation ([Ullm89]) evaluates the same query, with respect to $\mathcal{Q}$, in linear time

Base-case linearizability has only been studied for programs with a single recursive rule and a single basis rule, in which case it is called *ZYT-linearizability* ([RSUV89], [ZYT88], [Ioan89b], [Sara89], [Sara90]) In fact, $\mathcal{P}$ (in this example) is ZYT-linearizable [Sara89] describes the largest class of programs for which ZYT-linearizability is known to be decidable [RSUV89] shows that the decision procedure of [Sara89] does not extend naturally, and provides a powerful condition that is sufficient (but not necessary) for ZYT-linearizability in general [Sara90] contains further intractability results

1.1 Motivation

The detection of commutativity, separability and ZYT-linearizability concerns the detection of equivalence between safe, Datalog programs which

1 involve a single intensional predicate (p in Examples 1 and 2), and in which

2 the rule heads are *rectified* (i e , no variable is repeated in the head of any rule)

As we have discussed, such programs allow of powerful optimizations We consider, in this paper, the general problem of deciding containment (and equivalence) among such programs, and of deciding sequencability or base-case linearizability in such programs

Shmueli ([Shmu87]) and Abiteboul ([Abit89]) present general results that also yield undecidability results for program equivalence Shmueli considers programs with a single recursive predicate, however, these programs have several IDB predicates, and include rules whose

heads are not rectified Head-rectification occasionally enforces tractability, as in [Sara89] Abiteboul's result concerns single-IDB programs with a single recursive rule, however, those programs have rules that are not head-rectified, and contain an unbounded number of initialisation rules

We show that containment and equivalence are undecidable for safe, single-IDB, Datalog programs with head-rectified rules and a bounded number of initialisation rules, even if (a) the programs are linear, or, (b) if each program contains a single recursive rule Our construction involves the simulation of context-free grammars by such programs

1.2 Outline

In Section 2, we provide formal definitions of base-case linearizability and sequencability Section 3 contains a statement of our results In Section 4, we provide some preliminary results that will be of use in the results of the following sections In Section 5, we provide a construction whereby linear context-free grammars may be simulated by linear single-IDB programs, and prove the undecidability of base-case linearizability In Section 6, we show how Chomsky Normal Form grammars may be simulated by single-IDB programs with a single recursive rule and a bounded number of basis rules, and prove that sequencability is undecidable

2. Definitions

In this section, we present formal definitions of sequencability and base-case linearizability $\mathcal{P}$ is assumed to be a safe, Datalog program, in which the head of every rule is rectified (i e , contains no repetitions of any variable)

Let $\mathcal{P}$ consist of the n recursive rules

$$r_1 \quad p(\vec{X}_0) \quad - p(\vec{X}_{11}), \quad p(\vec{X}_{1k_1}), \mathcal{C}_1$$

$$r_i \quad p(\vec{X}_0) \quad - p(\vec{X}_{i1}), \quad p(\vec{X}_{ik_i}), \mathcal{C}_i$$

$$r_n \quad p(\vec{X}_0) \quad - p(\vec{X}_{n1}), \quad p(\vec{X}_{nk_n}), \mathcal{C}_n$$

and the m nonrecursive rules

$$b_1 \quad p(\vec{X}_0) \quad - \mathcal{D}_1$$

$$b_j \quad p(\vec{X}_0) \quad - \mathcal{D}_j$$

$$b_m \quad p(\vec{X}_0) \quad - \mathcal{D}_m$$

where the $\mathcal{C}_i$ and $\mathcal{D}_j$ are arbitrary conjunctions of EDB predicates Example 1 exhibits such a program, with $n = 2$ and $m = 1$, and Example 2 contains a program in which $n = 1$ and $m = 1$

Base-case linearizability

Let q be a new predicate symbol Construct the program $\mathcal{Q}$, with $n + m + 1$ rules $\{r'_i \mid 1 \leq i \leq n\} \cup \{b'_i \mid 1 \leq i \leq m\} \cup \{c\}$, as follows If r_i is a nonlinear rule ($k_i > 1$), then replace r_i by the linear rule

$$r'_i \quad p(\vec{X}_0) \quad - q(\vec{X}_{i1}), \quad , q(\vec{X}_{ik_i-1}), p(\vec{X}_{ik_i}), \mathcal{C}_i$$

That is, we replace all but the last recursive occurrence of p in r_i with a corresponding occurrence of q If r_i is linear ($k_i = 1$), then r'_i is the same as r_i

$$r'_i \quad p(\vec{X}_0) \quad - p(\vec{X}_{i1}), \mathcal{C}_i$$

Next, we introduce the rule c, which merely initialises p to q

$$c \quad p(\vec{X}_0) \quad - q(\vec{X}_0)$$

Finally, each nonrecursive rule b_i is replaced by the rule

$$b'_i \quad q(\vec{X}_0) \quad - \mathcal{D}_i$$

which initialises q using the nonrecursive rules for p We say that $\mathcal{P}$ is *base-case linearizable* iff $\mathcal{P} \equiv \mathcal{Q}$ with respect to p The idea is that $\mathcal{Q}$ produces those facts that would be produced by derivation trees of $\mathcal{P}$ in which only the rightmost occurrence of p in any rule is ever recursively expanded (see Figure 2)

In Example 2, the program $\mathcal{Q}$ is the program

$$\begin{array}{lll} r'_1 & p(X,Y) & - q(X,Z), p(Z,Y) \\ c & p(X,Y) & - q(X,Y) \\ b'_1 & q(X,Y) & - b(X,Y) \end{array}$$

The intermediate predicate q can, in this case, be eliminated to obtain the linear program in Example 2

Sequencability

Now, let $q_1 \quad q_n$ be new and distinct predicate symbols, and construct the program $\mathcal{R}$ from program $\mathcal{P}$, as follows First, replace the rule r_1 by the two rules r'_1 and s_1, where r'_1 is the same as r_1

$$\begin{array}{lll} r'_1 & p(\vec{X}_0) & - p(\vec{X}_{11}), \quad p(\vec{X}_{1k_1}), \mathcal{C}_1 \\ s_1 & p(\vec{X}_0) & - q_1(\vec{X}_0) \end{array}$$

Next, replace each recursive rule r_i ($i > 1$) by the two rules

$$\begin{array}{lll} r'_i & q_{i-1}(\vec{X}_0) & - q_{i-1}(\vec{X}_{i1}), \quad q_{i-1}(\vec{X}_{ik_i}), \mathcal{C}_i \\ s_i & q_{i-1}(\vec{X}_0) & - q_i(\vec{X}_0) \end{array}$$

Finally, replace each nonrecursive rule b_i by the rule

$$b'_i \quad q_n(\vec{X}_0) \quad - \mathcal{D}_i$$

The idea is that $\mathcal{R}$ generates those facts that are produced by derivation trees of $\mathcal{P}$ in which the rule r_i is never used to expand a subgoal introduced by r_j, if $i < j$ That is, $\mathcal{R}$ computes those facts which would be produced by a bottom-up evaluation of $\mathcal{P}$, in which we initialise p using the b_i, and then, in sequence, close under $r_n, r_{n-1}, \ldots, r_1$ The program $\mathcal{P}$ is called *sequencable* iff it is equivalent to $\mathcal{R}$ with respect to p

In Example 1, the program $\mathcal{R}$ is the program

$$\begin{array}{lll} r_1' & p(X,Y) & - p(X,B), a(B,Y) \\ s_1 & p(X,Y) & - q_1(X,Y) \\ r_2' & q_1(X,Y) & - q_1(U,V), a(V,Y), b(U,X) \\ s_2 & q_1(X,Y) & - q_2(X,Y) \\ b_1' & q_2(X,Y) & - c(X,Y) \end{array}$$

which can easily be transformed to the sequenced program in the example, by composing s_2 and b_1'

3. Statement of results.

We state below the main results of this paper In the following statements, $\mathcal{P}$ and $\mathcal{Q}$ are safe, Datalog programs defining a single intensional predicate, using head-rectified rules

Result 1. $\mathcal{P} \subset \mathcal{Q}$ ($\mathcal{P} \equiv \mathcal{Q}$) is undecidable, even if (a) $\mathcal{P}$ and $\mathcal{Q}$ are linear, and have no more than five basis rules, (b) each of $\mathcal{P}$ and $\mathcal{Q}$ contains only one recursive rule and nine nonrecursive rules □

Result 2. The base-case linearizability of $\mathcal{P}$ is undecidable, even if $\mathcal{P}$ contains only one nonlinear rule □

Result 3. The sequencability of $\mathcal{P}$ is undecidable, even if $\mathcal{P}$ contains only two recursive rules □

4. Background

In this section, we present results that will be of use in the proofs of the next two sections

4.1 Datalog programs

Our treatment will be concise, for a detailed treatment, see [Ullm89] Let $\mathcal{P}$ and $\mathcal{Q}$ be safe, Datalog programs defining the single intensional predicate p Each top-down expansion of p, using the rules in $\mathcal{P}$, is a *conjunctive query* ([CM77]) For example, the program of Example 2 generates the conjunctive query shown in Figure 3

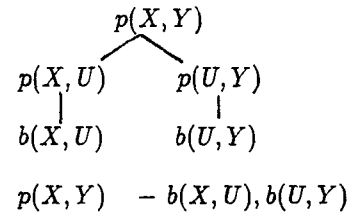


Figure 3 Top-down expansions as conjunctive queries

Hence, we may think of a Datalog program as the infinite union of the conjunctive queries that it generates By a theorem of [SY81], we conclude that $\mathcal{P} \subset \mathcal{Q}$ iff for every conjunctive query $C_{\mathcal{P}}$ generated by $\mathcal{P}$, there is a conjunctive query $C_{\mathcal{Q}}$ generated by $\mathcal{Q}$ such that $C_{\mathcal{P}} \subset C_{\mathcal{Q}}$ Using this result, we may see that the sequencability of $\mathcal{P}$ in Example 1 means that every conjunctive query generated by $\mathcal{P}$ is contained in a conjunctive query of the form that is illustrated in Figure 1 Similarly, the base-case linearizability of $\mathcal{P}$ in Example 2 may be visualized as the containment of every conjunctive query generated by $\mathcal{P}$ in a conjunctive query of the form illustrated in Figure 2

It is known ([CM77]) that for conjunctive queries C and D, $C \subset D$ iff there is a *containment mapping* from D into C (see [Ullm89] for a description of containment mappings)

Lemma 1. Let C and D be the conjunctive queries

$$\begin{array}{lll} C & p(\vec{X}) & - \mathcal{C} \\ D & p(\vec{X}) & - \mathcal{D}, f(Z) \end{array}$$

where $\mathcal{C}$ and $\mathcal{D}$ are conjunctions of EDB predicates, Z is a distinguished variable (i e Z appears in $\vec{X}$) and $f(Z)$ does not appear in $\mathcal{C}$ Then $C \not\subset D$

Proof There is no containment mapping from D into C □

Let $p(\vec{X}) \quad - \mathcal{C}$ be a (not necessarily safe) conjunctive query (i e , there may be variables in $\vec{X}$ that do not appear among the conjuncts in $\mathcal{C}$), and let e be a new, unary predicate Then the notation $e(\mathcal{ALL})$ denotes the conjunction of atoms $e(A)$, for every variable A that appears among $\vec{X}$, or appears as an argument to one of the conjuncts in $\mathcal{C}$ That is, the notation $p(X,Y) \quad - b(X,U), e(\mathcal{ALL})$ denotes the conjunctive query $p(X,Y) \quad - b(X,U), e(X), e(Y), e(U)$

Lemma 2. Let C and D be the conjunctive queries

$$\begin{array}{ll} C & p(R,X,Y,W,Z,S,N_1, \ldots, N_m) \quad - a_1(X,U_1), \\ & a_2(U_1,U_2), \ldots, a_k(U_{k-1},Y), f(S), f(W), e(\mathcal{ALL}) \\ D & p(R,X,Y,W,Z,S,N_1, \ldots, N_m) \quad - b_1(X,V_1), \\ & b_2(V_1,V_2), \ldots, b_l(V_{l-1},Y), f(S), f(W), e(\mathcal{ALL}) \end{array}$$

where the U_i and V_j are distinct nondistinguished variables (i e they do not appear among the variables in

the head of the query) Then, $C \subset D$ iff the strings $a_1 a_2 \quad a_k$ and $b_1 b_2 \quad b_l$ are identical

Proof sketch If the strings are identical, then the identity mapping on distinguished variables, and the mapping $h(V_i) = U_i$, is a containment mapping from D into C If $C \subset D$, then there is a containment mapping from D into C, a simple induction on l suffices to complete the proof, using the properties of a containment mapping □

The conjunction $a_1(X, U_1) \quad a_k(U_{k-1}, Y)$ is a *binary chain from X to Y, embedded in C*, and *representing* the string $a_1 \quad a_k$ The distinguished variable R will be of use in the proof of Theorem 3, in Section 5

4.2 Context-free grammars

The results of the following sections will be based upon reductions from undecidable problems in language theory In this subsection, we establish some preliminary results Our treatment is concise, and assumes concepts that are explained in [HU79]

Lemma 3. Assume $\Sigma = \{a, b\}$ It is undecidable, for an arbitrary context-free grammar (CFG) G over the alphabet Σ, whether $\Sigma^* \subset L(G)$ This result is true even if G is linear

Proof Straightforward reductions from the corresponding problems with arbitrary Σ([HU79])

Corollary 1 For any $k \geq 0$, $\Sigma^k \Sigma^+ \subset L(G)$ is undecidable

Proof We may write Σ^* as $F \cup \Sigma^k \Sigma^+$, where F is the finite set $\{\epsilon\} \cup \Sigma \cup \quad \cup \Sigma^k$ Our result follows because $F \subset L(G)$ is decidable

Corollary 2 $\Sigma^+ \subset L(G) - \{\epsilon\}$ is undecidable

Proof Since $\epsilon \notin \Sigma^+$, $\Sigma^+ \subset L(G)$ iff $\Sigma^+ \subset L(G) - \{\epsilon\}$, the former is undecidable, by Corollary 1

Corollary 3 $\Sigma^+ \subset L(G) - \{\epsilon\} \cup \Sigma$ is undecidable

Proof $\Sigma \subset L(G)$ is decidable □

Lemma 4. Let G be a linear CFG over the terminal alphabet Σ, such that $\epsilon \notin L(G)$ Then, $\Sigma^+ L(G) \subset \Sigma L(G)$ is undecidable

Proof We claim that $\Sigma^+ \subset L(G)$ iff

1 $\Sigma \subset L(G)$ and

2 $\Sigma^+ L(G) \subset \Sigma L(G)$

Since 1 is decidable, our result then follows by Corollary 1 to Lemma 3

Assume that $\Sigma^+ \subset L(G)$ Then, since (by definition) $L(G) \subset \Sigma^+$, conditions (1) and (2) are trivially true Now, assume $w \in \Sigma^+$ We use 1 and 2 to show that $w \in L(G)$ There are two cases

Case a $|w| = 1$ Then $w \in \Sigma$, and our result follows by (1)

Case b $|w| > 1$ Then $w = sa$, for some $s \in \Sigma^+$ and $a \in \Sigma$ Then, $as \in \Sigma^+$ and, by (1), $a \in L(G)$ Hence, $asa \in \Sigma^+ L(G)$, so by (2), $sa = w \in L(G)$ □

Now, we introduce a normal form for linear grammars, by introducing *unit productions* into the grammar Let a *modified linear grammar* over $\Sigma = \{a_1, a_2\}$ be a grammar in which

1 There is a unit production $S \rightarrow N_1$, where S is the start symbol of the grammar and S appears in no other production This production is the *start production* of the grammar

2 Every other production is of the form

a $N_i \rightarrow a_k N_j$, where a_k is a terminal and N_j is a nonterminal

b $N_i \rightarrow N_j a_k$, where a_k is a terminal and N_j is a nonterminal

c $N_i \rightarrow a_k$, where a_k is a terminal There are only two such productions (for $k = 1$ and $k = 2$), and in this case, N_i appears on the left-hand side of no other production

d $N_i \rightarrow N_j$, a unit production

Note that, if we chose N_1 as the start symbol, the language generated would not change

Lemma 5. For every linear grammar H over $\Sigma = \{a_1, a_2\}$, there is a modified linear grammar G such that $L(G) = L(H) - \{\epsilon\}$

Proof The transformations of [HU79] (Section 4 4) can be used to construct a linear grammar I, with no useless symbols, unit productions or ϵ-productions, for $L(H) - \{\epsilon\}$ (these transformations are easily seen to preserve linearity) Assume that A is the start symbol of I We construct G from I, as follows Every production of I is of the form $N \rightarrow s$, $N \rightarrow wMs$, $N \rightarrow wM$ or $N \rightarrow Ms$ for nonterminals N and M, and nonempty terminal strings s and w The introduction of new nonterminals, and the left- or right-factoring of these productions, can be used to convert these productions into productions of type 2(a) 2(c) in the definition of modified-linear grammars, without affecting the language recognized Then, we introduce new nonterminals B_1 and B_2, add productions $B_1 \rightarrow a_1$ and $B_2 \rightarrow a_2$, and convert every production $N_i \rightarrow a_j$ to the production $N_i \rightarrow B_j$ Finally, we add the production $S \rightarrow A$ to complete the transformation □

Finally, we introduce a modification of Chomsky Normal Form grammars ([HU79]) A *Modified Chomsky Normal Form (MCNF)* grammar over the terminal alphabet $\Sigma = \{a, b\}$ is a grammar with the following properties

1 The start symbol S appears on the right-hand side of no production

2 All productions are of the form $N \rightarrow c$, $N \rightarrow M$ or $N \rightarrow MK$, where c is a terminal The first of these is *nonrecursive*, the others are *recursive*

3 No nonterminal N appears on the left-hand side of more than two productions, further, if it does appear in two productions, then both productions are unit productions

Hence, we can classify recursive productions as follows

1 If $N \rightarrow M$ and $N \rightarrow K$ are productions, then these productions are *or-productions*
2 $N \rightarrow MK$ is an *and-production*
3 If N appears only on the left-hand side of the production $N \rightarrow M$, then this is a *copy production*

Lemma 6. For every grammar H, there is an MCNF grammar G generating $L(H) - \{\epsilon\} \cup \Sigma$
Proof Construct a Chomsky Normal Form grammar for $L(H) - \epsilon$, with start symbol T Add the productions $S \rightarrow T$, $T \rightarrow a$ and $T \rightarrow b$, to obtain a grammar I for $L(H) - \{\epsilon\} \cup \Sigma$ Next, introduce the nonterminals N_a and N_b, and the productions $N_a \rightarrow a$ and $N_b \rightarrow b$ Then, replace every other production of the form $N \rightarrow c$, where c is a terminal, by the production $N \rightarrow N_c$ All productions other than $N_a \rightarrow a$ and $N_b \rightarrow b$ are now recursive Replace every and-production $N \rightarrow MK$ by the two productions $N \rightarrow L$ and $L \rightarrow MK$, where L is a new nonterminal At this point, the only violations of MCNF are the presence of nonterminals N such that $N \rightarrow R_1, \quad , N \rightarrow R_{k+1}$, for $k > 1$, are the productions with N on the left-hand side Introduce new nonterminals $M_1, \quad , M_k$, then, add the productions $N \rightarrow M_1$ and $M_k \rightarrow R_{k+1}$, finally, for $2 \leq i \leq k$, replace the production $N \rightarrow R_i$ with the two productions $M_{i-1} \rightarrow R_i$ and $M_{i-1} \rightarrow M_i$ □

5. Linear logic programs.

We begin by proving Result 1(a), repeated below

Result 1a. Let $\mathcal{P}$ and $\mathcal{Q}$ be linear, head-rectified, single-IDB programs with at most five initialisation rules Then $\mathcal{P} \subset \mathcal{Q}$ is undecidable
Corollary $\mathcal{P} \equiv \mathcal{Q}$ is undecidable □

Given a modified linear grammar G over the terminal alphabet $\Sigma = \{a_1, a_2\}$, we construct linear programs $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathcal{P} \subset \mathcal{Q}$ iff $\Sigma^+ \subset L(G)$ Our result then follows by Lemma 5 and Corollary 2 to Lemma 3

Let us assume that the nonterminals of the grammar $S, N_1, \quad N_m$, where S is the start symbol of G The programs $\mathcal{P}$ and $\mathcal{Q}$ define the predicate p, with arity $m + 6$ The head of each rule is $p(R, X, Y, W, Z, S, N_1, \quad N_m)$, where the variables in the head are distinct

We will construct $\mathcal{P}$ so that the conjunctive queries that it creates contain binary chains representing the strings in Σ^+ For every $a_i \subset \Sigma$, $\mathcal{P}$ contains the rules

r_i $p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; - a_i(X, U),$
$p(Z, U, Y, W, Z, S, Z, \quad , Z), f(W), f(S), e(\mathcal{ALL})$
b_i $p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; - a_i(X, Y),$
$f(W), f(S), e(\mathcal{ALL})$

The distinguished variable R is relevant only in the proof of Theorem 3, at the end of this section The following lemmas are easily seen to be true of $\mathcal{P}$ The proofs are inductions on the length of derivations in G, or the depth of top-down expansions in $\mathcal{P}$

Lemma 7. Let $a_{i_1} a_{i_2} \quad a_{i_k} \in \Sigma^+$ Then $\mathcal{P}$ generates a conjunctive query of the following form (in which repeated conjuncts have been eliminated)
$p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; - a_{i_1}(X, U_1),$
$a_{i_2}(U_1, U_2), \quad , a_{i_k}(U_{k-1}, Y), f(S), f(W), e(\mathcal{ALL})$ □

Lemma 8. Let $\mathcal{C}$ be a conjunctive query generated by $\mathcal{P}$ Then $\mathcal{C}$ is of the form

$p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; - a_{i_1}(X, U_1),$
$a_{i_2}(U_1, U_2), \quad , a_{i_k}(U_{k-1}, Y), f(S), f(W), e(\mathcal{ALL})$

for some string $a_{i_1} \quad a_{i_k} \in \Sigma^+$ □

Let us now construct $\mathcal{Q}$ In the following, the notation $< N_i >$ denotes an m-vector in which the ith component is W, and all other components are Z Thus, the notation $p(R, X, Y, W, Z, S, < N_1 >)$ denotes the atom $p(R, X, Y, W, Z, S, W, Z, \quad , Z)$
$\mathcal{Q}$ has a rule for each production in G The production is one of four types (since G is modified-linear)

Case 1 It is the unit production $J \rightarrow N_j$ (where J is the start symbol S or a nonterminal N_i) We construct the rule

$p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; -$
$p(Z, X, Y, W, Z, Z, < N_j >), f(J), f(W), e(\mathcal{ALL})$

The recursive atom replaces N_j by W, and all other nonterminals by Z Further, f is made true of J (i e , S or N_i) and W If J is S, then this rule is the *start rule*

Case 2 $N_i \rightarrow a_k N_j$ We produce the rule

$p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; - a_k(X, U),$
$p(Z, U, Y, W, Z, Z, < N_j >), f(N_i), f(W), e(\mathcal{ALL})$

Case 3 $N_i \rightarrow N_j a_k$ We produce the rule

$p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; -$
$p(Z, X, U, W, Z, Z, < N_j >), a_k(U, Y),$
$f(N_i), f(W), e(\mathcal{ALL})$

Case 4 $N_i \rightarrow a_k$ We construct the rule

$p(R, X, Y, W, Z, S, N_1, \quad , N_m) \; - a_k(X, Y),$
$f(N_i), f(W), e(\mathcal{ALL})$

The intention is that these rules mimic derivations in G, to produce binary chains to represent every string in $L(G)$ The idea is illustrated in Figure 4 In the figure, the variable A may be a new nondistinguished variable, or the distinguished variable X Similarly, B may be a nondistinguished variable, or the distinguished variable Y U is a new nondistinguished variable

However, these rules also may be used to mimic "illegal" derivations in the grammar That is, the production $N_i \rightarrow a_k N_j$ cannot be used to expand the nonterminal N_l, if $N_i \neq N_l$ However, the rule (in $\mathcal{Q}$) that corresponds to the production $N_i \rightarrow a_k N_j$ can, in fact, be used to expand a p-atom resulting from the application of the rule for $N_m \rightarrow a_o N_l$ We detect such illegal top-down expansions through the use of the conjuncts $f(N_i)$ in the rules of $\mathcal{Q}$ A conjunctive query resulting from an illegal expansion as described above will contain the atom $f(Z)$, further, if the first rule applied is not the start rule, then the conjunctive query that is generated will contain the conjunct $f(N_i)$, for some i Hence, for the purpose of the containment $\mathcal{P} \subset \mathcal{Q}$, we may ignore these illegal top-down expansions in $\mathcal{Q}$, since the only f-atoms generated by $\mathcal{P}$ are $f(W)$ and $f(S)$

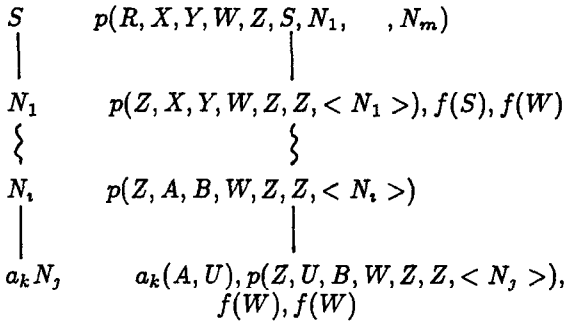


Figure 4

More formally, we say that a conjunctive query generated by $\mathcal{Q}$ is *illegal* if it contains the conjunct $f(A)$, where A is one of the distinguished variables $Z, N_1, N_2, \ldots N_m$ Further, let us define *legal($\mathcal{Q}$)* to be the union of all the conjunctive queries generated by $\mathcal{Q}$ that are not illegal

Lemma 9. $\mathcal{P} \subset \mathcal{Q}$ iff $\mathcal{P} \subset$ *legal($\mathcal{Q}$)*
Proof Let $\mathcal{C}$ be a conjunctive query generated by $\mathcal{P}$, and $\mathcal{D}$ an illegal conjunctive query generated by $\mathcal{Q}$ Then, $\mathcal{D}$ contains, as a conjunct, an atom $f(U)$, where U is one of the distinguished variables $Z, N_1, \ldots, N_m$ By Lemma 7, $f(U)$ does not appear in $\mathcal{C}$, hence by Lemma 1, $\mathcal{C} \not\subset \mathcal{D}$ □

Lemma 10. Let $a_{i_1} a_{i_2} \ldots a_{i_k} \in L(G)$ Then *legal($\mathcal{Q}$)* contains a conjunctive query of the following form (in which repeated conjuncts have been eliminated)

$p(R, X, Y, W, Z, S, N_1, \ldots, N_m) \; - \; a_{i_1}(X, U_1),$
$a_{i_2}(U_1, U_2), \ldots, a_{i_k}(U_{k-1}, Y), f(S), f(W), e(\mathcal{ALL})$
Proof By induction on the size of the sentential forms in G □

Lemma 11. Let $\mathcal{C}$ be a conjunctive query in *legal($\mathcal{Q}$)* Then $\mathcal{C}$ is of the form

$p(R, X, Y, W, Z, S, N_1, \ldots, N_m) \; - \; a_{i_1}(X, U_1),$
$a_{i_2}(U_1, U_2), \ldots, a_{i_k}(U_{k-1}, Y), f(S), f(W), e(\mathcal{ALL})$

for some string $a_{i_1} \ldots a_{i_k} \in \Sigma^+$
Proof By induction on the depth of the top-down expansions of $\mathcal{Q}$ □

Theorem 1. $\mathcal{P} \subset \mathcal{Q}$ iff $\Sigma^+ \subset L(G)$
Proof By Lemma 9, it suffices to show that $\mathcal{P} \subset$ *legal($\mathcal{Q}$)* iff $\Sigma^+ \subset L(G)$ The "if" direction follows by Lemmas 8, 10 and 2, and the "only if" direction follows by lemmas 7, 11, 2 and 10 □

Example 3.

The construction, and these lemmas, may be made clearer with an example Consider the modified linear grammar G, as described below

$S \rightarrow A \qquad A \rightarrow aA \qquad A \rightarrow bB$
$B \rightarrow b \qquad C \rightarrow a$

It is easily seen that $L(G)$ is a^*bb The program $\mathcal{P}$ is

$p(R, X, Y, W, Z, S, A, B, C) \; - \; a(X, U),$
$p(Z, U, Y, W, Z, S, Z, Z, Z), f(S), f(W), e(\mathcal{ALL})$

$p(R, X, Y, W, Z, S, A, B, C) \; - \; b(X, U),$
$p(Z, U, Y, W, Z, S, Z, Z, Z), f(S), f(W), e(\mathcal{ALL})$

$p(R, X, Y, W, Z, S, A, B, C) \; - \; a(X, Y),$
$f(S), f(W), e(\mathcal{ALL})$

$p(R, X, Y, W, Z, S, A, B, C) \; - \; b(X, Y),$
$f(S), f(W), e(\mathcal{ALL})$

and the program $\mathcal{Q}$ is

$p(R, X, Y, W, Z, S, A, B, C) \; -$
$p(Z, X, Y, W, Z, Z, W, Z, Z), f(S), f(W), e(\mathcal{ALL})$

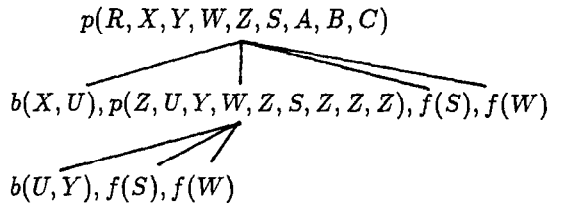
$p(R, X, Y, W, Z, S, A, B, C) \; - a(X, U),$
$p(Z, U, Y, W, Z, Z, W, Z, Z), f(A), f(W), e(\mathcal{ALL})$

$p(R, X, Y, W, Z, S, A, B, C) \; - \; b(X, U),$
$p(Z, U, Y, W, Z, Z, Z, W, Z), f(A), f(W), e(\mathcal{ALL})$

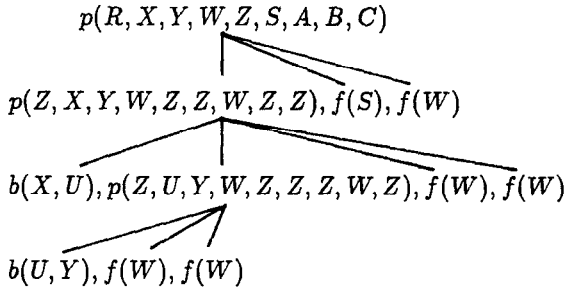
$p(R, X, Y, W, Z, S, A, B, C) \; - \; b(X, Y),$
$f(B), f(W), e(\mathcal{ALL})$

$p(R, X, Y, W, Z, S, A, B, C) \; - \; a(X, Y),$
$f(C), f(W), e(\mathcal{ALL})$

Figure 5 describes the representation of the string bb by $\mathcal{P}$ and $\mathcal{Q}$ (where we have ignored $e(\mathcal{ALL})$)



Representation of bb by $\mathcal{P}$



Representation of bb by $\mathcal{Q}$

Figure 5

However, it is easily seen that the conjunctive query

$$p(R,X,Y,W,Z,S,A,B,C) \quad - \\ a(X,Y), f(W), f(S), e(\mathcal{ALL})$$

is generated by $\mathcal{P}$, but the smallest conjunctive query generated by $\mathcal{Q}$, in which there are no appearances of $f(Z)$, $f(A)$, $f(B)$ or $f(C)$ is

$$p(R,X,Y,W,Z,S,A,B) \quad - \\ b(X,U), b(U,Y), f(S), f(W), e(\mathcal{ALL})$$

Hence, $\mathcal{P} \not\subset \mathcal{Q}$, as desired

Finally, we outline constructions that prove that program equivalence is undecidable for linear programs, and that base-case linearizability is undecidable The constructions are based on the emulation of Σ^+ and $L(G)$ by the programs $\mathcal{P}$ and $\mathcal{Q}$, as described above

Theorem 2. Let $\mathcal{R}$ and $\mathcal{S}$ be linear, single-IDB programs Then, $\mathcal{R} \equiv \mathcal{S}$ is undecidable

Proof sketch We construct $\mathcal{R}$ and $\mathcal{S}$ from $\mathcal{P}$ and $\mathcal{Q}$ of the preceding proof, such that $\mathcal{R} \equiv \mathcal{S}$ iff $\Sigma^+ \subset L(G)$ Let b be the rule

$$b \quad p(R,X,Y,W,Z,S,N_1, \quad ,N_m) \quad - f(Z), e(\mathcal{ALL})$$

Then, $\mathcal{R}$ consists of all the rules in $\mathcal{P}$, every rule in $\mathcal{Q}$ except for the start rule, and the rule b, as described above $\mathcal{S}$ consists of all the rules in $\mathcal{Q}$ (including the start rule), and the rule b □

Theorem 3. Let $\mathcal{T}$ be a single-IDB program Then, the base-case linearizability of $\mathcal{T}$ is undecidable

Proof sketch We construct a nonlinear program $\mathcal{T}$ from the programs $\mathcal{P}$ and $\mathcal{Q}$, as described in the proof of Theorem 1 The construction will yield a program that is base-case linearizable iff $\Sigma^+ L(G) \subset \Sigma L(G)$, and our result follows by Lemma 4 Let $\mathcal{B}$ consist of the nonrecursive rules

$$p(R,X,Y,W,Z,S,N_1, \quad ,N_m) \quad - f(A), e(\mathcal{ALL})$$

for $A = Z, N_1, \quad ,N_m$

Then, $\mathcal{T}$ consists of all the rules in $\mathcal{P}$ and $\mathcal{B}$, and all the rules in $\mathcal{Q}$ except the start rule $\mathcal{T}$ contains one additional, nonlinear rule n

$$n \quad p(R,X,Y,W,Z,S,N_1, \quad ,N_m) \quad - \\ p(Z,X,U,W,Z,W,Z, \quad ,Z), \\ p(Z,U,Y,W,Z,Z,< N_1 >), f(R), e(\mathcal{ALL}) \quad \square$$

6 . Single-recursive-rule programs

In this section, we present a construction whereby an arbitrary Modified Chomsky Normal Form (MCNF) grammar may be simulated using a head-rectified, single-IDB program with a single recursive rule and a bounded number of basis rules The construction may be used to show that sequencability is undecidable, even for programs with only two recursive rules In addition, the construction can be used to prove the undecidability of equivalence (or containment) of programs with a single recursive rule

Let H be an MCNF grammar for the language Σ^+ over the grammar $\Sigma = \{a,b\}$, with start symbol N_1 and nonrecursive productions $N_a \rightarrow a$ and $N_b \rightarrow b$ Let I be an MCNF grammar over Σ, with start symbol N_2 and the same nonrecursive productions We assume that $L(I)$ is ϵ-free, and that $\Sigma \subset L(I)$

Let S be a new nonterminal Construct G as the union of H and I, with the additional productions $S \rightarrow N_1$ and $S \rightarrow N_2$, and let S be the start symbol of G It is easily seen that G is an MCNF grammar for $\Sigma^+ \cup L(I)$

Assume that the nonterminals of G are $S, N_1, \quad ,N_m, N_a, N_b$, where N_1, N_2, N_a and N_b are as described above We construct two programs $\mathcal{P}$ and $\mathcal{Q}$, such that $\mathcal{Q} \subset \mathcal{P}$, and such that $\mathcal{P} \subset \mathcal{Q}$ iff $\Sigma^+ \subset L(I)$ This construction suffices to prove Result 1(b)

The programs $\mathcal{P}$ and $\mathcal{Q}$ define the IDB predicate p, and the head of each rule is $p(R,X,Y,W,Z,G,A,B,S,N_1, \quad ,N_m,N_a,N_b)$ We describe the construction of $\mathcal{P}$, $\mathcal{Q}$ is then obtained through a slight modification of $\mathcal{P}$

The variables in the head have the following purposes

1 R is a switch that is relevant only to the proof that sequencability is undecidable The R-position in the arguments of each p-atom in the body of the recursive rule will be occupied by the variable Z, described below

2 X and Y are the end-points of binary chains representing strings in Σ^+ and $L(G)$, as in the preceding section

3 W and Z are guard variables, as in the preceding section They are used to weed out illegal conjunctive queries generated by the programs (i e , queries representing impossible derivations in the grammar)

4 G is a guard position Intuitively, a p-atom may be legally expanded through the recursive rule if its G-th argument is W, but not if its G-th argument is Z

5 A and B are used to allow a choice in expanding one of two or-productions, in a manner to be described

6 $S, N_1, \ , N_m, N_a, N_b$ represent the corresponding nonterminals in the grammar As in Section 5, the notation $< N_i >$ represents an $m+2$-vector in which the indicated component is W, and the others are Z

The rules of $\mathcal{P}$ and $\mathcal{Q}$ are as follows Each rule is made safe through the use of $e(\mathcal{ALL})$, as described in Section 5

$\mathcal{P}$ and $\mathcal{Q}$ have the following basis rules

$p(R, X, Y, W, Z, G, A, B, S, N_1, \ , N_m, N_a, N_b)$ -

i_1 $a(X,Y), f(N_a), f(G)$
i_2 $b(X,Y), f(N_b), f(G)$
i_3 $f(Z)$
i_4 $g(G)$
i_5 $g(W)$
i_6 $f(U), g(U)$
i_7 $h(A,B), f(G)$
i_8 $h(U,V), h(V,W)$
i_9 $h(U,U)$

The body of the recursive rule $r_{\mathcal{P}}$ for $\mathcal{P}$ has the atoms $f(W)$,$f(G)$, $g(Z)$, $g(N_a)$, $g(N_b)$ and $h(E,F)$, where E and F are nondistinguished variables that appear nowhere else in the program (except in $e(\mathcal{ALL})$) It also has p-atoms for each recursive production in the grammar, as follows

1 Let $J \rightarrow K$ be a copy production, and let U_J be a new nondistinguished variable Then, the recursive rule contains the atom $p(Z, X, Y, W, Z, J, U_J, U_J, Z, < K >)$

2 Let $J \rightarrow K$ and $J \rightarrow L$ be a pair of or-productions, and let T_J, U_J and V_J be new nondistinguished variables Then, the body of $r_{\mathcal{P}}$ contains the atoms $p(Z, X, Y, W, Z, J, T_J, U_J, Z, < K >)$ and $p(Z, X, Y, W, Z, J, U_J, V_J, Z, < L >)$ These two atoms are known as *or-atoms* The idea is that we need never recursively expand both these p-atoms, since either one (but not both) may be expanded using initialisation rule i_7

3 Let $J \rightarrow KL$ be an and-production, and let T_J, U_J and V_J be new nondistinguished variables $r_{\mathcal{P}}$ has the two p-atoms $p(Z, X, T_J, W, Z, J, U_J, U_J, Z, < K >)$ and $p(Z, T_J, Y, W, Z, J, V_J, V_J, Z, < L >)$ The idea is that both atoms are (recursively) expanded, to create binary chains from X to T_J, and from T_J to Y

The recursive rule $r_{\mathcal{Q}}$ in $\mathcal{Q}$ is identical to $r_{\mathcal{P}}$, except that the p-atom $p(Z, X, Y, W, Z, S, U_S, V_S, Z, < N_2 >)$ (representing the production $S \rightarrow N_2$) is replaced by the atom $p(Z, X, Y, W, Z, S, A_S, A_S, Z, < N_2 >)$, where A_S is a new nondistinguished variable

For either program, we say that a conjunctive query generated by the program is *illegal* if it is contained in one of the initialisation rules $i_1 \ \ i_9$, and *legal* otherwise Further, we say that a conjunctive query is *restricted* if no sibling or-atoms are recursively expanded

Lemma 12. For each of $\mathcal{P}$ and $\mathcal{Q}$, every legal conjunctive query is contained in a conjunctive query that is both legal and restricted

Proof idea If two sibling or-atoms are recursively expanded, we may initialise one atom (but not both) using basis rule i_7 □

For any nonterminal J in G, let *yield*(J) represent all the strings generated by J (i e , the strings that would be generated if J were the start symbol of G)

Lemma 13. Let $\mathcal{C}_{\mathcal{P}}$ be a restricted legal conjunctive query generated by $\mathcal{P}$ Then the body of $\mathcal{C}_{\mathcal{P}}$ consists of the atoms $f(W)$, $f(G)$,$g(Z)$, $g(N_a)$ and $g(N_b)$, atoms of the form $h(U,V)$ for nondistinguished variables U and V appearing nowhere else in the program, and

1 Either the atom $g(S)$, or the atom $f(S)$ and a binary chain representing a string $s \in \Sigma^+ \cup L(I)$, but not both

2 For $1 \leq i \leq m$, either $g(N_i)$, or $f(N_i)$ and a binary chain representing some string $t_i \in$ *yield*(N_i), but not both

Further, the converse is also true That is, given strings $s \in \Sigma^+ \cup L(I)$ and (for $1 \leq i \leq m$) $t_i \in$ *yield*(N_i), $\mathcal{P}$ generates all the restricted legal conjunctive queries that may be obtained by applying (1) and (2) above

Finally, the two statements above are also true for $\mathcal{Q}$, except that "$\Sigma^+ \cup L(I)$" is replaced by "$L(I)$" □

Theorem 4. Let $\mathcal{P}$ and $\mathcal{Q}$ be safe, single-IDB programs with a single recursive rule and nine initialisation rules Then, the containment or equivalence of such programs is undecidable

Proof sketch By Lemmas 6, 12 and 13, and Corollary 3 to Lemma 3 □

Theorem 5. The sequencability of single-IDB programs is undecidable

Proof Given an MCNF grammar I, we construct a program $\mathcal{R}$ with basis rules $i_1 \ \ i_9$ and two recursive rules r_1 and r_2, such that r_2 is sequencable under r_1 iff $\Sigma^+ \subset L(I)$ r_1 is obtained from $r_{\mathcal{Q}}$ by deleting the p-atom corresponding to the production $S \rightarrow N_1$ r_2 is obtained from $r_{\mathcal{P}}$ by deleting the p-atoms corresponding to productions $S \rightarrow N_1$ and $S \rightarrow N_2$, adding the atom $p(Z, X, Y, W, Z, S, U_S, U_S, Z, < N_1 >)$ (where U_S is a new variable), and adding the atom $f(R)$ □

7. Conclusions.

The simplest recursive programs are those which involve a single intensional predicate ("single-IDB programs") Such programs permit of powerful optimizations, including the detection of commutativity and ZYT-linearizability We show that the natural generalizations of the commutativity and ZYT-linearizability problems (respectively, the sequencability and base-case linearizability problems) are undecidable The techniques developed for these results can also be used to provide tight undecidability results for program containment and equivalence

Acknowledgements

As always, Jeff Ullman has been an invaluable source of information and insight I am indebted to Moshe Vardi for posing the problem of sequencability, and for helpful discussions

References

[Abit89] Abiteboul, S [1989] "Boundedness is undecidable for Datalog programs with a single recursive rule," *Information Processing Letters 32*, pp 281-287

[BMSU86] Bancilhon, F , D E Maier, Y Sagiv and J D Ullman [1986] "Magic sets and other strange ways to implement logic programs," *Proc Fifth ACM Symposium on Principles of Database Systems*, pp 1-15

[BR87] Beeri, C and R Ramakrishnan [1987] "On the power of magic," *Proc Sixth ACM Symposium on Principles of Database Systems*, pp 269-283

[CM77] Chandra, A K and P M Merlin [1977] "Optimal implementation of conjunctive queries in relational databases," *Proc Ninth Annual ACM Symposium on the Theory of Computing*, pp 77–90

[HU79] Hopcroft, J E and J D Ullman [1979] *Introduction to automata theory, languages and computation*, Addison-Wesley

[Ioan89a] Ioannidis, Y E [1989] "Commutativity and its role in the processing of linear recursion," Tech Report 804, University of Wisconsin-Madison

[Ioan89b] Ioannidis, Y E [1989] "Towards an algebraic theory of recursion," Tech Report 801, University of Wisconsin-Madison

[Naug88] Naughton, J F [1988] "Compiling separable recursions," *ACM SIGMOD Intl Conf on the Management of Data*, pp 312-319

[RSUV89] Ramakrishnan, R , Y Sagiv, J D Ullman and M Y Vardi [1989] "Proof-tree transformation theorems and their applications," *Proc Eighth ACM Symposium on Principles of Database Systems*, pp 172-181

[SY81] Sagiv, Y and M Yannakakis [1981] "Equivalence among relational expressions with the union and difference operators," *J ACM 27*, pp 633-655

[Sagi87] Sagiv, Y [1987] "Optimizing Datalog programs," *Proc Sixth ACM Symposium on Principles of Database Systems*, pp 349-362

[Sara89] Saraiya, Y [1989] "Linearizing nonlinear recursions in polynomial time," *Proc Eighth ACM Symposium on Principles of Database Systems*, pp 182-189

[Sara90] Saraiya, Y [1990] "Polynomial-time program transformations in deductive databases," to appear in *Proc Ninth ACM Symposium on Principles of Database Systems*

[Shmu87] Shmueli, O [1987] "Decidability and expressiveness aspects of logic queries," *Proc Sixth ACM Symposium on Principles of Database Systems*, pp 237-249

[Ullm89] Ullman, J D [1989] *Principles of Database and Knowledge-base Systems*, Vol II, Computer Science Press, Rockville, Md

[ZYT88] Zhang, W , C T Yu and D Troy, "A necessary and sufficient condition to linearize doubly recursive programs in logic databases," unpublished manuscript, Dept of EECS, University of Illinois at Chicago