On the Complexity of Equivalence between Recursive and Nonrecursive Datalog Programs

Surajit Chaudhuri
Hewlett-Packard Laboratories
Palo Alto, CA 94304
E-mail: chaudhuri@hpl.hp.com

Moshe Y. Vardi*
IBM Almaden Research Center
San Jose, CA 95120-6099
E-mail: vardi@almaden.ibm.com

Abstract

In a previous paper, we have proved tight complexity bounds for the equivalence of recursive and nonrecursive Datalog programs: triply-exponential time in general and doubly-exponential space for linear programs. In this paper, we show that under realistic restrictions on the classes programs under consideration, equivalence of recursive and nonrecursive programs can be less intractable; for the classes of programs we consider the complexity of equivalence ranges from NP to co-NEXPTIME.

1 Introduction

It has been recognized for some time that first-order database query languages are lacking in expressive power [AU79, GM78, Zl76]. Since then, many higher-order query languages have been investigated [AV89, CH80, Ch81, CH82, Im86, Va82]. A query language that has received considerable attention is *Datalog*, the language of logic programs (known also as Horn-clause programs) without function symbols [Ka90, Ul89], which is essentially a fragment of fixpoint logic [CH85, Mo74].

The gain in expressive power does not, however, come for free; evaluating Datalog programs is harder than evaluating first-order queries [Va82]. Recent works have addressed the problems of finding efficient evaluation methods for Datalog programs ([BR86] is a good survey on this topic) and developing optimization techniques for Datalog [MP91, Na89, NRSU89]. The techniques to optimize evaluation of queries are often based on the ability to transform a query into an equivalent one that can be evaluated more efficiently [RSUV93]. Therefore, determining equivalence of queries is one of the most fundamental optimization problems. Naturally, the problem of determining equivalence of Datalog programs has received attention. Unfortunately, Datalog program equivalence is undecidable [Sh87].

Since the source of the difficulty in evaluating Datalog programs is their recursive nature, the first line of attack in trying to optimize such programs is to eliminate the recursion. Thus, a problem of special interest is that of determining the equivalence of a given recursive Datalog program to a given nonrecursive[1] program. We call this the *equivalence* problem. Note that the equivalence problem is different from that of determining whether a given recursive Datalog program is equivalent to *some* nonrecursive program. The latter program is called the *boundedness* problem and is known to be undecidable [GMSV93]. Recently, we showed [CV92] that the problem of deciding equivalence of a recursive and a nonrecursive program is decidable in triply exponential time and this bound is optimal, i.e., the problem is 3EXPTIME-complete. This result shows that while the equivalence problem is decidable, it is highly intractable. (In fact, to our knowledge this is the most intractable decidable problem in database theory.)

The highly intractable nature of the equivalence problem limits possible use of the decidability result. The triply-exponential lower bound applies, however, to arbitrary nonrecursive and arbitrary recursive programs. Thus, the result in [CV92] leaves open the possibility that under realistic restrictions on the classes of recursive and nonrecursive programs, the equivalence problem is *less* intractable.

In this paper, we study the effect of reasonable restrictions on the complexity of the equivalence problem. For example, we know that a nonrecursive program can be expressed as a finite union of conjunctive queries. We observed, however, in [CV92] that the equivalence of a recursive program and a union of conjunctive queries is complete for *doubly* exponential time (see also [Me93]). As we showed in [CV92], an additional exponential blow-up in complexity of equivalence

*Current address: Dept of Computer Science, Rice University, P.O. Box 1892, Houston, TX 77251-1892, E-mail: vardi@cs.rice.edu.

[1] A Datalog program is nonrecursive if its dependency graph is acyclic.

occurs because there exist succinct nonrecursive programs that expand into a union of conjunctive queries such that the latter is exponential in the size of the nonrecursive program. On the other hand, many nonrecursive programs, e.g., "flat" nonrecursive programs, where no intensional predicate occurs in the body of a rule, translate into unions of conjunctive queries without any such blow-up. In this paper, we express our results in terms of equivalence to a union of conjunctive queries, implicitly assuming that our nonrecursive programs are "flat". In addition, we also consider the effect on complexity if we restrict the nonrecursive program to be *connected*. A program is *connected* if every rule is connected, which refers to the connectedness of the graph "corresponding" to the rule (see Section 2). Intuitively, connectedness ensures that there is no cross-product in the body of the rule and the rule is *safe* [Ul89].

We study in details the complexity issues for the class of *monadic* Datalog programs. Monadic programs are Datalog programs where the intensional predicates have arity one. Important queries such as *path systems* are expressible as monadic programs. We will also show how complexity results vary with additional restrictions on monadic programs. In particular, we will consider well-known restricted classes such as the classes of *linear programs*, and *single-rule programs*. In a linear program there is at most one occurrence of an intensional predicate in every rule. A single-rule program (*sirup*) contains at most one recursive rule. Both restrictions have attracted a lot of attention in the literature (cf. [Ka88]). We also consider *nonpersistent* [Co89] programs. These are programs where in each rule no distinguished variable occurs in the recursive subgoals. Well-known programs such as the *same-generation* query (see [Ul89]) are nonpersistent.

We show that for the many of these restricted classes of program the equivalence problem is indeed less intractable, ranging from co-NEXPTIME to NP-complete (We note that since equivalence of conjunctive queries is NP-hard, we can not expect the complexity of equivalence of recursive and nonrecursive programs to be any easier than NP-hard). We use containment mappings as well as techniques from automata theory to derive our results.

The overview of the paper is as follows. In the next section, we discuss the preliminary notations and concepts that are used throughout this paper. Section 3 and Section 4 discuss results for monadic and nonpersistent Datalog programs respectively. We conclude with some reflections on the results so far and important open questions.

2 Preliminaries

2.1 Conjunctive Queries

A *conjunctive query* is a positive existential conjunctive first-order formula, i.e., the only connective allowed is $\wedge$ and the only quantifier allowed is $\exists$. Without loss of generality, we can assume that conjunctive queries are given as formulas $\varphi(x_1, \ldots, x_k)$ of the form $(\exists y_1, \ldots, y_m)(a_1 \wedge \ldots a_n)$ with free variables among $x_1, \ldots, x_k$, where the a_i's are atomic formulas of the form $p(z_1, \ldots, z_l)$ over the variables $x_1, \ldots, x_k, y_1, \ldots, y_m$. The variables $x_1, \ldots, x_k$ are called *distinguished* variables. variables. For example, the conjunctive query $(\exists y)(E(x, y) \wedge E(y, z))$ captures all pairs $\langle x, z\rangle$ such that there is a path of length 2 between x and z. In this example, x and z are the distinguished variables.

A union of conjunctive queries is a disjunction of conjunctive queries. A union of conjunctive queries $\Theta(x_1, \ldots, x_k)$ can be applied to a database D. The result

$$\Theta(D) = \{(a_1, \ldots, a_k) | D \models \Theta(a_1, \ldots, a_k)\}$$

is the set of k-ary tuples that satisfy Θ in D. If Θ has no distinguished variables, then it is viewed as a Boolean query; the result is either the empty relation or the relation containing the 0-ary tuple.

For every conjunctive query Q, we can define a *connection graph* (V_Q, E_Q) where V_Q is the set of variables in Q (including the distinguished variables of the query) and there is an edge between v and v' iff there is a literal in the query where variables v and v' occur in the same literal.

A conjunctive query is *connected* if its connection graph is a single connected component. Thus, connected queries contain no cross products and are safe. By *distance* between two variables in such a query, we will refer to the length of the shortest path between these variables in its connection graph (denoted $dist_Q(v, v')$).

2.2 Datalog

A (Datalog) program consists of a set of Horn clauses. A predicate that occurs in head of a rule is called an *intensional* (IDB) predicate. The rest of the predicates are called *extensional* (EDB) predicates.

Let Π be a Datalog program. Let $P^i_\Pi(D)$ be the collection of facts about an IDB predicate P that can be deduced from D by at most i applications of the rules in Π and let $P^\infty_\Pi(D)$ be the collection of facts about P that can be deduced from D by applications of the rules in Π, that is, $P^\infty_\Pi(D) = \bigcup_{i \geq 0} P^i_\Pi(D)$. It is known that the relation defined by an IDB predicate in a Datalog program Π, e.g., $P^\infty_\Pi(D)$, can be expressed by an infinite union of conjunctive queries. That is, for each IDB predicate P there is an infinite sequence $\varphi_0, \varphi_1, \ldots$ of conjunctive queries such that for every database D, we have $P^\infty_\Pi(D) = \bigcup_{i=0}^{\infty} \varphi_i(D)$. The φ_i's are called the *expansions* of P. We will also denote by P^i_Π, the finite union of all expansions for a predicate P that are obtained by at most i applications of rules (i.e., unfolding the recursion) in a Datalog program Π. Similarly, P^∞_Π will denote the infinite union of expansions of the predicate P in the program Π.

2.3 Containment

Let $\theta(x_1, \ldots, x_k)$ and $\psi(x_1, \ldots, x_k)$ be two conjunctive queries with the same vector of distinguished variables. We say that θ is *contained* in ψ (denoted $\theta \subseteq \psi$) if $\theta(D) \subseteq \psi(D)$ for each database D, i.e., if the following implication is valid.

$$\forall x_1 \ldots \forall x_k (\theta(x_1, \ldots, x_k) \rightarrow \psi(x_1, \ldots, x_k))$$

Equivalence of conjunctive queries can be defined in terms of containment. Thus, θ is *equivalent* to ψ iff θ is contained in ψ and vice-versa.

Definition 2.1 : A *containment mapping* from a conjunctive query ψ to a conjunctive query θ is a renaming of variables of ψ subject to the following constraints: (a) every distinguished variable must map to itself, and (b) after renaming, every literal in ψ must be among the literals of θ. ∎

Conjunctive-query containment can be characterized in terms of containment mappings (cf. [Ul89]).

Theorem 2.2: *A conjunctive query $\theta(x_1, \ldots, x_k)$ is contained in a conjunctive query $\psi(x_1, \ldots, x_k)$ iff there is a containment mapping from ψ to θ.*

Furthermore, we can make the following observation.

Lemma 2.3: *Let σ be a containment mapping from ψ to θ. Then, for every two variables u and v in ψ, $dist_\theta(\sigma(u), \sigma(v)) \leq dist_\psi(u, v)$.*

We say that the program Π with the goal predicate P is *contained* in a union of conjunctive queries Θ if $P^\infty_\Pi(D) \subseteq \Theta(D)$ for each database D. Equivalently, P is contained in Θ if the infinite union of conjunctive queries P^∞_Π is contained in Θ.

2.4 Expansion Trees

Expansions can be described in terms of *expansion trees*. The nodes of an expansion tree for a Datalog program Π are labeled by pairs of the form (α, ρ), where α is an IDB atom and ρ is an instance of a rule r of Π such that the head of ρ is α. The atom labeling a node χ is denoted α_χ and the rule labeling a node χ is denoted ρ_χ. In an expansion tree for an IDB predicate P, the root is labeled by a P-atom. Consider a node χ, where α_χ is the atom $R(\mathbf{t})$, ρ_χ is the rule

$$R(\mathbf{t}) : -P_1(\mathbf{t}^1), \ldots, P_m(\mathbf{t}^m),$$

and the IDB atoms in the body of the rule are $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$. Then χ has children $\chi 1, \ldots, \chi l$ labeled with the atoms $R_{i_1}(\mathbf{t}^{i_1}), \ldots, R_{i_l}(\mathbf{t}^{i_l})$, In particular, if all atoms in ρ_χ are EDB atoms, then χ must be a leaf. The query corresponding to an expansion tree is the conjunction of all EDB atoms in ρ_χ for all nodes χ in the tree, with the variables in the root atom as the free variables. Thus, we can view an expansion tree τ as a conjunctive query. Let $trees(P, \Pi)$ denote the set of expansion trees for an IDB predicate P in Π, then $P^\infty_\Pi \equiv \bigcup_{\tau \in trees(P,\Pi)} \tau$. Note that $trees(P, \Pi)$ is typically an infinite set. We say that an expansion tree τ' is *shorter* than an expansion tree τ if the longest distance from the root to a leaf in τ' is less than that in τ. We now introduce a notion of redundancy of expansion trees, adopted from [SV89].

Definition 2.4: an expansion tree τ for a predicate Q is *redundant* if there exists a strictly shorter expansion tree τ' such that $\tau \subseteq \tau'$. ∎

In the above definition, $\tau \subseteq \tau'$ indicates that the conjunctive query corresponding to the expansion tree τ is contained in the conjunctive query corresponding to the expansion tree τ'. We will use the above definition to argue that for containment we can ignore redundant expansion trees in P^∞_Π.

2.5 Containment of a Union of Conjunctive Queries in a Datalog Program

An upper bound on the complexity of equivalence can be obtained by solving two containment problems. Thus, an IDB predicate P in a program Π is equivalent to a union of conjunctive queries Θ if (a) Θ is contained in P, i.e., for all databases D, $\Theta(D) \subseteq P^\infty_\Pi(D)$ (b) P is contained in Θ, i.e., for all databases D, $P^\infty_\Pi(D) \subseteq \Theta(D)$.

In this section, we present results on (a), i.e., on containment of a union of conjunctive queries in a Datalog predicate. The complexity of containment of a union of conjunctive queries in a Datalog predicate is the combined data and expression complexity of evaluation of Datalog programs. Therefore, the above containment problem is EXPTIME-complete [CK86, CLM81, Sa88]. For restricted classes of Datalog predicates, the following results hold.

Theorem 2.5: *Containment of a union of conjunctive queries in a Datalog program is:*

- *PSPACE-complete for linear programs.*
- *NP-complete for linear monadic programs.*

Proof: We sketch the proof for the case of linear monadic programs. Let Θ be the union of conjunctive queries. Over a database, consisting of literals of Θ, a linear monadic Datalog program Π can derive no more than $|\Pi| * |\Theta|$ literals and therefore Θ must be contained among expansions in $P_{\Pi}^{|\Pi|*|\Theta|}$. Observe that for linear programs, the size of each expansion in $P_{\Pi}^{|\Pi|*|\Theta|}$ is polynomial in the size of Π and Θ. Hence, the result follows. ∎

In the rest of the paper, we concentrate primarily on the problem of containment of a Datalog program in a finite union of conjunctive queries. However, we will use Theorem 2.5 to determine complexity of the equivalence problem. The results on complexity of the containment and the equivalence problems are stated with respect to the combined size of the Datalog program and the finite union of conjunctive queries.

3 Monadic Programs

A Datalog program is *monadic* if all its intensional predicates are monadic. Important queries such as *path systems* may be expressed as monadic programs. For example, consider the *graph-reachability* problem, which finds all nodes that are reachable from a given set of nodes. This may be expressed by a linear monadic Datalog program.

Our results in this section show that complexity of containment (and equivalence) of a monadic program in a union of conjunctive queries may vary from NP to co-NEXPTIME under well-known restrictions on the programs. The main reason for the lower computational complexity is that for monadic programs there is no need to "remember" the atoms labeling nodes of expansion trees. In other words, we only need to consider the rules in the labels. For *linear* monadic programs, every expansion tree degenerates into a chain. Therefore, we can represent expansion trees for such programs as regular expressions over rule names.

3.1 Linear Monadic Sirups

Since the containment between two boolean conjunctive queries is NP-hard, it follows that the problem of containment of linear monadic sirups in a finite union of conjunctive queries is also NP-hard. Therefore, the best we can hope is to show is that the problem is NP-complete. Indeed, that turns out to be the case. The proof uses a *pumping* argument. The intuition behind the proof is that since conjunctive queries are of *bounded* length, they can only map to a small part of a long expansion. It turns out that expansions of a linear sirup have a strong regularity; it suffices to check containment of only a polynomially bounded number of expansions in the conjunctive queries. Without loss of generalization, we can assume that the recursive rule is not persistent, i.e., the variable in its recursive subgoal is distinct from its distinguished variable. (Observe that the recursive rule is redundant if it is persistent.)

Proposition 3.1: *Let P be the IDB predicate in a linear monadic sirup Π. Let Θ be a union of conjunctive queries. There exists an integer $lim(\Pi,\Theta) \in O(|\Theta|)$ such that over all databases D, $P_{\Pi}^{\infty}(D) \subseteq \Theta(D)$ iff $P_{\Pi}^{lim(\Pi,\Theta)} \subseteq \Theta$.*

Proof: Let us assume that the sirup has a single nonrecursive rule s and each conjunctive query in Θ is connected. Let the recursive rule be r. Therefore, the expansions have the form r^*s. Let $lim(\Pi,\Theta)$ be $|\Theta|+2$. We will use the symbol l for $lim(\Theta,\Pi)$. We know that $P_{\Pi}^{l} = \bigcup_{0\leq i<l} r^i s$. Let us assume that $\varphi \in \Theta$ has a containment mapping to $r^{l-1}s$. Since $l-1 > |\Theta| \geq |\varphi|$, it follows from Lemma 2.3 that φ must also map to literals from either r^{l-1} or $r^{l-2}s$ in the expansion $r^{l-1}s$. Let us assume its the former. In that case, it follows that φ maps to $r^{l-1}r^j s$ for any $j \geq 0$. If its the latter, then φ does not have the distinguished variable and hence the mapping may be extended to $r^j r^{l-2}s$, $j \geq 0$ as well. Therefore, $r^k s \subseteq \varphi$ where $k \geq l$. Hence, the containment holds. In the full paper, we show how we can generalize the above proof when Θ is not connected or when there are many nonrecursive rules. ∎

Theorem 3.2: *Containment of a linear monadic sirup in a union of conjunctive queries is NP-complete.*

Proof: Let P is the IDB predicate in a linear monadic sirup Π, and Θ be the union of conjunctive queries. The nontrivial direction is to show membership in NP. In view of Proposition 3.1, it follows that containment may be tested by checking whether $P_{\Pi}^{lim(\Pi,\Theta)} \in \Theta$. For linear monadic sirups, $P_{\Pi}^{lim(\Pi,\Theta)}$ is a polynomial in the size of Π and Θ, since $lim(\Pi,\Theta)$ is in $O(|\Theta|)$. Therefore, containment can be tested in NP. ∎

Corollary 3.3 is a consequence of Theorem 3.2. We note that equivalence of conjunctive queries is NP-hard, even when queries are monadic. Therefore, the equivalence between a linear monadic sirup and a union of conjunctive queries is NP-hard as well.

Corollary 3.3: *Equivalence between a linear monadic sirup and a union of conjunctive queries is NP-complete.*

3.2 Linear Monadic Programs

In this section, we relax the constraint that the Datalog program is a sirup and allow arbitrary number of recursive rules in the program. It turns out that for this class, the containment and the equivalence problems are PSPACE-complete.

3.2.1 Upper Bound:

In this section, we will establish the upper bound (membership in PSPACE) on the problem of containment of a linear monadic predicate P in a program Π in a union of conjunctive queries Θ. We prove this result by proposing an automaton, exponential in the size of the input, that accepts only those expansion trees among $tree(P, \Pi)$ that are *not* contained in Θ. Then, containment of P_Π^∞ in Θ can be tested by checking emptiness of the automaton. It is interesting to contrast our approach with that in [CV92]. To prove decidability of containment for arbitrary Datalog programs, we constructed two automata, both exponential in the size of the input, and checked whether one (which accepts all expansion trees of P) is contained in the other (which accepts all proof trees that are contained in Θ). Containment of automata is PSPACE-complete in the sizes of the automata whereas emptiness can be tested in NLOGSPACE.

Let us consider what property of linear monadic programs makes it possible for us to get away with constructing a single automaton and testing its nonemptiness. First, we note that in a monadic program, persistence of a variable is limited (see Proposition 3.4). Therefore, literals in Θ can not be mapped[2] to parts of an expansion tree that are "too far" apart. Such locality is a consequence of the restriction on the arity. Next, and no less important, is the restriction that the programs are linear. For linear programs, the neighborhood in an expansion tree where a given variable may appear is linear in the size of the program (as against exponential in the nonlinear case).

We now present a proposition that captures part of the above intuition. It says that we need only consider those expansion trees where each variable that occurs in a label of the expansion tree of a monadic program persists only in nodes that are apart at most by the size of the program.

Proposition 3.4: *Let τ be an expansion tree for an IDB predicate P in a linear monadic program Π that corresponds to the application of rules $r_{i_1} \dots r_{i_l}$. If a variable $\mathbf{x}$ appears in literals in r_{i_j} and in r_{i_k} of τ where $k - j > |\Pi|$, then τ is redundant.*

Proof: Since $k-j > |\Pi|$, there must be some rule s such that there are two *distinct* occurrences r_{i_m} and r_{i_n} of s among $r_{i_1} \dots r_{i_l}$. Without loss of generality assume $m < n$. Then, we can construct a *strictly* shorter expansion tree $\tau' = r_{i_1} \dots r_{i_m} r_{i_{n+1}} \dots r_{i_l}$. It follows that $\tau \subseteq \tau'$. Hence, τ is redundant. ∎

Example 3.5: Let us consider a monadic program with the following three recursive rules. We have omitted the nonrecursive rules.

$$\begin{aligned} \mathbf{r}:\ p(X) &\ :-\ a(X, W), q(X) \\ \mathbf{s}:\ q(X) &\ :-\ b(X, U), p(X) \\ \mathbf{t}:\ q(X) &\ :-\ c(X, Y), p(Y) \end{aligned}$$

[2] For simplicity, assume that Θ is connected.

Consider any expansion of p where a variable persists for more than two rule-compositions. Such an expansion must have embedded in them expansions for rs or sr. In other words, the expansion must be of the form $\varphi rs\psi$ or $\varphi sr\psi$ where φ and ψ are legal sequences of rule applications. However, such an expansion is redundant since it must be contained in $\varphi\psi$. ∎

Intuitively, Proposition 3.4 tells us that in order to consider containment of a linear monadic program in a union of conjunctive queries, we can ignore expansions of the program where variables persist for "too long". This motivates the following definition.

Definition 3.6 : Let τ be an expansion tree for an IDB predicate P in a linear monadic program Π that corresponds to the application of rules $r_{i_1} \dots r_{i_l}$. Such an expansion tree τ satisfies *bounded-persistence* property if every variable in τ persists in no more than $|\Pi|$ consecutive rule applications, i.e., if a variable in τ appears in literals of r_{i_j} as well as in literals r_{i_k}, then $k - j \leq |\Pi|$. ∎

Based on Definition 3.6, we define $BPtrees(P, \Pi)$ to be the set of all expansion trees in $trees(P, \Pi)$ that have the bounded-persistence property. In view of Proposition 3.4, it follows that the predicate P in the program Π is contained in a union of conjunctive queries Θ iff $BPtrees(P, \Pi)$ is contained in Θ. We now describe an automaton that accepts all expansions in $BPtrees(P, \Pi)$ that are not contained in a union Θ of conjunctive queries.

Proposition 3.7: *Let P be an IDB predicate in a linear monadic Datalog program Π. Let Θ be a union of conjunctive queries. There exists an automaton $A_{p,\Pi}^{\Theta}$, whose size is exponential in the size of Π and Θ, such that*

$$T(A_{p,\Pi}^{\Theta}) = \{\tau \in BPtrees(P, \Pi) |\ \tau \not\subseteq \Theta\}.$$

Proof: We will describe the automaton assuming that Θ is a single connected conjunctive query. Let us assume that the total size of Θ and Π is w. From Lemma 2.3, it follows that the maximum distance between any two variables in the image of Θ in an expansion τ can be no more than w. Let $\tau = r_{i_1} \dots r_{i_n}$. Since $\tau \in BPtrees(P, \Pi)$, we conclude from Proposition 3.4 that if r_{i_j} and r_{i_k} are two rule instances that contain literals to which Θ maps, then $|k-j| \leq w^2$. Let $u = w^2+1$. Given any sequence of u rule applications, we can determine whether or not there is a mapping from Θ to this sequence. Let $\mathcal{M}$ be the set of rule application sequences of length u, ending with a nonrecursive rule, that are *not* contained in Θ. Let $\mathcal{R}$ be the set of rules. Our goal is to construct an automaton that checks whether a given $\tau \in BPtrees(P, \Pi)$ is such that there is no subsequence of u rule applications in τ to which Θ maps. Let $A^{\Theta}_{p,\Pi} = (\Sigma, S, S_0, \rho, F)$, where $\Sigma = \mathcal{R}$. Let the set of states $S = V^{u+1} \cup \{fail\}$, where $V = \mathcal{R} \cup \{\$\}$. The symbol \$ denotes "no rule application". The state $fail$ denotes that there is some subsequence in τ to which Θ maps. Every other state is a sequence of the form $< x_{-u} \dots x_{-1} x_0 >$ that encodes the past u rule applications, followed by the current rule application (x_0). The set of starting states is $S_0 = \$^u \times r$ where r is a rule that has the predicate P in the head. The leading u rule applications in a starting state are \$ and they signify that no rules were applied. The set of final states is $F = \mathcal{M} \times \{\$\}$. Thus, the final state are those sequences where the last rule application corresponds to a nonrecursive rule.

We omit the description of the transition function, which ensures that transitions permit only valid sequences of rule applications. Since R and u are in $O(|\Pi|)$ and $O(w^2)$ respectively, it follows that the size of the automaton is at most exponential in Π and Θ. The result generalizes when Θ is a union of conjunctive queries (connected or otherwise). ∎

Proposition 3.8: *Containment of a linear monadic program in a union of conjunctive queries is in PSPACE.*

Proof: From Proposition 3.7, it follows that containment may be decided by testing emptiness of an automaton exponential in the size of the program and the union of conjunctive queries. However, nonemptiness of an automaton may be tested in NLOGSPACE in the size of the automaton. Therefore, it follows that the problem of deciding containment of an IDB predicate of a linear monadic program in a union of conjunctive queries is in PSPACE. Note that an explicit construction of the automaton, which may be of exponential size, can be avoided by generating states of the automaton on the fly while testing for reachability of states. ∎

3.2.2 Lower Bound:

In this section, we show that containment of a linear monadic program in a union of conjunctive queries is PSPACE-hard. The proof is by reduction from the problem of deciding *regular expression non-universality*, which is PSPACE-complete (see [HU79]). We use the following variant of the above result. The abbreviation NFA stands for non-deterministic finite state automaton.

Proposition 3.9: *Let A be an arbitrary NFA with an alphabet $\Sigma = \{0, 1\}$ and without any ϵ-transitions. Let $L(A)$ be the language accepted by the NFA. Then, the problem of deciding $\Sigma^+ \subseteq L(A)$ is PSPACE-hard.*

Let the NFA A (without any ϵ transitions) be $A = (\Sigma, Q, \{q_1\}, \delta, F)$. The transition function δ is a map from $Q \times \Sigma$ to 2^Q. Assume that our NFA has n states, i.e., $|Q| = n$. We can represent operation of such an automaton by using the notion of a *configuration*. Intuitively, a configuration will contain a complete description of the current state of a family of computations. Thus, we will be able to represent the family of all computations of the NFA on a given input word by a sequence of configurations. We represent a configuration of the automaton by $\# s_1 \dots s_n\ a$, where for all $i \in 1 \dots n$, s_i is either 1 or 0 depending on whether q_i is one of the reachable states from the initial state with the input consumed so far. The symbol a represents the next input symbol and is either $0, 1$ or @. The symbol @ denotes the end of input string. The symbol # acts as the separator between successive configurations. A configuration $\# s_1 \dots s_n\ a$ is initial if $s_1 = 1$ and $s_j = 0$ for $1 < j \leq n$, and is accepting if it has some $s_j = 1$ where $q_j \in F$. A successive pair of configurations $\# s_1 \dots s_n\ a \# t_1 \dots t_n\ b$ obeys the transition function when $t_i = 1$ iff $q_i \in \delta(q_j, a)$ for some q_j such that $s_j = 1$. Thus, a word $a_1 \dots a_l$ is accepted by the NFA if there is a sequence of $l+1$ configurations over $a_1 \dots a_l$ that starts with an initial configuration, ends with an accepting configuration, and obeys the transition function.

The idea in the encoding is that unfolding expansions of the recursive program Π corresponds to a sequence of configurations of the NFA. The role of the union Θ of conjunctive queries is to check whether the sequence corresponds to either an computation that is illegal or that ends in an accepting state. Thus, if an expansion τ of Π represents a legal but not an accepting computation, then $\tau \not\subseteq \Theta$. Thus, we have containment in Θ iff the NFA accepts Σ^+.

The symbols in the configuration are drawn from the extended alphabet $\Sigma' = \Sigma \cup \{\#, @\}$. We have a monadic EDB predicate q_a for every symbol $a \in \Sigma'$ and $q_a(X)$ denotes that the symbol to which X points is a. We also have an EDB predicate $first$ to point to the first symbol in the first configuration. The binary predicate $succ$ denotes the adjacency relation among symbols in the configuration. We can encode a string such as "011" by the set of conjuncts $q_0(X), q_1(Y), q_1(Z), succ(X, Y), succ(Y, Z)$. We use the abbreviation $succ^{j_1}(X) = a_1, \dots, succ^{j_k}(X) = a_k$ to denote

$$succ(X, X_1), \dots, succ(X_{m-1}, X_m), q_{a_1}(X_{j_1}), \dots q_{a_k}(X_{j_k})$$

We now describe the monadic Datalog program Π. The IDB predicates in Π simulate the NFA by moving a pointer along a sequence of configurations, starting at the symbol pointed to by $first$ and stopping when the end of the string symbol @ is encountered. There is a monadic IDB predicate $Bit_i(X)$ for each of the position $0 \leq i \leq n+1$ in configurations. Thus, Bit_0 points to the separator # position, Bit_i points to the position that indicates whether the state q_i of NFA is reachable $(1 \leq i \leq n)$ from the current configuration. The rule for initialization begins moving the pointer from the symbol pointed to by $first$. The IDB predicate C is 0-ary and serves as the goal predicate of the Datalog program.

$$C :- first(Y), Bit_0(Y), q_\#(Y)$$

The rules for moving along the sequence of configurations are given below. In the first rule we have that $0 \leq i \leq n$ and $a \in \Sigma$.

$$\begin{array}{lll} Bit_i(X) & :- & Bit_{i+1}(Y), succ(X,Y), q_a(Y) \\ Bit_{n+1}(X) & :- & Bit_0(Y), succ(X,Y), q_\#(Y) \end{array}$$

The rule for termination recognizes the end of string @:

$$Bit_{n+1}(X) :- succ(X,Y), q_@(Y)$$

The union of conjunctive queries Θ will be a 0-ary predicate. It detects the following two conditions. First, Θ ensures that the sequence of configurations is a legal one. Next, it detects that the NFA has reached the final state. It turns out that the number of queries in Θ as well as the size of each query is polynomial in the size of NFA.

The *error-detecting* queries make Θ true in the event of an error. For example, the following conjunctive query checks for a violation of the transition relation when $q_i \in \delta(q_j, a)$:

$$\begin{array}{l} q_\#(X), succ^j(X) = 1, succ^{n+1}(X) = a, \\ \quad succ^{n+2+i}(X) = 0 \end{array}$$

In addition to such errors in the transition function, errors could also appear in initialization, encoding (e.g., exactly one symbol may appear in one position), and in sequencing (e.g., every # is followed by another # after $n+1$ letters).

We now turn to the queries in Θ that become true when when the final configuration is accepting. For every $q_j \in F$ in the NFA, there is a conjunctive query in Θ:

$$q_\#(X), succ^j(X) = 1, succ^{n+1}(X) = @.$$

Using the notations introduced above, we can prove the following proposition.

Proposition 3.10: $\Sigma^+ \subseteq L(Q) \Leftrightarrow C_\Pi^\infty \subseteq \Theta$

Theorem 3.11 : *Containment and Equivalence of a linear monadic program in a union of conjunctive queries are PSPACE-complete.*

Proof: Using Propositions 3.10 and 3.8, we conclude the containment problem is PSPACE-complete. Let us consider the equivalence problem now. In view of Theorem 3.10 and Theorem 2.5, we infer that the equivalence problem is in PSPACE. We now prove that the problem is PSPACE-hard by reducing the containment problem (which is PSPACE-hard) to the equivalence problem. To check whether $\Pi_1 \subseteq \Pi_2$, we rename the IDB predicates of the two programs to ensure that they are distinct, and then we add the rule $q(X) :- p_i(X)$ to Π_i, where q is a new IDB predicate and p_i is the goal predicate of Π_i, for $i = 1, 2$. Let the new programs be Π_1' and Π_2'. It follows that $\Pi_1 \subseteq \Pi_2$ iff $\Pi_1' \cup \Pi_2' \equiv \Pi_2'$ ∎

3.3 Connected Queries

In this section, we consider containment and equivalence of monadic programs in *connected* conjunctive queries (see Section 2.1). Such queries contain no cross products and are "safe", i.e., all distinguished variables occur in the body of the query.

Example 3.12: Consider the following three queries.

$$\begin{array}{lll} Q_1(X) & :- & s(X,Z), t(U,V) \\ Q_2(X) & :- & s(X',Z), t(Z,V) \\ Q_3(X) & :- & s(X,Z), t(Z,V) \end{array}$$

The query Q_1 has a connection graph that is disconnected. The connection graph for the query Q_2 is also disconnected since the distinguished variable is not connected, i.e., the query is not safe. The query Q_3 is connected.

∎

We prove that the upper bound of containment in (and equivalence with) connected queries is in the second level of polynomial hierarchy.

Proposition 3.13: *Let P be an IDB predicate in a linear monadic program Π. Let Θ be a union of conjunctive queries. There exists an integer $lim(\Pi, \Theta) \in O(|\Theta| * |\Pi|)$ such that over all databases D, $P_\Pi^\infty(D) \subseteq \Theta(D)$ iff $P_\Pi^{lim(\Pi,\Theta)} \subseteq \Theta$.*

Proof: We need only show that expansion trees in $BPtrees(P, \Pi)$ are contained in Θ. Since the queries in Θ are connected (and hence safe), each of them contains the distinguished variable. Using Proposition 3.4, it follows that if $\varphi \in \Theta$ maps to an expansion tree τ, then at least one literal in φ must map to a literal in the first $|\Pi|$ rule applications of τ. In general, since there are at most $|\Theta|$ variables in Θ, it suffices to check containment of $P_\Pi^{lim(\Pi,\Theta)}$ where $lim(\Pi, \Theta) \in O(|\Theta| * |\Pi|)$. ∎

Theorem 3.14: *Containment and equivalence of a linear monadic program in a union of connected conjunctive queries are in Π_2^p.*

Proof: Assume that Π is a linear monadic program and Θ is a union of connected conjunctive queries. The containment fails to hold iff there is some expansion in $P_{\Pi}^{lim(\Pi,\Theta)}$ which is not contained in Θ. Therefore, we can nondeterministically guess such an expansion φ and see if $\varphi \not\subseteq \Theta$. From Proposition 3.13, it follows that $lim(\Pi, \Theta)$ is a polynomial in the size of the input. Therefore, φ is polynomial in the size of the program. Hence, $\varphi \not\subseteq \Theta$ is in co-NP in the size of Π and Θ. Thus, the containment problem is in Π_2^p. From Theorem 2.5, it follows that equivalence is in Π_2^p as well. ∎

3.3.1 Nonlinear Programs

So far we have worked with the restriction that monadic programs are linear. Recall that for linear programs, each expansion tree is a chain. However, for arbitrary monadic programs, it is a tree and need not be a chain. Nonetheless, some of the proof techniques that we have adopted so far carries over to the nonlinear case. In particular, we can prove the following result using an argument similar to that in Proposition 3.4.

Proposition 3.15: *Let P be an IDB predicate in a monadic program Π. An expansion tree τ of P is redundant if there is a chain of nodes of length $m > |\Pi|$ in τ such that a variable $\mathbf{x}$ appears in the label of each of the nodes in the chain.*

Let us assume that there is a containment of a recursive predicate P in a query Θ. We know that if Θ is connected (and hence safe), it must contain the distinguished variable. Using Proposition 3.15, we conclude that for each nonredundant expansion tree τ, at least one of the literal in the query must map within depth $|\Pi|$. We extend this argument to show that for containment of the predicate P, it suffices to check containment of all expansion trees of depth bounded by $O(|\Pi| * |\Theta|)$. However, note that if the program is nonlinear, then the size of such expansions can be exponential in the size of the program. This leads to the following theorem.

Theorem 3.16: *Containment and Equivalence of a monadic program in a union of connected conjunctive queries are in co-NEXPTIME.*

Proof: Let P be an IDB predicate in a monadic Datalog program Π. Let Θ be a union of connected conjunctive queries. Note that if the depth of expansions is bounded by $O(|\Pi| * |\Theta|)$, then the size of every expansion φ of P is bounded by an exponential in the size of input. We can test for containment as in Proposition 3.14. However, now the size of each expansion φ can be an exponential in the size of the program Π. Therefore, it follows that the containment problem is in co-NEXPTIME. ∎

4 Nonpersistent Programs

A rule is *nonpersistent* [Co89] if none of the distinguished variables occurs in any of the recursive subgoals. Many interesting queries can be expressed as nonpersistent programs. For example, the well-known *same-generation* query has a single recursive rule which is given below. It satisfies the nonpersistence property:

$$sg(X, Y) : -par(X, X'), par(Y, Y'), sg(X', Y')$$

The proof techniques developed in the previous section for monadic programs can be used for the class of nonpersistent programs. This is so because to arrive at the results on upper bound of the containment problem, we exploit only the fact that no variable in a nonredundant expansion tree of a monadic program can be persistent in a "long" sequence of labels in the expansion tree (see Section 3.2.1). The above property holds not only for monadic programs, but also for nonpersistent programs. Therefore, we can prove results analogous to Theorems 3.2, 3.8, 3.14, and 3.16.

Theorem 4.1: *Containment of a linear nonpersistent sirup in a union of conjunctive queries is NP-complete.*

Theorem 4.2: *Containment of a linear nonpersistent program in a union of conjunctive queries is PSPACE-complete.*

Theorem 4.3: *Containment of a linear nonpersistent program in a union of connected conjunctive queries is in Π_2^p.*

Theorem 4.4: *Containment of a nonpersistent program in a union of connected conjunctive queries is in co-NEXPTIME.*

As mentioned in the introduction, containment of unions of conjunctive queries in Datalog programs is in EXPTIME. Combining this with Theorem 4.4, we get:

Theorem 4.5: *The problem of deciding equivalence of a Datalog program with a union of connected conjunctive queries is in co-NEXPTIME for nonpersistent programs.*

From Theorems 4.2 and Theorem 2.5, we conclude that the complexity of equivalence goes down to PSPACE for linear nonpersistent programs. This bound is optimal since it turns out that we can reduce the containment problem (PSPACE-hard) for linear nonpersistent programs to the equivalence problem.

Theorem 4.6: *The problem of deciding equivalence of a Datalog program with a union of conjunctive queries is PSPACE-complete for linear nonpersistent programs.*

Complexity of Equivalence					
Linear	Monadic	Sirup	NonPersistent	Connected	Complexity Bound
$\surd$					EXPSPACE-complete
$\surd$	$\surd$				PSPACE-complete
$\surd$	$\surd$	$\surd$			NP-complete
$\surd$	$\surd$			$\surd$	Π_2^p
	$\surd$			$\surd$	co-NEXPTIME
$\surd$			$\surd$		PSPACE-complete
			$\surd$	$\surd$	co-NEXPTIME

5 Concluding Remarks

In [CV92], using automata-theoretic techniques, we showed that the complexity of equivalence of a Datalog predicate with a nonrecursive predicate is 3EXPTIME-complete in general and 2EXPSPACE-complete for linear programs. In this paper, we have shown that under realistic restrictions, the complexity of equivalence can span a spectrum of complexity bounds, ranging from being NP-complete to being in co-NEXPTIME.

The following table summarizes our results on the upper bounds on the equivalence problem for restricted Datalog programs with union of conjunctive queries. The symbol $\surd$ signifies that the restriction applies. Thus, the first row presents the complexity bound for equivalence of the "flat" nonrecursive programs with arbitrary linear Datalog programs. The complexity results refer to the upper bounds but for the first three rows as well as the sixth row, they also represent matching lower bounds.

Several open questions remain. For example, while for linear monadic programs, the complexity of equivalence is PSPACE-complete, the best upper bound that we are able to show for arbitrary monadic programs is 2EXPTIME (from [CV92]). A full understanding of such a "gap" in complexity remains an open issue.

References

[AU79] Aho, A.V., Ullman, J.D.: Universality of data retrieval languages. *Proc. 6th ACM Symp. on Principles of Programming Languages*, 1979, pp. 110–117.

[AV89] Abiteboul, S., Vianu, V.: Fixpoint Extensions of First-Order Logic and Datalog-like languages. *Proc. 4th IEEE Symp. on Logic in Computer Science*, 1989, pp. 71–79.

[BR86] Bancilhon, F., Ramakrishnan, R.: An amateur's introduction to recursive query processing strategies. *Proc. ACM Conf. on Management of Data*, Washington, 1986, pp. 16–52.

[Ch81] Chandra, A.K.,: Programming primitives for database languages. *Proc. 8th ACM Symp. on Principles of Programming Languages*, Williamsburg, 1981, pp. 50–62.

[CH80] Chandra, A.K., Harel, D.: Computable queries for relational databases. *J. Computer and Systems Sciences* 21(1980), pp. 156–178.

[CH82] Chandra, A.K., Harel, D.: Structure and Complexity of Relational Queries. *J. Computer and Systems Sciences* 25(1982), pp. 99–128.

[CH85] Chandra, A.K., Harel, D.: Horn Clause Queries and Generalizations. *J. Logic Programming*, **2** (1985), 1–15.

[CLM81] Chandra, A. K., H. R. Lewis, and J. A. Makowsky, Embedded implicational dependencies and their inference problem. *Proc. 13th ACM Symp. on Theory of Computing*, 1981, pp. 342–354.

[Co89] Cosmadakis, S.S.: On the First-Order Expressibility of Recursive Queries. *Proc. 8th ACM Symp. on Principles of Database Systems*, Austin, 1989, pp. 311–323.

[CK86] Cosmadakis, S.S., Kanellakis, P.: Parallel evaluation of recursive rule queries. *Proc. 5th ACM Symp. on Principles of Database Systems*, Cambridge, 1986, pp. 280–293.

[CV92] Chaudhuri S., Vardi M. Y.: On the Equivalence of Recursive and Nonrecursive Programs. *Proc. 11th ACM Symp. on Principles of Database Systems*, San Diego. Full version available as IBM Research Report RJ9596, Nov 1993.

[GMSV93] Gaifman, H., Mairson, H., Sagiv, Y., Vardi M.Y.: Undecidable optimization problems for database logic programs. J. ACM 40(1993), pp. 683-713.

[GM78] Gallaire, H., Minker, J.: *Logic and Databases.* Plenum Press, 1978.

[HKMV91] Hillebrand, G.G., Kanellakis, P.C., Mairson, H.G., Vardi, M.Y.: Tools for Datalog boundedness. *Proc. 10th ACM Symp. on Principles of Database Systems*, May 1991, pp. 1–12.

[HU79] Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Im86] Immerman, N.: Relational queries computable in polynomial time. *Information and Control* 68(1986), pp. 86–104.

[Ka88] Kanellakis, P. C.: Logic programming and parallel complexity. in *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann, Los Altos, pp. 545–585, 1988.

[Ka90] Kanellakis P.C.: Elements of Relational Database Theory. *Handbook of Theoretical Computer Science*, Vol. B, Chapter 17, J. van Leeuwen, A.R. Meyer, N. Nivat, M.S. Paterson, D. Perrin ed., North-Holland 1990.

[MP91] Mumick, I.S.,Pirahesh, H.: Overbound and right-linear queries. *Proc. 10th ACM Symp. on Principles of Database Systems*, 1991, pp. 127–141.

[Me93] Van der Meyden, R.: Recursively indefinite databases. Theoretical Computer Science 116(1993), 151-194.

[Mo74] Moschovakis, Y.N.: *Elementary Induction on Abstract Structures.* North Holland, 1974.

[Na89] Naughton, J.F.: Minimizing function-free recursive definitions. *J. ACM* 36(1989), pp. 69–91.

[NRSU89] Naughton, J.F., Ramakrishnan, R., Sagiv, Y., Ullman, J.D.: Efficient evaluation of right , left , and multilinear rules. *Proc. ACM-SIGMOD Int'l Conf. on Management of Data*, 1989, pp. 235–242.

[RSUV93] Ramakrishnan, R., Sagiv, Y., Ullman, J.D., Vardi, M.Y.: Logical query optimization by proof-tree transformation. J. Computer and System Sciences 47(1993), pp. 222-248.

[Sa88] Sagiv, Y.: Optimizing Datalog programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker (ed.), Morgan Kaufmann Publishers, 1988, pp. 659–698.

[SV89] Sagiv, Y. and Vardi M. Y.: Safety of Datalog Queries over Infinite Databases. *Proc. 8th ACM Symp. on Principles of Database Systems*, 1989, pp. 160-171.

[SY81] Sagiv,Y., Yannakakis M.: Equivalences among Relational Expressions with the union and difference operators. *JACM*, 27:4, pp 633-655.

[Sh87] Shmueli, O.: Decidability and expressiveness aspects of logic queries. *Proc. 6th ACM Symp. on Principles of Database Systems*, 1987, pp. 237–249.

[Ul89] Ullman, J.D.: *Principles of Database and Knowledge Base Systems*, Vol. 2, Computer Science Press, 1989.

[Va82] Vardi, M.Y.: The complexity of relational query languages. *Proc. 14th ACM Symp. on Theory of Computing*, San Francisco, 1982, pp. 137–146.

[Zl76] Zloof, M.; *Query-by-Example: Operations on the Transitive Closure.* IBM Research Report RC5526, 1976.