Predicate Boundedness of Linear Monadic Datalog is in PSPACE

Ron van der Meyden

School of Computer Science and Engineering
University of New South Wales, Sydney, NSW 2052
Australia

phone: +61-2-9385-4897
fax: +61-2-9385-5995
email: meyden@cse.unsw.edu.au
www: http://www.cse.unsw.edu.au/~meyden

Aug 26, 2000

Abstract: Datalog programs, are a special case of logic programs without function symbols. Detection of boundedness permits Datalog programs to be optimized by the elimination of recursion. To determine whether a predicate is bounded in a Datalog program is known to be undecidable. However, previous work (Cosmadakis et al., 20th ACM Symposium on the Theory of Computing, 1988) has show that for monadic Datalog, this problem is decidable, with upper and lower bounds of EXPSPACE and PSPACE respectively for linear monadic programs. We establish that predicate boundedness for linear monadic programs is in fact in PSPACE, yielding a tight bound for this problem.

submitted to:
International Journal of the Foundations of Computer Science

Topic: Database Theory, Query Optimization, Datalog, Boundedness, Recursive Queries

1 Introduction

Database logic programs [10], also known as Datalog programs, are a special case of logic programs without function symbols. Datalog programs have been the subject of considerable study over the past decade, motivated by the fact that they provide greater expressive power than first order relational database query languages such as the relational calculus. Specifically, Datalog provides the ability to define *recursive* queries, such as transitive closure, that are not expressible in first order languages [1]. This increased expressive power comes at the cost of increased query evaluation time, however. The use of recursion also prevents the application of query optimization techniques applicable to first order queries. It is therefore desirable to study program optimizations that eliminate the use of recursion where possible. An approach proposed for such optimization is the detection of boundedness. A recursive Datalog program is said to be *bounded* if it is equivalent to some nonrecursive program.

As an example of a recursive, but bounded, Datalog program, consider the following program (from [8]) consisting of two rules:

$$\begin{array}{lcl} buys(X,Y) & :- & likes(X,Y) \\ buys(X,Y) & :- & trendy(X), buys(Z,Y) \end{array}$$

It is assumed that we are to be given an *extensional database* consisting of a set of facts D in the predicates "*likes*" and "*trendy*". The program can be viewed as a recursive definition of the *intensional database* predicate "*buys*". The semantics of the program, intuitively, is to find substitutions for the variables X, Y, Z so that all the atoms to the right of the "$:-$" in some rule match to facts in the database, and to generate the atom in the left hand side of the rule (under the same substitution) as a new atom of the database as a consequence of the match. This process is repeated on the resulting larger database, until a fixpoint is reached, which then provides the interpretation of the intensional database predicate.

It can be shown that this program is equivalent, in the sense that the same fixpoint is computed for all extensional databases, to the following program [8]

$$\begin{array}{l} buys(X,Y) :- likes(X,Y) \\ buys(X,Y) :- trendy(X), likes(Z,Y) \end{array}$$

As the right hand sides of the rules involve only extensional database predicates, this program is non-recursive.

In above example, we find that the *whole* program may be replaced by a nonrecursive program. It is sometimes possible to remove dependencies on recursive rules for some of the predicates even when the program itself is not equivalent to a nonrecursive program. For example, consider the following program (from [3]):

$$\begin{array}{lcl} Q'(X) & :- & A(X,Y), Q(Z) \\ Q(X) & :- & A(X,Y), Q(Y) \\ Q(X) & :- & B(X). \end{array}$$

In this program the predicate Q' calls the recursive predicate Q. However, it is easily seen that the recursion may be eliminated from Q' by replacing the rule for Q' by the rule

$$Q'(X) \quad :- \quad A(X,Y), B(T)$$

This yields a program equivalent to the original program. In situations such as this, the *predicate* Q' is said to be bounded. (The predicate Q is unbounded in this example, so the program itself is not bounded.)

The programs above may be seen to be bounded by straightforward considerations, which might lead one to expect that a simple test for detecting boundedness can be constructed. In fact, boundedness can arise in ways much more complex than the above cases suggest, and in its most general form, deciding whether a given Datalog program is bounded turns out to be undecidable [4]. However, a number of decidable cases are known, and upper and lower bounds on the complexity of these cases have been determined [3, 5]. In some instances the known upper and lower bounds do not match. In this paper we close the gap for one such case: predicate boundedness of *linear monadic* Datalog. A program is *linear* if the body of each rule contains at most one occurrence of an IDB predicate, and *monadic* if all intensional database predicates are monadic. We prove that this problem is in PSPACE, improving the previous EXPSPACE upper bound. Together with the known PSPACE lower bound, this implies that the problem is PSPACE complete. (Boundednes is a simpler problem, already known to be PSPACE-complete [3].) The proof requires the introduction of some new techniques, that we hope may find application elsewhere in the theory of Datalog programs.

The structure of the paper is as follows. The following section recalls the necessary background from the Datalog literature. Section 3 contains the proof of the main result. Section 4 concludes.

2 Definitions

In this section we briefly recall some standard definitions from the Datalog literature. The reader is referred to [10, 3] for more details and motivation.

We assume given a set of *extensional database* (EDB) and *intensional database* (IDB) predicate symbols. As we deal only with monadic programs, the latter are required to have arity one. An *atom* is an expression of the from $P(x_1, \ldots, x_n)$ where P is an EDB or IDB predicate of arity n and the x_i are either constants or variables. A *rule* is an expression of the form

$$Q(X): -B(X, \mathbf{Y})$$

where $Q(X)$ is an atom, called the *head* of the rule, such that Q is an IDB predicate, and $B(X, \mathbf{Y})$, called the *body* of the rule, is a finite set of atoms in the EDB and IDB predicates, whose variables are among X and the variables $\mathbf{Y}$. (We may omit reference to the variables X and $\mathbf{Y}$ according to convenience.) A rule is said to be an *initialization* rule if its body contains no IDB predicates; otherwise, the rule is said to be *recursive*. We assume that rules contain no constant symbols.[1] We also assume that all rules are *range-restricted*, i.e., that the variable in the head of a rule appears in some atom in its body. This is a commonly made assumption that guarantees that rules are *safe* in the sense of returning finite answers when run on a finite database. The range-restrictedness assumption could be eliminated at the cost of some complications to our constructions.

[1]This assumption appears to be essential to our proof, but we do not know if it is necessary for the result. In other contexts, constants in Datalog rules act in a way that is similar to increasing the arity of the predicates. See [7] for an example of a construction for the elimination of constants that illustrates this point.

A *Datalog program*, henceforth simply "program", is a finite set of rules. We denote programs by Π. A program is said to be *linear* if all its rules have at most one IDB atom in their body. (There exists in the literature an alternate definition of linearity (e.g. in [2, 11]), according to which a program is linear if each rule has its head mutually recursive with at most one atom in its body. In general, the two definitions are equivalent, as it is possible to transform a program linear according to the alternate definition into an equivalent program linear in our sense. However, the transformation is at the cost of additional arguments in some predicates, so does not work for monadic programs. Our definition is the one used in the literature on boundedness [3].)

A *database* will be a finite set of atoms in the intensional and extensional database predicates. Databases are usually taken to consist of *ground* atoms, in which all arguments of atoms are constants. For technical reasons, it is convenient for us to generalize this slightly to allow atoms containing variables. From the point of view of the semantics of programs, such variables will be treated precisely as constants are in the usual definitions.

A *substitution* is a mapping θ from variables to variables and constants. If A is an atom and θ is a substitution, we write $A\theta$ for the atom obtained from A by replacing each occurrence of each variable X on which θ is defined by $\theta(X)$. If S is a set of atoms we write $S\theta$ for $\{A\theta \mid A \in S\}$.

If D is a database in the intensional and extensional predicates and Π is a program, then we write $\Pi(D)$ for the union of D with the set of all atoms A' such that there exists a rule $A{:}-B$ in Π and a substitution θ for which $B\theta \subseteq D$ and $A' = A\theta$. We write $\Pi^n(D)$ for the result of iterating this operation n times. Formally, $\Pi^0(D) = D$ and $\Pi^{n+1}(D) = \Pi(\Pi^n(D))$. We also write $\Pi^\infty(D)$ for $\bigcup_{n=0}^\infty \Pi^n(D)$. If Q is an intensional predicate of Π and n is either a natural number or ∞ then we write $Q^n_\Pi(D)$ for the set of all atoms in $\Pi^n(D)$ with predicate Q.

Note that since only constants and variables in D can occur in $\Pi^\infty(D)$, the latter is equal to $\Pi^N(D)$ for some finite N. However, this number N may depend on the size of D. A program Π is said to be *bounded* if there exists a constant N such that $\Pi^N(D) = \Pi^\infty(D)$ for all extensional databases D. A more refined notion of boundedness is *predicate boundedness.* The predicate Q is said to be bounded in the program Π if there exists a constant N such that $Q_\Pi^N(D) = Q_\Pi^\infty(D)$ for all extensional databases D.

3 Main Result

The remainder of the paper is devoted to a proof of the following:

Theorem 1 *The problem of determining, given a predicate Q in a linear monadic Datalog program Π, whether Q is bounded in Π is in PSPACE.*

The structure of the proof is roughly as follows. We first recall from the literature the idea of an *expansion* of a Datalog program, that enables such a program to be characterized as a (possibly infinite) disjunction of conjunctive queries. We next introduce another idea from the prior literature, *connectedness* of program, and develop a characterization of predicate boundedness (Lemma 6) that reduces this problem to one dealing with expansions of connected programs. Next, we present a result (Lemma 7) that states that to solve the simplified problem it suffices to focus on sequences of expansions having a certain simple form, and develop as part of the proof of this result a polynomial size data structure that helps to characterize the progress being made towards a proof of boundedness by a sequence of expansions having this form.

We begin with some definitions and several lemmas relating to expansions. A *query* will be a finite set of atoms, possibly containing contants. In case a query is a singleton set $\{A\}$ we sometimes denote it by the atom A. If R is a query then a database D is said to *realize* R if there is a substitution θ for the variables of R such that $R\theta$ is a subset of D. If D does not realize R then we say that D *omits* R. We will also say that a database D realizes (omits) a query R *with respect to* a program Π if $\Pi^\infty(D)$ realizes (respectively, omits) R. We say D realizes a set of queries Γ (with respect to a program Π) if D realizes *some* element of Γ

(with respect to Π.) Conversely, D *omits* Γ (with respect to Π) if D realizes *no* element of Γ (with respect to Π.)

Datalog queries are known to be expressible as (possibly infinite) disjunctions of conjunctive formulae. The disjuncts are obtained as follows. (See, e.g., [11] for more detail and examples.) Let R be a set of atoms in the intensional and extensional predicates of a program Π. Each *expansion of R by* Π will be a set of atoms in the extensional predicates. If R contains only extensional predicates, then the only expansion is R itself, and we say this expansion has height 0. Suppose $R = R' \cup \{Q_1(X_1), \ldots, Q_k(X_k)\}$ where each Q_i is an IDB predicate and R' contains only EDB predicates. For each i let $Q_i(X_i)\colon -B_i(X_i, \mathbf{Y}_i)$ be an instance of a rule of Π such that the variables $\mathbf{Y}_i$ of the body do not occur in R or any B_j for $j \neq i$. Then each expansion of $R' \cup \{B_1(X_1, \mathbf{Y}_1), \ldots, B_k(X_k, \mathbf{Y}_k)\}$ of height n is an expansion of R of height $n+1$. We write $R * \Pi$ for the set of expansions of R by Π. The following result, a slight variant of results of Naughton [8] relates expansions to the semantics of programs. (It follows from this result that a query is equivalent to the possibly infinite disjunction of its expansions.)

Lemma 1 *If D is a database then $\Pi^n(D)$ realizes R iff there exists an expansion S of R by Π of height less than or equal to n such that D realizes S.*

Since we permit variables to occur in databases, each expansion of a set of atoms R may be considered to be an extensional database. A straightforward induction suffices to see that if S is an expansion of R by Π, of height n, then $\Pi^n(S)$ realizes R.

Lemma 2 *Let R, S and T be sets of atoms. If T realizes S and $\Pi^n(S)$ realizes R then $\Pi^n(T)$ realizes R.*

Proof: Suppose T realizes S and $\Pi^n(S)$ realizes R. By Lemma 1, there exists an expansion S' of R by Π of height at most n, such that S realizes S'. Since T realizes S it follows that T realizes S', so, again by Lemma 1, $\Pi^n(T)$ realizes R. $\Box$

We next recall some notions from [3], the *variable graph*, the *triggers* of a rule, *connectedness* of rules and programs, and a program transformation producing connected programs. These ideas will be used to give a new characterization of predicate boundedness.

Given a set of atoms R, define the graph $G_R = (V, E)$ with vertices V equal to the set of constants or variables which occur in R, and edge set E defined by $\{X, Y\} \in E$ if and only if there exists an atom in R that contains both X and Y. If $U \subseteq V$ is a connected component of G_R, then we say that the set of atoms in R containing variables or constants in U is a connected component of R. If R has just one component, we say it is *connected.*

If r is a rule of the form $Q(X)\colon -B$, define the graph G_r to be the graph of the set of atoms B. Let R_i for $i = 0 \ldots m$ be the connected components of B, where R_0 is the connected component containing the variable X in the head of the rule. (There must be such a component because rules have been assumed to be range-restricted.) Then we may write the rule r as $Q(X)\colon -R_0, R_1, \ldots, R_m$, where the R_i share no variables. The R_i for $i \geq 1$, if they exist, are called the *triggers* of the rule. A program is said to be *connected* if every rule has an empty set of triggers.

Example 1 *The rule* $Q(X)\colon -A(X, Y), Q(Y), B(Z, T), B(T, U), A(V, V)$ *has variable graph with vertices* X, Y, Z, T, U, V *and edges* $\{X, Y\}$, $\{Z, T\}$, $\{T, U\}$, *plus an edge from each vertex to itself. The connected components are* $\{X, Y\}$, $\{Z, T, U\}$, *and* $\{V\}$. *Consequently, this rule has* $R_0 = \{A(X, Y), Q(Y)\}$ *and triggers* $\{B(Z, T), B(T, U)\}$ *and* $\{A(V, V)\}$.

If D_1 and D_2 are databases then the disjoint union $D_1 \oplus D_2$ is the database obtained by first renaming constants and variables so that the two databases share no constants or variables, then taking the union. Renaming should be such that distinct variables and constants remain distinct. If $\mathcal{D}_1, \mathcal{D}_2$ are sets of databases then $\mathcal{D}_1 \oplus \mathcal{D}_2$ is the set of all databases obtained by taking disjoint unions of databases in $\mathcal{D}_1, \mathcal{D}_2$. Strictly, we should specify a renaming function in these definitions, but our results will be largely independent of the choice of this function. We will, however, occassionally need to ensure that a specific constant is not renamed, and will explicitly indicate when this is the case. We say that renaming *preserves* a constant a if a itself is not renamed, and no other constant or variable has its name changed to a. Note that it is possible to take a disjoint union of databases, preserving a constant a, iff a occurs in at most one of the databases. The proof of the following lemma is immediate from the observation that the image of a connected set of atoms under a substitution is connected.

Lemma 3 *Let R be a nonempty connected query (possibly containing constants), and let $\{D_i \mid i \in I\}$ be databases. Provided renaming preserves constants in R, the disjoint union $\bigoplus_{i\in I} D_i$ realizes R if and only if D_i realizes R for some $i \in I$.*

Lemma 4 *Let Π be a connected program and let $\{D_i \mid i \in I\}$ be databases, not sharing any constants or variables. Then $\Pi^n(\bigoplus_{i\in I} D_i) = \bigoplus_{i\in I} \Pi^n(D_i)$, where n is either a natural number or ∞.*

Proof: By induction on n. The base case is trivial. For the inductive case, suppose $Q(X) : -B(X, \mathbf{Y})$ is a rule of Π, and c is a constant or variable in D_k. Then since B is connected, and non-empty by the range-restrictedness assumption, $\bigoplus_{i\in I} D_i$ realizes $B(c, \mathbf{Y})$ iff D_k realizes $B(c, \mathbf{Y})$. It follows that $Q(c) \in \Pi(\bigoplus_{i\in I} D_i)$ iff $Q(c) \in \Pi(D_k)$. This observation, together with the fact that the $\Pi^n(D_i)$ are disjoint, suffices to establish the inductive case. □

If Π is a program and Γ is a subset of the set of triggers of Π, define Π_Γ to be the program obtained by deleting from Π all rules containing triggers not in Γ and erasing the triggers from the remaining rules. That is, Π_Γ contains a rule $Q(X) : -R_0$ just when there is a rule

$$Q(X) : -R_0, R_1, \ldots, R_m$$

in Π such that $\{R_1, \ldots, R_m\}$ is a subset of Γ. Note that Π_Γ is a connected program. Moreover, if Π is range-restricted, then so is Π_Γ. The proof of the following lemma is straightforward from the semantics of programs.

Lemma 5 *Let Π be a program and Γ a subset of the triggers of Π. Let Γ' be the set of the remaining triggers of Π. Then for all databases D, and numbers n and N,* *(a) if $\Pi^n(D)$ omits Γ' then $\Pi^n(D) \subseteq \Pi^n_\Gamma(D)$, and*

(b) if $\Pi^N(D)$ realizes all triggers in Γ then $\Pi^n_\Gamma(D) \subseteq \Pi^{N+n}(D)$, and

(c) if $\Pi^\infty(D)$ realizes all triggers in Γ but omits Γ', then $\Pi^\infty(D) = \Pi^\infty_\Gamma(D)$.

(d) if $\Pi^n_\Gamma(D)$ omits Γ' then $\Pi^n(D) \subseteq \Pi^n_\Gamma(D)$, and

Proof: For (a), note that if D omits Γ' then no rule of Π containing a trigger in Γ' generates a new fact in $\Pi(D)$. Moreover, any new fact that may be generated using a rule of Π may also be generated using the corresponding rule of Π_Γ, since this rule requires the realization of a smaller query. The result now follows by a straightforward induction.

The proof of (b) is also by induction on n. The base case $n = 0$ is trivial. For the inductive case, note that if $\Pi^n_\Gamma(D) \subseteq \Pi^{N+n}(D)$ and the latter realizes all triggers in Γ, then any fact that may be generated from $\Pi^n_\Gamma(D)$ using a rule of Π_Γ may also be generated using the corresponding rule of Π.

Statement (c) follows from (a) and (b).

The proof of (d) is by induction on n. The base case is trivial. For the inductive step, suppose that $\Pi^{n+1}_\Gamma(D)$ omits Γ'. Then also $\Pi^n_\Gamma(D)$ omits Γ', so $\Pi^n(D) \subseteq \Pi^n_\Gamma(D)$ by the induction hypothesis. We show each atom $P(t)$ in $\Pi^{n+1}(D)$ is also in $\Pi^{n+1}_\Gamma(D)$. Suppose $P(t)$ is a new atom in $\Pi^{n+1}(D)$ produced by firing a rule $P(X){:}\ -R_0(X), R_1, \ldots, R_k$. Then $R_0(t)$ and the $R_1, \ldots, R_k$ are realized in $\Pi^n(D)$, hence in $\Pi^n_\Gamma(D)$. It follows that $R_1, \ldots, R_k$ are not in Γ', so these triggers must be in Γ. Hence $P(X){:}\ -R_0(X)$ is a rule in Π_Γ. Since $\Pi^n_\Gamma(D)$ realizes $R_0(t)$, it follows that $P(t)$ is in $\Pi^{n+1}_\Gamma(D)$. □

The programs Π_Γ are used in by Cosmadakis et. al. [3] to give a characterization of boundedness of monadic programs Π, leading to a proof of decidability. For predicate boundedness, they use alternate techniques, which fail to yield a tight bound on the complexity of this problem. To obtain a tight bound, we will develop a characterization of predicate boundedness along the lines of their characterization for boundedness.

Let D be a database, R a query and Π a program. Define the *order* of R on D with respect to Π, denoted $ord(R, D, \Pi)$, to be the least number n such that $\Pi^n(D)$ realizes R. If $\Pi^\infty(D)$ does not realize R we let $ord(R, D, \Pi) = \infty$. The order $ord(\Gamma, D, \Pi)$ of a set of queries Γ is the minimum of the $ord(R, D, \Pi)$ for $R \in \Gamma$, i.e., the least number such that *some* query in Γ is realized.

Clearly, a predicate Q is bounded in a program Π if there exists a natural number k such that $ord(\{Q(a)\}, D, \Pi) < k$ for every extensional database D and constant a such that $\Pi^\infty(D)$ realizes $\{Q(a)\}$. We will need a more informative characterization than this, however.

Let Γ_1, Γ_2 be sets of queries and Π_1, Π_2 be programs. Say that a set $\mathcal{D}$ of databases has Γ_1 *unbounded with respect to* Π_1*, simultaneously*[2] *omitting* Γ_2 *with respect to* Π_2 if there exists an infinite sequence $D_1, D_2, \ldots$ in $\mathcal{D}$ such that

1. the sequence of numbers $ord(\Gamma_1, D_k, \Pi_1)$ is unbounded, and

2. for all k, the database D_k omits Γ_2 with respect to Π_2.

Note that this definition is satisfied if there exists a database $D \in \mathcal{D}$ such that D omits Γ_1 with respect to Π_1 and also omits Γ_2 with respect to Π_2. We will also say simply that $\mathcal{D}$ has Γ_1 *unbounded with respect to* Π_1 if there exists a sequence of databases in D satisfying just condition (1).

Example 2 *Consider the programs*

$$\Pi_1 = \left\{ \begin{array}{lcl} Q(X) & :- & A(X), R(X) \\ R(X) & :- & A(X), B(X,Y), R(Y) \\ R(X) & :- & C(X) \end{array} \right. \qquad \Pi_2 = \left\{ \begin{array}{lcl} S(X) & :- & C(X), T(X) \\ T(X) & :- & C(X), B(X,Y), T(Y) \\ T(X) & :- & A(X) \end{array} \right.$$

Here $Q(X)$ holds when there is a path through the relation B from a fact $A(X)$ to a C fact, and $S(X)$ holds when there exists such a path from a fact $C(X)$ to an A fact. The set of all databases of the form

$$\{A(c_1), B(c_1, c_2), \ldots B(c_n, c_{n+1}), C(c_{n+1})\}$$

[2] It is not possible to split this definition into two parts: "simultaneously" here refers to the requirement that the same sequence of databases is used to satisfy both parts of the definition.

for some $n \geq 1$ *has* $\{\{Q(X)\}\}$ *unbounded with respect to* Π_1*, simultaneously omitting* $\{\{S(X)\}\}$ *with respect to* Π_2*.*

The following result reduces predicate boundedness for arbitrary programs to a problem dealing only with connected programs.

Lemma 6 *Let* Π *be a monadic program. Then the predicate* Q *is unbounded in* Π *if and only if there is a partition* $(\Gamma_1, \{R_1, \ldots, R_n\}, \Gamma_2)$ *of the triggers of* Π *such that*

1. *the set of databases*

$$Q(a) * \Pi_{\Gamma_1 \cup \{R_1,\ldots,R_n\}} \oplus \bigoplus_{i=1}^{n} R_i * \Pi_{\Gamma_1 \cup \{R_1,\ldots,R_{i-1}\}}$$

where a *is a constant left invariant under renaming, has* $\{R_1, \ldots, R_n, Q(a)\}$ *unbounded with respect to* Π_{Γ_1}*, simultaneously omitting* Γ_2 *with respect to* $\Pi_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}$*, and*

2. *there exists a database* D *such that* $\Pi^\infty(D)$ *realizes all elements of* Γ_1 *but omits* $\{R_1, \ldots, R_n\}$*, and* $\Pi^\infty_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}(D)$ *omits* Γ_2*.*

Proof: For the implication from right to left, let $D_1, D_2, \ldots$ be the sequence of databases witnessing the first condition and D the database witnessing the second. The databases D_k are obtained by taking the disjoint union of certain expansions of the queries $Q(a)$ and R_i. For all $k > 0$, put $D'_k = D_k \oplus D$, with a invariant under renaming in D_k. (That is, if a occurs in D, we rename it before taking the disjoint union.) We will show that the databases D'_k witness the fact that Q is unbounded in Π.

We first show that each D'_k realizes $Q(a)$ with respect to Π. Each query $R \in \Gamma_2$ is connected, as is the program $\Pi_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}$. Thus, by Lemma 3 and Lemma 4, if $\Pi^\infty_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}(D'_k)$ realizes R then either $\Pi^\infty_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}(D_k)$ realizes R or $\Pi^\infty_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}(D)$ realizes R, both of which are impossible. Thus, all the databases D'_k omit Γ_2 with respect to $\Pi_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}$. Since $\Pi^\infty(D)$ realizes all Γ_1, for some N_0 we have that $\Pi^{N_0}(D)$ realizes all Γ_1. Fix some k. For $1 \leq i \leq n$, let N_i be the height of the expansion of R_i used to construct D_k, and

N_{n+1} the height of the expansion of $Q(a)$ used to construct D_k. Then it follows using $n+1$ applications of Lemma 5(b) that $\Pi^{N_0+N_1+\ldots+N_{n+1}}(D'_k)$ realizes $Q(a)$, hence $\Pi^\infty(D'_k)$ realizes $Q(a)$ for all k.

We next show that if $\Pi^n_{\Gamma_1}(D_k)$ omits $\{Q(a), R_1, \ldots, R_n\}$ then so does $\Pi^n(D'_k)$. The claim that the set of D'_k has $Q(a)$ unbounded with respect to Π is then immediate, since we may make the numbers n such that $\Pi^n_{\Gamma_1}(D_k)$ omits $\{Q(a), R_1, \ldots, R_n\}$ arbitrarily large, by condition (1).

Suppose that that $\Pi^n_{\Gamma_1}(D_k)$ omits $\{Q(a), R_1, \ldots, R_n\}$. Note that it follows from condition (2) and Lemma 5 that $\Pi^n_{\Gamma_1}(D) \subseteq \Pi^\infty(D)$. Using the fact that all queries in $\{Q(a), R_1, \ldots, R_n\}$ are connected, it follows that $\Pi^n_{\Gamma_1}(D_k) \oplus \Pi^n_{\Gamma_1}(D)$ omits $\{Q(a), R_1, \ldots, R_n\}$. (We assume here and below that renaming preserves a in the leftmost term of a sum.) Thus, using connectedness of Π_{Γ_1}, we have that $\Pi^n_{\Gamma_1}(D_k) \oplus \Pi^n_{\Gamma_1}(D) = \Pi^n_{\Gamma_1}(D'_k)$ omits $\{Q(a), R_1, \ldots, R_n\}$. As noted above, $\Pi^n_{\Gamma_1 \cup \{R_1,\ldots,R_n\}}(D'_k)$ omits Γ_2, from which it follows that the smaller database $\Pi^n_{\Gamma_1}(D'_k)$ omits Γ_2 also. Thus $\Pi^n_{\Gamma_1}(D'_k)$ omits all triggers not in Γ_1. Consequently, using Lemma 5(a), $\Pi^n(D'_k) \subseteq \Pi^n_{\Gamma_1}(D'_k)$, from which we obtain that $\Pi^n(D'_k)$ omits $\{Q(a), R_1, \ldots, R_n\}$. This completes the proof from right to left.

For the converse, assume that $D_1, D_2, \ldots$ is a sequence of databases such that $\Pi^\infty(D_k)$ realizes $Q(a)$ for all k, but the numbers $ord(Q(a), D_k, \Pi)$ are strictly increasing. Some set of triggers Γ_0 is infinitely often the set of all triggers realized by $\Pi^\infty(D_k)$. Without loss of generality, the set of triggers realized by $\Pi^\infty(D_k)$ is always Γ_0, so that the set Γ_2 of all triggers not in Γ_0 is never realized. Let Γ_1 be the set of triggers $R \in \Gamma_0$ such that the $ord(R, D_k, \Pi)$ are bounded, and let $\Gamma_0 \setminus \Gamma_1 = \{R_1, \ldots, R_n\}$. Without loss of generality, the numbers $ord(\{R_1, \ldots, R_n\}, D_k, \Pi)$ are strictly increasing also. There exists an N such that for all k, $\Pi^N(D_k)$ realizes all elements of Γ_1, i.e., some expansion, of height less than or equal to N, of the disjoint union of the elements of Γ_1 is realized in D_k. Since there are only finitely many such expansions, some such expansion D is realized in D_k infinitely often. Because D is finite, there exists an M such that $\Pi^\infty(D) = \Pi^M(D)$. From this it follows that $\Pi^\infty(D)$ realizes no element of $\{R_1, \ldots, R_n\}$, since otherwise we would have $ord(\{R_1, \ldots, R_n\}, D_k, \Pi) < M$ infinitely often, by Lemma 1. Finally, it follows from the fact that D_k realizes D and the fact that $\Pi^\infty(D_k)$ realizes all elements of $\Gamma_1 \cup \{R_1, \ldots, R_n\}$ but omits Γ_2 that $\Pi^\infty_{\Gamma_1 \cup \{R_1, \ldots, R_n\}}(D)$ omits Γ_2, using Lemma 2. Thus D satisfies the second condition of the lemma.

Without loss of generality we can assume that for all $i < n$ and all k we have $ord(R_i, D_k, \Pi) \leq ord(R_{i+1}, D_k, \Pi)$. This implies that for all i and k there is an expansion $E_{i,k}$ in $R_i * \Pi_{\Gamma_1 \cup \{R_1, \ldots, R_{i-1}\}}$ realized in D_k, and similarly there is an expansion $E_{n+1,k}$ in $Q(a) * \Pi_{\Gamma_1 \cup \{R_1, \ldots, R_n\}}$ realized in D_k. Let $E_k = \oplus_{i=1 \ldots n+1} E_{i,k}$, preserving a. Then the sequence of databases E_k is of the form required for condition 1 of the lemma, and because it is realized in D_k it has $\{R_1, \ldots, R_n, Q(a)\}$ unbounded with respect to Π_{Γ_1} and simultaneously omits Γ_2 with respect to $\Pi_{\Gamma_1 \cup \{R_1, \ldots, R_n\}}$. $\square$

This result resembles Propositions 5.3 and 5.4 of [3], but provides a somewhat more informative characterization. The second condition of the lemma is decidable in PSPACE for monadic programs, using the automata-theoretic techniques of [3]. Thus to show that boundedness of linear monadic Datalog is in PSPACE it suffices to show (by Lemma 6) that the following is in PSPACE:

> **Input:** A connected query I, connected linear monadic programs Π_1, Π_2, Π_3 and sets Γ_1, Γ_2 of connected queries, such that I and the queries in Γ_1 and Γ_2 contain at most one occurrence of an IDB predicate.
> **Question:** Does $I * \Pi_1$ have Γ_1 unbounded with respect to Π_2, simultaneously omitting Γ_2 with respect to Π_3? (*)

The reason we are able to consider just $I * \Pi_1$, rather than a disjoint union of such sets, is that the property of interest holds for a disjoint union of sets of databases just when it holds for each of the sets individually. Queries can be assumed to contain at most one occurrence of an IDB predicate because $Q(a)$ and the triggers of linear programs satisfy this condition.

For brevity we simplify the problem further by assuming that Γ_2 is empty. We explain later how this assumption may be removed. Further, we assume that Π_1 has just one defined predicate P and all recursive rules are of the form $P(X) : -R(X,Y), P(Y)$ where X and Y are distinct variables and $R(X,Y)$ is a set of extensional atoms. There is no loss of generality in these assumptions, as Π_1 may be modified so as to satisfy these conditions. Similar simplifications have previously been noted [3], but without explanation, so we now briefly justify this claim. (Note that we apply these simplifications to Π_1 only, not Π_2 or Π_3.)

We may ensure that Π_1 contains no rules making recursive calls on the head variable by a program transformation that, intuitively, eliminates redundant uses of such rules. We note first that we may assume that no rule of the form $P(X){:}-B(X), P(X)$ is used in an expansion of I by Π_1, since removing $B(X)$ from such an expansion results in an expansion of I by Π_1 in which Γ_1 has higher order, and which still omits Γ_2 with respect to Π_2 if the orginal did.

For similar reasons, we may assume that we recurse over each IDB predicate Q_i at most once with respect to each expansion variable X. Assume without loss of generality that all rules in Π_1 in which the recursive call is on the head variable have head variable X. Given IDB predicates P, Q, say that a *chain C from P to Q on variable X* is a sequence of rules

$$\begin{array}{l} P_1(X) : -B_1(X), P_2(X). \\ \quad\vdots \\ P_{n-1}(X) : -B_{n-1}(X), P_n(X). \end{array}$$

of Π_1 with $P = P_1$ and $Q = P_n$, such that P_1 and P_j are distinct predicates for all $i \neq j$. The *result* of the chain C is the set $B_C(X) = B_1(X) \cup \ldots \cup B_n(X)$.

To eliminate rules making a recursive call on the head variable, we proceed as follows:

1. Let P be a predicate for which there is a rule $P(X){:}-B(X), P'(X)$ making a recursive call on the head variable. For each chain C from P to some predicate Q, and for each rule $Q(X){:}-D(X)$ (recursive or non-recursive) that does not make a recursive call on

the head variable, add to the program the rule

$$P(X){:}\, -B_C(X), D(X)$$

2. eliminate all the rules making a recursive call on the head variable.

We apply this transformation for each recursive predicate Q. Call the resulting program Π_1'. Then it may be shown that for all expansions E in $I * \Pi_1$ there exists an expansion E' in $I * \Pi_1'$ such that E' realizes E, and vice versa. It follows that Π_1 and Π_1' are equivalent with respect to (*). We note that this transformation causes an exponential blowup in the size of the program, but that each of the rules produced has size polynomial in the size of the original program. As we discuss below, the algorithm can avoid the exponential cost by generating rules of the transformed program on the fly.

Having performed the transformation of the previous paragraph, we may ensure that Π_1 has at most one IDB predicate, by replacing each rule of the form $Q_1(X) : -R(X,Y), Q_2(Y)$ with the rule $P(X) : -Q_1^*(X), R(X,Y), Q_2^*(Y), P(Y)$, where the Q_i^* are new EDB predicates, and similarly replacing each non-recursive rule $Q_1(X){:}\, -R(X,Y)$ by the rule $P(X){:}\, -Q_1^*(X), R(X,Y)$. Simultaneously, we add to Γ_1 the query $\{Q_i^*(X), Q_j^*(X)\}$ for each pair $i \neq j$. This transformation again does not change the property (*), because any expansion of the resulting program that does not realize the queries $\{Q_i^*(X), Q_j^*(X)\}$ (equivalently, has the order of these queries equal to ∞, hence unbounded) yields an expansion of Π_1 simply by deleting the atoms of the form $Q_i^*(X)$. Note that the predicates Q_i^* do not interact in any way with Π_2, Π_3 or Γ_1, Γ_2.

Next, we note that since I is a query containing exactly one intensional atom, and Π_1 is a connected linear monadic program satisfying the simplifying assumptions just made, the expansions of I by Π_1 are of a particularly simple form. If $I = R'(Z_1) \cup \{P(Z_1)\}$ where $P(Z_1)$ is the intensional atom in I then each expansion may be written as

$$R'(Z_1) \cup R_1(Z_1, Z_2) \cup R_2(Z_2, Z_3) \cup \ldots \cup R_n(Z_n)$$

where for $i = 1 \ldots n-1$ there is a recursive rule $P(X) \colon - R_i(X,Y), P(Y)$ of Π_1 and the final component R_n is drawn from an initialization rule $P(X) \colon - R_n(X)$ of Π_1. Note that if $i < j$ then R_i and R_j share variables only if $j = i+1$, and in this case the only variable in common is Z_{i+1}. We call this variable the *join variable* between R_i and R_{i+1}. Note also that the expansion is connected, by connectedness of Π_1.

In the following, we let A be the set of recursive rules of Π_1 and B the set of initialization rules of Π_1. Each expansion of I by Π_1 can then be represented by a string in the regular set $\{I\}A^*B$. Conversely, given a string in this set, we obtain an expansion of I by Π_1, unique up to renaming of variables. Similarly, a string in $A^*B \cup A^*$ corresponds to an extensional database unique up to variable renaming. In the sequel, we will use the term *string* for a string of one of the above forms. We will also identify a string σ with the corresponding database. Thus, by a *variable* of σ we will mean a variable of the corresponding database. We assume some canonical way of naming the variables of each string.

Let I be a connected query containing at most one intensional atom, let Π_1, Π_2 be connected programs, with Π_1 satisfying the simplifying assumptions above, and let Γ_1 be a set of connected queries containing at most one intensional atom.

Define a *solution* of the problem $(I, \Pi_1, \Pi_2, \Gamma_1)$ to be an infinite sequence (σ_n) of strings in A^*B such that for all n, the database $\Pi_2^n(I\sigma_n)$ does not realize Γ_1. Note that this implies that $ord(\Gamma_1, I\sigma_n, \Pi_2) > n$ for all n, so $R * \Pi_1$ has Γ_1 unbounded with respect to Π_2. If Π_1, Π_2, Γ_1 are clear from the context we will speak of a solution of the problem I.

The following result states that if solutions exist, there exist solutions of a simple form. This result, and its proof, will be the key to the PSPACE algorithm to follow.

Lemma 7 *If $I * \Pi_1$ has Γ_1 unbounded with respect to Π_2 then there exist strings $\alpha_1, \ldots, \alpha_{r+1}$ and $\beta_1, \ldots, \beta_r$ such that the sequence $\sigma_n = \alpha_1 \beta_1^n \alpha_2 \beta_2^n \ldots \alpha_r \beta_r^n \alpha_{r+1}$ is a solution to $(I, \Pi_1, \Pi_2, \Gamma_1)$.*

The idea underlying the proof of this result is that to capture the contribution that a string makes to the construction of a solution, it suffices to focus on a small initial segment and small final segment, and to abstract the way that the remainder of the string interacts with the program Π_2 into a finite number of different types. We will need several definitions before we can state the nature of these types.

First, we make an observation that enables us to focus on small segments of a string. If R is a query, define the distance between two vertices in the graph G_R defined above to be the length of the shortest path between these vertices in G_R. Define the *diameter* of a connected query R to be the maximal distance between vertices in the graph G_R. The diameter of a program will be the maximal diameter of the body of any rule.

Lemma 8 *Suppose R is a connected query of diameter d. If R is realized in a string σ, then it is realized in a substring of σ of length less than or equal to d.*

Proof: The image of a connected set of atoms of diameter d under a substitution is a connected set of atoms of diameter as most d. Such a set of atoms must lie within at most d contiguous elements of the string. $\Box$

For the next definition, it is convenient to define a database D to S-realize a query R, where S is a subset of the set of variables occurring in R, when D realizes R by means of a substitution that leaves the variables in S invariant.

Part of the interaction of a string with the program Π_2 is captured by the following notion. If σ is a string then define a *path* in σ of length m to be a sequence $(Q_1, X_1), \ldots, (Q_m, X_m)$ of pairs, each consisting of a defined predicate Q_i of Π_2 and a variable X_i of σ, such that for all $i = 1, \ldots, m-1$ there exists a rule $Q_{i+1}(X) : -R(X,Y), Q_i(Y)$ of Π_2 such that σ $\{X_i, X_{i+1}\}$-realizes $R(X_{i+1}, X_i)$. Intuitively, such a path captures dependencies between IDB atoms in computing the fixpoint with respect to Π_2. Note that if such a path exists and $Q_1(X_1) \in \Pi_2^n(\sigma)$ then $Q_m(X_m) \in \Pi_2^{n+m}(\sigma)$.

We now combine the notion of path and the observation that realization of connected queries depends on small segments of strings into some definitions whose aim is to capture the contribution that a string makes, when combined with other strings, to the realization of queries in Γ_1 with respect to Π_2. Let d be the maximum of the diameter of the program

Π_2 and the diameters of the queries in Γ_1. Let σ_l be the prefix of σ of length d and σ_r the suffix of σ of length d. (If σ has length less than d let $\sigma_r = \sigma_l = \sigma$.) Recall that we are dealing with linear rules, so each query in Γ_1 contains at most one IDB atom. The *arcs* of σ are defined as follows. There are two types of arc, *simple* arcs and *enumerated* arcs, the latter bearing more information about σ.

1. A *simple head arc* of σ will be a pair (Q_1, X_1) consisting of an intensional predicate Q_1 of Π_2 and a variable X_1 of σ_l or σ_r such that there exists a path $(Q_1, X_1), \ldots, (Q_m, X_m)$ of length m in σ and a query $R(Y) \cup \{Q_m(Y)\}$ in Γ_1 such that σ $\{X_m\}$-realizes $R(X_m)$. An *enumerated head arc* is a triple (Q_1, X_1, m) satisfying the above condition.

2. A *simple tail arc (enumerated tail arc)* of σ will be a pair (Q_m, X_m) (respectively, a triple (Q_m, X_m, m)) such that X_m is a variable of σ_l or σ_r and there exists a path $(Q_1, X_1), \ldots, (Q_m, X_m)$ in σ and an initialization rule $Q_1(X) : -R(X)$ of Π_2 such that σ $\{X_1\}$-realizes $R(X_1)$.

3. A *simple chain arc (enumerated chain arc)* of σ will be a tuple (Q_1, X_1, Q_m, X_m) (respectively, a tuple (Q_1, X_1, Q_m, X_m, m)) such that X_1, X_m are variables of σ_l or σ_r and there exists a path $(Q_1, X_1), \ldots, (Q_m, X_m)$ in σ. A chain arc will be said to *span* σ if the variables X_1, X_m are from different sides of σ.

In the case of an enumerated arc, we call the component m in the tuple the *weight* of the arc.

The contribution a string σ makes to realization of queries in Γ_1 with respect to Π_2 is by a construct which we call its *type*. The type of a string includes its arcs. As with arcs, there are two kinds of type, simple and enumerated. Simple types will be used in the algorithm for deciding boundedness to be described below; enumerated types are required only for the proof of its correctness. Moreover, we distinguish between *one-sided* and *two-sided* types. The former correspond to strings comprising initial segments of solutions, which will be concatenated with other strings only to the right. Accordingly, we do not need the arcs associated with variables in σ_l in this case. Two-sided types correspond to strings which are concatenated with other strings to both the left and the right.

The *simple two-sided type* of a string σ will be the tuple $(\sigma_l, \sigma_r, Head, Tail, Chain)$ where $Head$ is the set of all simple head arcs of σ, $Tail$ the set of all simple tail arcs and $Chain$ the set of all simple chain arcs. The *enumerated two-sided type* of σ will be a tuple $(\sigma_l, \sigma_r, EHead, ETail, EChain)$. Here $EHead$ is the set of all enumerated head arcs (Q, X, m) of σ such that (Q, X, n) is a head arc of σ implies $m \leq n$. The sets $ETAil, EChain$ are defined similarly. That is, to obtain the enumerated type we add to each simple arc the minimal length of a path witnessing that arc.

The *one-sided type* of a string σ will be a tuple $(\sigma_r, Head_r, Tail_r, Chain_r)$, where we take only those arcs containing only variables of σ_r. That is, an arc (Q_1, X_1, Q_m, X_m) in $Chain$ is in $Chain_r$ if and only if X_1, X_m are both variables of σ_r; for head and tail arcs (Q, X) we require that X is a variable of σ_r. The *one-sided enumerated type* is obtained by adding to these arcs the length of the shortest witness. We write $type_i(\sigma)$ and $etype_i(\sigma)$ for the i-sided simple type and the i-sided enumerated type of σ, respectively.

For both one-sided and two-sided types, we call the type of the null string the *null type*. The set of all possible one-sided or two-sided simple types is finite, up to renaming of variables, and each such type can be represented in polynomial space.

Lemma 9 *If σ_1, σ_2 are strings, it is possible to determine the simple two-sided type of $\sigma_1\sigma_2$ from the simple two-sided types of σ_1, and σ_2. The computation can be done in PSPACE. Similarly the one-sided type of $\sigma_1\sigma_2$ may be determined in PSPACE from the one-sided type of σ_1 and the two-sided type of σ_2.*

Proof: We show how the two-sided type of $\sigma_1\sigma_2$ may be determined from the two-sided types of σ_1 and σ_2. The construction is illustrated in Figure 1. The argument for computing the one-sided type is similar. We assume that the variables of σ_1, σ_2 and their types have been renamed so as to agree only on the join variable between σ_1 and σ_2. Note first that the

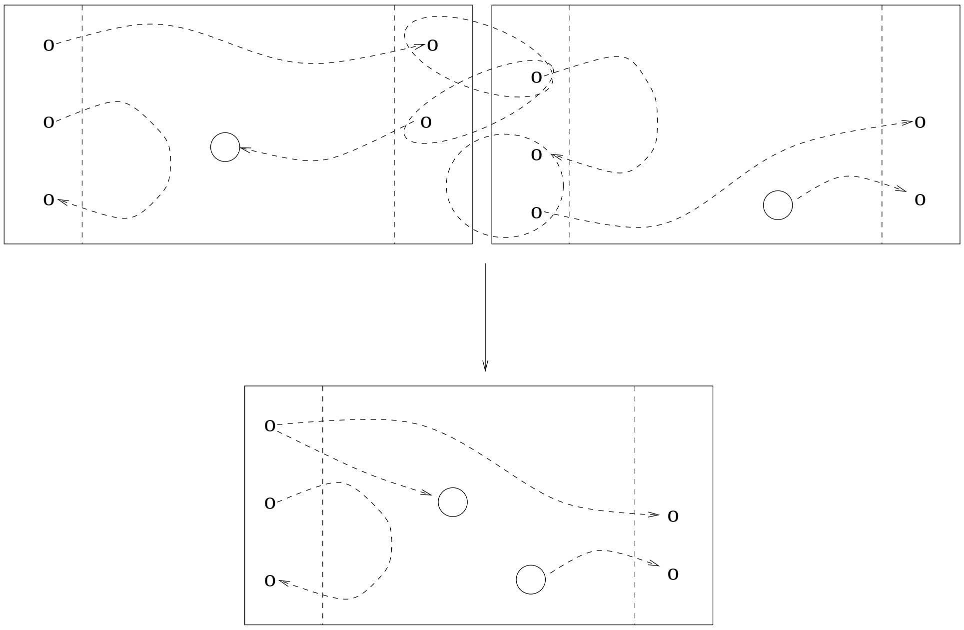

Figure 1: Composing two-sided types

maximal prefix and suffix of $\sigma_1\sigma_2$ of length at most d can be determined from the prefixes and suffices contained in the types of σ_1 and σ_2.

We will show how the head arcs of $\sigma_1\sigma_2$ may be determined from the two-sided types of σ_1 and σ_2. Intuitively, the paths witnessing a head arc may be decomposed into a sequence of arcs of σ_1 and σ_2. Whenever a path witnessing the head arc has a segment that crosses over between σ_1 and σ_2 there is enough information in $(\sigma_1)_r(\sigma_2)_l$ to capture such a path segment. Specifically, we prove that for every head arc (Q_1, X_1) of $\sigma_1\sigma_2$ there exists a finite sequence $(Q_1, X_1, Q_2, X_2), (Q_2, X_2, Q_3, X_3), \ldots, (Q_{m-1}, X_{m-1}, Q_m, X_m), H$ where each element of the sequence (except H) is a chain arcs of σ_1 or σ_2, or a chain arc of the string $(\sigma_1)_r(\sigma_2)_l$. The last element H is either a head arc (Q_m, X_m) of σ_1 or σ_2, or $R(X_m)$, where $R(Y) \cup \{Q_m(Y)\}$ is a query in Γ_1 such that $R(X_m)$ is $\{X_m\}$-realized in $(\sigma_1)_r(\sigma_2)_l$. The converse of this claim, that every such sequence (with X_1 in $(\sigma_1\sigma_2)_l$ or $(\sigma_1\sigma_2)_r$) corresponds to a head arc of $\sigma_1\sigma_2$, is readily seen to be true; the proof is left to the reader.

Suppose (Q_1, X_1) is a head arc of $\sigma_1\sigma_2$. Then there exists a sequence $(Q_1, X_1), \ldots, (Q_m, X_m)$ of length m and rules $Q_{i+1}(X) :- R(X, Y), Q_i(Y)$ of Π_2, and substitutions θ_i invariant on $\{X_i, X_{i+1}\}$ such that for $i = 1, \ldots, m-1$ we have $R(X_{i+1}, X_i)\theta_i \subseteq \sigma_1\sigma_2$. In addition there exists a query $R(Y) \cup \{Q_m(Y)\}$ in Γ_1 and a substitution θ invariant on X_m such that $R(X_m)\theta \subseteq \sigma_1\sigma_2$. We assume that X_1 occurs on the left side of $\sigma_1\sigma_2$, the other case is similar. We prove by induction on m that a sequence of arcs of the structure claimed in the previous paragraph exists.

There are two possibilities: either all the variables in the range of θ and the θ_i occur in σ_1, or some such variable occurs in σ_2 and is not the join variable between σ_1 and σ_2. In the first case, $H = (Q_1, X_1)$ is an enumerated head arc of σ_1, and we have established our claim. In the second case, let Z be a variable that is mapped to σ_2. Here again we have two possibilities.

Suppose first that Z occurs in $R(X_m)$, and that no variable in any R_i is mapped to σ_2. Then, in particular, X_m is mapped to σ_1. It follows using connectedness of $R(X_m)$ and choice of d that $R(X_m)$ is $\{X_m\}$-realized in $(\sigma_1)_r(\sigma_2)_l$. Thus, in this case, the sequence $(Q_1, X_1, Q_m, X_m), R(X_m)$ has the claimed structure.

In the other case, choose the least k such that a variable Z in $R_k(X_k, X_k + 1)$ is mapped to σ_2. In particular, X_k is mapped to σ_1. It follows that R_k is mapped to $(\sigma_1)_r(\sigma_2)_l$. Thus, we have that (Q_1, X_1, Q_k, X_k) is a chain arc of σ_1, and $(Q_k, X_k, Q_{k+1}, X_{k+1})$ is a chain arc of $(\sigma_1)_r(\sigma_2)_l$. Moreover, the sequence $(Q_{k+1}, X_{k+1}), \ldots, (Q_m, X_m)$ is a path in $\sigma_1\sigma_2$ and $R(X_m)$ is $\{X_m\}$-realized in $\sigma_1\sigma_2$. That is, this sequence is of the same type as the sequence with which we started, but shorter. The existence of the claimed sequence is now immediate using the inductive hypothesis.

This completes the proof that we may construct the head arcs of $\sigma_1\sigma_2$ from the types of σ_1 and σ_2 respectively. A similar argument deals with the other types of arcs. We leave the details to the reader. The computation may be done in PSPACE by means of a fixpoint computation on arcs similar to a transitive closure computation. $\square$

If T is an enumerated type, define the projection $p(T)$ to be the simple type obtained by deleting the length of each arc of T. The binary relation $\sqsubseteq$ on the set of enumerated types is defined by $T_1 \sqsubseteq T_2$ if $p(T_1) = p(T_2)$ and each arc of T_1 has length less than or equal to the corresponding arc of T_2. Clearly $\sqsubseteq$ is a partial order.

The following result shows that we may determine the two-sided enumerated type of a string from the two-sided enumerated types of its components. We do not go into the complexity of the construction since this lemma is not used in our decision procedure, only in its proof of correctness.

Lemma 10 *The two-sided enumerated type of a string $\alpha\beta$ may be determined from the two sided enumerated types of α and β. Moreover if $etype_2(\alpha_1) \sqsubseteq etype_2(\alpha_2)$, then $etype_2(\alpha_1\beta) \sqsubseteq etype_2(\alpha_2\beta)$.*

Proof: Similar to the proof of Lemma 9. Note that if we break a minimal length path witnessing an arc of $\alpha\beta$ into segments lying entirely in α, entirely in β or crossing the boundary, then these segments are all of minimal length, else we could shorten the original path. For the second claim, note that we may replace segments lying entirely in α_2 by corresponding shorter segments lying entirely in α_1. □

The type of a string also contains enough information to support a definition of computation of fixpoints of Π_2 that reflects the behavior of this computation in the string itself. If $T = (\tau, EHead, ETail, EChain)$ is a one-sided enumerated type, we extend the relation "$\Pi_2^n(D)$ realizes Γ_1" to strings $T\sigma$ in $\{T\}A^*B$ as follows. Let D_0 be the database corresponding to the string $\tau\sigma$. For $i \geq 1$ define D_i to be the database consisting of the atoms of $\Pi_2(D_{i-1})$ together with all atoms $Q(X)$ such that either

1. there exists an arc (Q, X, m) in $ETail$ and $i \geq m$, or

2. there exists an arc (Q', Y, Q, X, m) in $EChain$ such that $Q'(Y) \in D_{i-m+1}$.

Now say that $\Pi_2^n(T\sigma)$ realizes Γ_1 if either D_n realizes Γ_1 or there exists an arc (Q, X, m) in $EHead$ such that $n \geq m$ and D_{n-m+1} contains $Q(X)$. This definition may also be applied to simple types by treating a simple type as an enumerated type with all arcs of length 1.

We may also say that $\Pi_2^\infty(T\sigma)$ omits Γ_1 if $\Pi_2^n(T\sigma)$ realizes Γ_1 for no n. Similarly, we may define the order $ord(\Gamma_1, T\sigma, \Pi_2)$ to be the least n such that $\Pi_2^n(T\sigma)$ realizes Γ_1. It follows that we may also speak of solutions of the problem $(T, \Pi_1, \Pi_2, \Gamma_1)$, where T is a one-sided enumerated or simple type. Notice that if $p(T_1) = p(T_2)$ then $\Pi_2^\infty(T_1\sigma)$ omits Γ_1 iff $\Pi_2^\infty(T_2\sigma)$ omits Γ_1. Further, if $T_1 \sqsubseteq T_2$ then any solution (σ_n) of T_1 is a solution of T_2.

Remark: It can be shown that if a string α omits Γ_1 with respect to Π_2 then $\Pi_2^n(etype_1(\alpha)\sigma)$ realizes Γ_1 iff $\Pi_2^n(\alpha\sigma)$ realizes Γ_1. This result does not extend to α that realize Γ_1 with respect to Π_2, however, since the type $etype_1(\alpha)$ may "forget" that α realizes Γ_1 with respect to Π_2. We could extend the definition of type to include such memory, but we have not done so because it is unnecessary for our constructions. Our algorithm searches for strings that *omit* Γ_1, so backtracks as soon as it is detected that a string has been constructed that realizes Γ_1.

We now introduce some notions needed for the argument to follow. A partially ordered set $(X, \leq)$ is said to be *well-quasi-ordered* [6] if it has the property that there exists no infinite sequence $x_0, x_1, \ldots$ of elements of X such that $i < j$ implies not $x_i \leq x_j$. A well-quasi-ordered set is clearly well-founded, i.e. there does not exist an infinite descending sequence $x_0 > x_1 > x_2 > \ldots$. Moreover, it can be shown that for any $x \in X$ there exists a finite number of elements $y \in X$ with $y < x$.

Let the partial order $\leq$ on $\mathbf{N}^k$ be defined by $u \leq v$ if $u[i] \leq v[i]$ for all $i = 1 \ldots k$. It is known that $(\mathbf{N}^k, \leq)$ is well-quasi-ordered. Moreover, the disjoint union $(X_1 \cup X_2, \leq_1 \cup \leq_2)$ of two well-quasi-orders $(X_1, \leq_1)$ and $(X_2, \leq_2)$ is a well-quasi-order. Clearly, the set of one-sided (two-sided) enumerated types T with $p(T) = T'$ for some fixed simple type T', ordered by $\sqsubseteq$, is isomorphic to $(\mathbf{N}^k, \leq)$ for some k. It follows that set of all set of all one-sided (two-sided) enumerated types, ordered by $\sqsubseteq$, is a well-quasi-order.

If $f : X \to \mathcal{P}(X)$, say that the sequence $x_0, x_1, \ldots, x_M$ is *f-bounded* if for each $i < M$ there exists $x \in f(x_i)$ such that $x_{i+1} \leq x$.

Lemma 11 *Suppose that $(X, \leq)$ is a well-quasi-order. Let $f : X \to \mathcal{P}(X)$ map each element* *of x to a* finite *subset of X. Fix $x_0 \in X$. There exists a constant M such that for all f-bounded sequences $x_0 \dots x_M$ in X there exist indices $n < m$ with $x_n \leq x_m$.*

Proof: Suppose that for all M there exists an f-bounded sequence $x_0 \dots x_M$ such that $i < j$ implies not $x_i \leq x_j$. Note that for any fixed $v \in X$ there exist only finitely many $u \in X$ such that $u \leq x$ for some $x \in f(v)$. It follows by Koenig's Lemma that there exists an infinite sequence $x_0, x_1, \dots$ with $i < j$ implies not $x_i \leq x_j$, contradicting the assumption that the set X is well-quasi-ordered. $\Box$

In our algorithm, we would like to deal with simple types rather than with enumerated types. It is not necessarily the case that if T is an enumerated type and (σ_n) is a solution of T then (σ_n) is a solution of $p(T)$. However, this claim is almost true, as the following lemma shows.

Lemma 12 *Let T be an enumerated one-sided type. There exists a constant c such that if (σ_n) is a solution of T then there exists a solution (σ'_n) for $p(T)$ with $|\sigma'_n| \leq |\sigma_n| + c$ for all n.*

Proof: If there exists a string $\sigma \in A^*B$ such that $\Pi_2^\infty(T\sigma)$ omits Γ_1 the claim is immediate, since we may take $\sigma'_n = \sigma$ for all n and $c = |\sigma|$. Otherwise, let δ_n be the maximal length prefix of σ_n such that $\Pi_2^\infty(T\delta_n)$ omits Γ_1. Since (σ_n) is a solution, the length of δ_n is eventually greater than any constant, for otherwise the numbers $ord(\Gamma_1, T\sigma_n, \Pi_2) \leq ord(\Gamma_1, T\delta_n, \Pi_2)$ are bounded by a constant. For each n, consider the sequence of enumerated two-sided types of the increasing sequence of all prefixes of δ_n.

Note that the rate of growth of the weights of arcs in these types is bounded. Specifically, there exists a constant e such that if σ is any string and a any letter in $A \cup B$, then each arc of $etype_2(\sigma a)$ has weight at most e plus the sum of the weights of the arcs of $etype_2(\sigma)$. Let f be the function that maps a two sided enumerated type T to the set of two-sided enumerated types T' in which all arcs have weight equal to e plus the sum of the weights of the arcs of

$etype_2(\sigma)$. Then if $\sigma_n = a_1a_2a_3\ldots$ the sequence $etype_2(a_1), etype_2(a_1a_2), etype_2(a_1a_2a_3), \ldots$ is f-bounded.

It follows by Lemma 11 that there exists a constant c_1 such that for all sufficiently large n we may write $\sigma_n = \alpha_n\beta_n\gamma_n$ where $etype_2(\alpha_n) \sqsubseteq etype_2(\alpha_n\beta_n)$, the string β_n has length less than c_1 and $\Pi_2^\infty(T\alpha_n\beta_n)$ does not realize Γ_1. Using Lemma 10, we obtain $etype_2(\alpha_n\beta_n) \sqsubseteq etype_2(\alpha_n\beta_n^2)$. Continuing this argument shows $etype_2(\alpha_n\beta_n) \sqsubseteq etype_2(\alpha_n\beta_n^k)$ for all $k > 0$.

Let c_2 be the sum of the lengths of all the arcs of T. We claim that $etype_1(T\alpha_n\beta_n) \sqsubseteq etype_1(p(T)\alpha_n\beta_n^{dc_2})$. For, let a be an enumerated arc of $etype_1(p(T)\alpha_n\beta_n^{dc_2})$. If this arc is witnessed by a path realized entirely within $\alpha_n\beta_n^{dc_2}$ then there is a corresponding arc of $\alpha_n\beta_n$ which is of lower weight, so the claim is immediate. Suppose the arc a requires at least one spanning arc b of $\alpha_n\beta_n^{dc_2}$, together with some arcs of T. Then the weight of b is at least c_2 greater than the length of the corresponding spanning arc of $\alpha_n\beta_n$. This increase compensates for the possible decrease of up to c_2 due to the replacement of T by $p(T)$. It follows that $\sigma_n' = \alpha_n\beta_n^{dc_2}\gamma_n$ is a solution of $p(T)$ for sufficiently large n, with $|\sigma_n'| \leq |\sigma_n| + dc_1c_2$. $\square$

We now introduce an idea that will help to identify the "pumpable" segments β_n of solutions in the form promised by Lemma 7. If there is a string σ in A^*B such that $\Pi_2^\infty(T\sigma)$ omits Γ_1 we say that T is *directly solvable.* Suppose that T_1, T_2, T_3 are simple one-sided types and that (U, V) is a partition of the set of arcs of T_3 such that the result of deleting from T_3 the arcs in U is the type T_2. Let (σ_n) be an infinite sequence of strings and c a constant such that for all n,

1. $type_1(T_1\sigma_n) = T_3$,
2. $\Pi_2^\infty(T_1\sigma_n)$ omits Γ_1,
3. all the arcs in U have length greater than n in $etype_1(T_1\sigma_n)$,
4. all the arcs in V have length less than c in $etype_1(T_1\sigma_n)$.

In these circumstances we say that T_1 *reduces to* T_2 and write $T_1 \to T_2$.

Lemma 13 below will be used to break a solution to T into a sequence of reductions of the above form. Lemma 14 will show that each reduction can be done by pumping a single string.

Lemma 13 *If T_0 is a solvable simple one-sided type then there exists a sequence $T_0 \to T_1 \to \ldots \to T_k$ of simple one-sided types such that T_k is a directly solvable type.*

Proof: If T_0 is directly solvable there is nothing to prove. Otherwise, let (σ_n^0) be an optimal solution of T_0, i.e., for each n let σ_n^0 be a minimal length string in A^*B such that $\Pi_2^n(T_0\sigma_n^0)$ omits Γ_1. Let α_n^0 be the maximal length prefix of σ_n^0 such that $\Pi_2^\infty(T_0\alpha_n^0)$ omits Γ_1, and write $\sigma_n^0 = \alpha_n^0\beta_n^0$. Define $f_0(n)$ to be the length of α_n^0. Thus, $\alpha_n^0 = (\sigma_n^0)_1 \ldots (\sigma_n^0)_{f_0(n)}$. Since the set of simple types is finite, some simple type T_0' is infinitely often the one-sided simple type of $T_0\alpha_n^0$. Since $\Pi_2^\infty(T_0(\sigma_n^0)_1 \ldots (\sigma_n^0)_{f_0(n)+1})$ realizes Γ_1, but $\Pi_2^n(T_0\sigma_n^0)$ does not, some arcs of T_0' have unbounded weight in $etype_1(T_0\alpha_n^0)$. As all arcs in the enumerated types of a bounded length string must be bounded, we also have that $f_0(n)$ is unbounded.

We now show that it is possible to choose an infinite set $S_0 \subseteq \mathbf{N}$, a constant c_0, an unbounded function g_0 and a partition (U, V) of the arcs of T_0', with U nonempty, such that for all $n \in S_0$, $etype_1(T_0\alpha_n^0)$ has all arcs in U of length greater than $g_0(n)$ and all arcs in V of length less than c_0. This implies that $T_0 \to T_1$, where T_1 is the simple type obtained by deleting the arcs in U from T_0'. To obtain the partition (U, V), first choose an arc of T_0' that has unbounded weight in $etype_1(T_0\alpha_n^0)$. Let $S \subseteq \mathbf{N}$ be an infinite set and $g : \mathbf{N} \to \mathbf{N}$ a strictly increasing function such that for all $n \in S$ this arc has weight greater than $g(n)$ in $etype_1(T_0\alpha_n^0)$. We put this arc in U, guaranteeing this set is non-empty. Consider another arc of T_0'. Such an arc either has weight in $etype_1(T_0\alpha_n^0)$ bounded by a constant for all $n \in S$, or is infinitely often greater than some strictly increasing function. In case of the latter, we add the arc to U and refine the set S and the function g so that $g(n)$ provides a lower bound for the weight of both arcs for $n \in S$. In case of the former, we add the arc to V. Repeating this process for all the arcs of T_0', we obtain the required partition, set, function and constant.

If T_1 is directly solvable we are finished. Otherwise, let (σ_n^1) be an optimal solution of T_1. Let T_0'' be the enumerated one-sided type obtained by letting all arcs in T_1 have weight c_0. That is, all arcs in T_0'' have weight at least as great as the weight of the corresponding arc in $etype_1(T_0\alpha_n^0)$. Thus, $ord(\Gamma_1, T_0''\beta_n^0, \Pi_2) \geq ord(\Gamma_1, T_0\alpha_n^0\beta_n^0, \Pi_2)$ for $n \in S_0$. Consequently, by Lemma 12 and the optimality of σ_n^1, we obtain that $|\sigma_n^1| \leq |\beta_n^0| + k_0$ for some constant k_0. Since $|\sigma_n^0| = f_0(n) + |\beta_n^0|$, we have

$$|\sigma_n^1| + f_0(n) \leq |\sigma_n^0| + k_0$$

for $n \in S_0$. Applying the argument above to the sequence (σ_n^1) on the set S_0, we obtain a simple type T_2 such that $T_1 \to T_2$, and an infinite subset S_1 of S_0 such that, for $n \in S_1$,

$$|\sigma_n^2| + f_1(n) \leq |\sigma_n^1| + k_1$$

for some growing function f_1 and constant k_1. Continuing this argument, we obtain a sequence of inequalities, holding on an infinite set, of the form

$$|\sigma_n^{i+1}| + f_i(n) \leq |\sigma_n^i| + k_i$$

It follows that we cannot continue this process indefinitely. For, otherwise, some simple type would repeat and we would obtain $\sigma_n^i = \sigma_n^{j+1}$ for some $i \leq j$. But then we deduce from the inequalities above an inequality of the form $f_i(n) + f_{i+1}(n) + \ldots + f_j(n) \leq k_i + \ldots + k_j$, which contradicts the fact that the f_i are unbounded. Thus we terminate with some T_k which is directly solvable. □

Lemma 14 *If T_0 and T_1 are simple one-sided types and $T_0 \to T_1$ there exists a witness to this fact of the form $\sigma_n = \alpha\beta^n\gamma$, for fixed strings α, β, γ.*

Proof: Let (σ_n) be a witness to $T_0 \to T_1$, where $type_1(T_0\sigma_n) = T_3$ for all n and (U, V) is the partition of the arcs of T_3. If the strings σ_n are bounded in length we are done. (For, some σ_i recurs infinitely often, and we may take β to be the empty string and write $\sigma_i = \alpha\gamma$ in any way to obtain a witness to $T_0 \to T_1$.)

Otherwise, write $\sigma_n = \delta_n \epsilon_n$ where ϵ_n is the maximal suffix of σ_n such that no arc in U is contained in $type_1(\epsilon_n)$. Since the arcs in U are unbounded in weight in the enumerated types $etype_1(T_0\sigma_n)$, the length of the ϵ_n is unbounded. Since the number of two-sided types is finite, for sufficiently large n we may find a suffix $\beta_1\gamma$ of ϵ_n, where β_1 is nonempty, such that $type_2(\gamma) = type_2(\beta_1\gamma)$. Since the arcs in V are bounded in length, we may futhermore assume that all these arcs lie within γ. Thus, for some n we may write $\sigma_n = \alpha\beta_1\gamma$, where no arc in U is in $type_1(\beta_1\gamma)$ (but every arc of V is) and $type_2(\gamma) = type_2(\beta_1\gamma)$. Since no arc of U is in $type_2(\beta_1\gamma)$, every path in $\alpha\beta_1\gamma$ witnessing such an arc contains a subpath witnessing a spanning arc of $\beta_1\gamma$. It follows that $\sigma_n' = \alpha(\beta_1^d)^n\gamma$ is a witness to $T_0 \to T_1$. $\Box$

Lemma 7 now follows from Lemma 13 and Lemma 14. Let $\sigma_n^i = \alpha_i(\beta_i)^n\gamma_i$ be a witness to $T_i \to T_{i+1}$ for $i = 0 \ldots k-1$, where $T_0 = I$, and let $\sigma_n^k = \sigma$ be a direct solution to T_k. Let $\tau_n = \sigma_n^0 \ldots \sigma_n^k$. Then the sequence (τ_n) is a solution to I. For, consider a path witnessing the realization in τ_n of an expansion by Π_2 of some $R \in \Gamma_1$ and let i be the least number such that this path is realized entirely in $T_0\sigma_n^0 \ldots \sigma_n^i$. Since $\Pi_2^\infty(T_i\sigma_n^i)$ omits Γ_1, the path must contain a subpath witnessing some "deleted" arc of $type_1(T_0\sigma_n^0 \ldots \sigma_n^{i-1})$. But the latter type of path must have length greater than n, so the original path has length greater than n also. This completes the proof of Lemma 7.

To determine if $I * \Pi_1$ has Γ_1 unbounded with respect to Π_2, it therefore suffices to look for strings of the sort promised by Lemma 7. This may be done by the nondeterministic algorithm shown in Figure 2, which operates in two phases. In phase 1, we look for the strings α_i promised by Lemma 7. In phase 2, we look for the strings β_i of Lemma 7.

The function *pumpresult* used at line 13 is depicted in Figure 3. The function *rhs* it calls projects a two-sided type $(\sigma_l, \sigma_r, Head, Tail, Chain)$ to the one-sided type $(\sigma_r, Head', Tail', Chain')$, where $Head'$, and $Tail'$, are the sets of arcs (Q, X) in $Head$ (respectively, $Tail$) with X a variable of σ_r, and $Chain'$ is the set of arcs (Q, X, Q', Y) in $Chain$ with both X and Y variables of σ_r.

1. $T_1 := \mathit{type}_1(I)$

PHASE 1:

2. Nondeterministically choose to go to PHASE 2 or continue
3. Guess a rule a of Π_1
4. If $\Pi_2^\infty(T_1 a)$ realizes Γ_1 then reject
5. If a is an initialization rule then accept
6. $T_1 := \mathit{type}_1(T_1 a)$
7. Go to 2

PHASE 2:

8. $T_2 :=$ null two sided type
9. Guess a rule a of Π_1
10. If $\Pi_2^\infty(T_1 T_2 a)$ realizes Γ_1 then reject
11. $T_2 := \mathit{type}_2(T_2 a)$
12. If $\mathit{type}_1(T_1 T_2) \neq T_1$ then go to 9
13. $T_1 := \mathit{pumpresult}(T_2)$
14. go to PHASE 1

Figure 2: A PSPACE Algorithm

pumpresult(T)
repeat
$T_0 := T$
$T := \mathit{type}_2(T T_0)$
until $rhs(T) = rhs(T_0)$
return $rhs(T)$

Figure 3: Pumping a two-sided type

The function *pumpresult*(T) operates by computing $\mathit{type}_2(T^n)$ for increasing values of n until the set of arcs on the right hand side of this type stabilise for some $n = N$. If β is a string with $\mathit{etype}_2(\beta) = T$, then the arcs on the right hand side of β^N all occur in the types $\mathit{etype}_1(T_1\beta^n)$ (for $n \geq N$) with bounded weight. All other arcs of $\mathit{etype}_1(T_1\beta^n)$ require a spanning arc of β^n, hence have unbounded weight. Thus the function call *pumpresult*(T_1) computes the one-sided type T_3 such that $(\gamma^d)^n$ is a witness for $T_1 \rightarrow T_3$, where γ is the sequence of choices made during phase 2. (We use γ^d rather than γ here to ensure that any spanning arc of $\mathit{etype}_2(\gamma^d)^n$ has weight n. Strictly, there may be unbounded arcs of $\mathit{etype}_1(T_1\gamma^n)$ that have weight less than n, whereas the definition of $T_1 \rightarrow T_3$ requires the unbounded arcs to have weight at least n.)

It is not difficult to check that the algorithm for boundedness runs in nondeterministic polynomial space, which is equivalent to polynomial space [9]. As we remarked above, the tranformation to eliminate recursive calls to the head variable and ensure that there is a single IDB predicate may result in an exponential blowup of the size of the program Π_1. However, we can avoid paying this cost and maintain the PSPACE bound by constructing the rules a of the transformed program on the fly in steps 3 and 9 of the algorithm.

This completes the proof of Theorem 1, modulo a simplifying assumption. We have assumed so far the emptiness of the set of queries Γ_2 that must be omitted with respect to Π_3 in condition (*). To eliminate this assumption, it is necessary to extend the notion of type to also include arcs deriving from Π_3-paths. The solution (σ_n) constructed must have the property that in each string in $I\sigma_n$ realizes no expansion by Π_3 of a query in Γ_2. This can be easily checked during the type computations done by the algorithm by using head, tail and chain arcs. Note that in pumping, we should not delete arcs associated with Π_3, only those of Π_2. It is necessary to first rename the IDB predicates in Π_3 and Γ_2 to be disjoint from those in Π_2 in order to make this distinction possible.

4 Conclusion

Several gaps in the bounds of [3] remain. For example, predicate boundedness of nonlinear monadic Datalog is known to be in 3-EXPTIME and 1-EXPTIME hard. For program boundedness the known upper and lower bounds are 2-EXPTIME and 1-EXPTIME, respectively. We do not know if our techniques can be generalized to apply to the remaining open problems; such generalization does not look to be straightforward.

Acknowledgement: Thanks to Ann Yasuhara and Eric Allender for helpful discussion.

References

[1] V.H. Aho and J.D. Ullman. Universality of data retrieval languages. In *Proc. 6th ACM Symp. on Principles of Programming Languages*, pages 214–226, 1979.

[2] François Bancilhon and Raghu Ramakrishnan. An amateur's introduction to recursive query processing strategies. In Carlo Zaniolo, editor, *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 16–52, Washington, D.C., 28–30 May 1986. *SIGMOD Record* 15(2), June 1986.

[3] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 477–490, Chicago, Illinois, 2–4 May 1988.

[4] Haim Gaifman, Harry Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *Journal of the ACM*, 40(3):683–713, July 1993.

[5] Gerd G. Hillebrand, Paris C. Kanellakis, Harry G. Mairson, and Moshe Y. Vardi. Tools for Datalog boundedness. In *Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–12, Denver, Colorado, 29–31 May 1991.

[6] J.B. Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory (Ser. A)*, 13:297–305, 1972.

[7] R. van der Meyden. Recursively indefinite databases. *Theoretical Computer Science*, 116(1,2):151–194, 1993.

[8] Jeffrey F. Naughton. Data independent recursion in deductive databases. *Journal of Computer and System Sciences*, 38(2):259–289, April 1989.

[9] W. Savitch. Relationships between non-deterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[10] J.D. Ullman. *Principles of Database and Knowledge Base Systems, volume I.* Computer Science Press, 1988.

[11] J.D. Ullman. *Principles of Database and Knowledge Base Systems, volume II: The New Technologies.* Computer Science Press, 1989.