On Datalog vs. LFP

Anuj Dawar and Stephan Kreutzer

[1] University of Cambridge Computer Lab
anuj.dawar@cl.cam.ac.uk
[2] Oxford University Computing Laboratory
kreutzer@comlab.ox.ac.uk

Abstract. We show that the homomorphism preservation theorem fails for LFP, both in general and in restriction to finite structures. That is, there is a formula of LFP that is preserved under homomorphisms (in the finite) but is not equivalent (in the finite) to a Datalog program. This resolves a question posed by Atserias. The results are established by two different methods: (1) a method of diagonalisation that works only in the presence of infinite structures, but establishes a stronger result showing a hierarchy of homomorphism-preserved problems in LFP; and (2) a method based on a pumping lemma for Datalog due to Afrati, Cosmadakis and Yannakakis which establishes the result in restriction to finite structures. We refine the pumping lemma of Afrati et al. and relate it to the power of Monadic Second-Order Logic on tree decompositions of structures.

1 Introduction

Among the important classical results of model theory, relating syntactic to semantic properties of first-order logic, are the preservation theorems. For instance, the Łoś-Tarski theorem tells us that a sentence of first-order logic is equivalent to an *existential sentence* if, and only if, the class of its models is closed under extensions and Lyndon's theorem states that a sentence is monotone in a relation R if, and only if, it is equivalent to one that is positive in R (see [12]). The study of preservation theorems has played an important role in the development of finite model theory, with many early results demonstrating that such results fail when we restrict consideration to finite structures (see, for instance, [8]).

One important exception to the general failure of preservation theorems in the finite is Rossman's proof of the homomorphism preservation theorem [17]. This shows that on the class of finite structures, just as on the class of all structures (finite or infinite) a sentence of first-order logic is equivalent to an *existential positive* sentence if, and only if, it is preserved under homomorphisms. The homomorphism preservation property in finite structures has aroused much interest in theoretical computer science through its connections with questions in database theory and the study of constraint satisfaction problems (CSPs).

Each of the preservation theorems mentioned has two directions, one of which is generally quite easy to establish: namely that the syntactic restriction (such as the restriction to existential positive sentences) implies the semantic restriction (being preserved under homomorphisms). Moreover, this direction holds generally on any class of structures $\mathcal{C}$.

L. Aceto et al. (Eds.): ICALP 2008, Part II, LNCS 5126, pp. 160–171, 2008.

The other direction, sometimes known as *expressive completeness*, states that any sentence that satisfies the semantic restriction is equivalent to one of the simple syntactic form. When we restrict this statement to a class $\mathcal{C}$, we weaken both the hypothesis and the conclusion of the statement. Thus, even for classes $\mathcal{C}$ and $\mathcal{C}'$ where $\mathcal{C} \subseteq \mathcal{C}'$, it is impossible to deduce either the validity or the failure of a preservation theorem on $\mathcal{C}$ from the statement for $\mathcal{C}'$. In particular, the statements for the class of all structures and for the class of finite structures alone are quite independent statements. Recently, there has been a growing interest in investigating the status of preservation theorems for classes $\mathcal{C}$ more restrictive than the class of all finite structures [5,6].

Atserias [3] (see [1, Question 4.3]) asked whether the homomorphism preservation theorem holds for LFP—the extension of first-order logic with an operator for defining least fixed points of monotone formulas. Fixed-point logics have arguably played a more important role in finite model theory than first-order logic. In particular, it is known that LFP expresses all polynomial time computable properties of finite ordered structures [13,18]. Thus, the question of whether a homomorphism preservation theorem can be established for this logic arises naturally. The language formed by extending existential positive formulas by means of a least fixed-point operator is Datalog and it has been extensively studied as a database query language. It has also received attention in the study of constraint satisfaction problems as it provides a general means of classifying many CSPs as tractable. It is easily seen that any query defined in Datalog is preserved under homomorphisms. Thus, Atserias' question asks whether it is the case that every sentence of LFP that is preserved under homomorphisms is equivalent to a Datalog program. We show in this paper that this is not the case, either on the class of all (finite or infinite) structures or in restriction to the class of finite structures.

The homomorphism preservation question for extensions of first-order logic was also studied by Feder and Vardi [10]. They showed that on finite structures, the homomorphism preservation property holds for a number of *existential* infinitary and fixed-point logics. In particular, they established that any query definable in Datalog$(\neg, \neq)$ that is closed under homomorphisms is already definable in Datalog. The former language is the extension of Datalog with inequality and negation on EDB predicates. Just as Datalog can be seen as the existential positive fragment of LFP, Datalog$(\neg, \neq)$ is its existential fragment. Thus, our results show that the theorem of Feder and Vardi cannot be extended from Datalog$(\neg, \neq)$ to LFP.

The two examples we construct separating LFP from Datalog bear some similarity to each other in that they are defined in terms of graphs having path lengths in some set S. In addition, to guarantee that the classes we consider are closed under homomorphisms, we take the union with the class of all graphs containing a cycle. The main differences in the two results are in the choice of the set S and in the method used to prove that the resulting class of graphs is not definable in Datalog. In the case where we allow infinite structures, the proof is somewhat simpler as we can construct a set S that is undefinable in Datalog (over the natural numbers) using standard diagonalisation arguments and then obtain the result by means of a reduction of the graph problem to this set. This actually establishes something stronger. It shows that for every k, there are formulas of LFP that are preserved under homomorphisms but not definable by a formula with

only k nested alternations of the fixed-point operator with negation. These results are established in Section 4.

When we restrict ourselves to finite structures, such diagonalisation methods are unavailable and we adapt a pumping lemma due to Afrati et al. [2] for our purpose. Afrati et al. use their pumping lemma to demonstrate polynomial-time monotone properties that are not definable in Datalog. In order to adapt it to the LFP-definable properties we are interested in, we need to show that it works on a class of acyclic graphs. What we establish is that if π is a Datalog program which accepts a directed acyclic graph $(\mathbf{G}, s, t)$ if, and only if, $\mathbf{G}$ contains a path from s to t of length p for some p in a given set S, then S cannot grow too fast (the precise statement is given in Lemma 5.2). This suffices to establish the result we seek. An apparently stronger pumping lemma (saying that S cannot grow faster than linearly) is stated in [2], but without the restriction to acyclic graphs. In the absence of this restriction, we cannot use their lemma directly and it is not clear from their description of the proof that it can be adapted. This is explained in more detail in Section 5. One virtue of our proof of this pumping lemma is that it connects it with other recent innovations in the analysis of Datalog queries, namely their relationship with tree decompositions and with the power of monadic second-order logic over these. This new insight into Datalog may be of independent interest.

One source of interest in the relationship between LFP and Datalog is research on the classification of tractable constraint satisfaction problems. We can associate with any structure $\mathbf{B}$, the decision problem CSP($\mathbf{B}$) of determining for a given structure $\mathbf{A}$ whether there is a homomorphism $\mathbf{A} \to \mathbf{B}$. This is the *constraint satisfaction problem* associated with $\mathbf{B}$ (see [9]). Much research work has been devoted to classifying those structures $\mathbf{B}$ for which this problem is decidable in polynomial time. It is immediate from the above definition that the complement of CSP($\mathbf{B}$) is closed under homomorphisms. If we could find a *finite* structure $\mathbf{B}$ for which the complement of CSP($\mathbf{B}$) is definable in LFP but not in Datalog, this would resolve certain conjectures on the classification of tractable CSPs (see [4] for a discussion). We note here that our example of a homomorphism closed class separating LFP from Datalog on finite structures is not the complement of CSP($\mathbf{B}$) for any finite $\mathbf{B}$, but is of this form for an *infinite* structure $\mathbf{B}$.

We begin in Section 2 with definitions, including those of LFP and Datalog as well as first-order and monadic second-order logic. We also recall the definitions of tree decompositions of structures and relate them to Datalog programs.

2 Preliminaries

We briefly introduce the fundamental concepts and notation we need in later sections.

Homomorphisms and Preservation. Let σ be a finite signature. We use boldface letters for structures $\mathbf{A}, \mathbf{B}, ...$ and corresponding Roman letters $A, B, ...$ to denote their universe. We also write $\boldsymbol{a}$ for a tuple $a_1, \ldots, a_k$.

Definition 2.1. Let σ be a relational signature possibly with constant symbols and let $\mathbf{A}, \mathbf{B}$ be σ-structures. A *homomorphism* from $\mathbf{A}$ to $\mathbf{B}$ is a function $h : A \to B$ such that for every k-ary relation symbol $R \in \sigma$ and every k-tuple $\boldsymbol{a} \in A^k$ if $\boldsymbol{a} \in R^{\mathbf{A}}$ then $(h(a_1), \ldots, h(a_k)) \in R^{\mathbf{B}}$ and for every constant symbol $c \in \sigma$, $h(c^{\mathbf{A}}) = c^{\mathbf{B}}$. We write $\mathbf{A} \to \mathbf{B}$ to denote that there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.

Definition 2.2. Let $\mathcal{C}$ be a class of structures. A subclass $\mathcal{D} \subseteq \mathcal{C}$ is *closed under homomorphisms* if whenever $\mathbf{A}, \mathbf{B} \in \mathcal{C}$ so that $\mathbf{A} \in \mathcal{D}$ and there is a homomorphism $h : \mathbf{A} \to \mathbf{B}$ then $\mathbf{B} \in \mathcal{D}$. We are particularly interested in model classes of sentences. If φ is a sentence of a logic, we say *φ is preserved under homomorphisms on $\mathcal{C}$* if the class $Mod_{\mathcal{C}}(\varphi) := \{\mathbf{A} \in \mathcal{C} : \mathbf{A} \models \varphi\}$ is closed under homomorphisms.

First-Order Logic, Monadic Second-Order Logic and Types. Let σ be a signature. We assume that the reader is familiar with first-order logic. We write FO(σ) for the class of all first-order formulas over the signature σ. *Monadic Second-Order Logic* (MSO) is the extension of first-order logic by quantification over sets of elements, i.e. there are quantifiers $\exists X, \forall X$, where X is a unary relation variable, and a formula $\exists X \varphi$ is true in a structure $\mathbf{A}$, written $\mathbf{A} \models \exists X \varphi$, if there is a set $X \subseteq A$ such that $(\mathbf{A}, X) \models \varphi$. The semantics of $\forall X \varphi$ is defined analogously. See e.g. [8] for more on MSO.

The *quantifier rank* qr(φ) of a formula φ (of FO or MSO) is the maximal depth of nesting of quantifiers in φ. Note that up to logical equivalence there are only finitely many MSO-formulas of quantifier rank at most q in a finite signature σ. We write MSO_q for the class of MSO-formulas of quantifier rank at most q. We write $\mathbf{A} \equiv^q \mathbf{B}$ to denote that two structures $\mathbf{A}$ and $\mathbf{B}$ cannot be distinguished in MSO_q.

A *type* is a maximally consistent class of formulas. For a structure $\mathbf{A}$ and $q \in \mathbb{N}$, the MSO_q-type of $\mathbf{A}$ is the class of MSO-sentences of quantifier rank at most q which are true in $\mathbf{A}$ and if $\boldsymbol{a} \in A^k$, then the MSO_q-type of $\boldsymbol{a}$ in $\mathbf{A}$ is the class of all MSO_q-formulas $\varphi(\boldsymbol{x})$ such that $\mathbf{A} \models \varphi(\boldsymbol{a})$. As, for each $q \in \mathbb{N}$, MSO_q only contains finitely many formulas up to equivalence, the MSO_q-type of a tuple or a structure can completely be described by a single formula in MSO_q (see [8]). We will use the following decomposition theorem for MSO. See e.g. [15] or [11].

Lemma 2.3. *Let $\mathbf{A}$ and $\mathbf{B}$ be structures and let $\boldsymbol{u}$ be a tuple listing the vertices in the intersection of A and B. The MSO_q-type of $\boldsymbol{u}$ in $\mathbf{A} \cup \mathbf{B}$ is uniquely determined by the MSO_q-types of $\boldsymbol{u}$ in $\mathbf{A}$ and in $\mathbf{B}$.*

In particular, if $\mathbf{A}$, $\mathbf{B}_1$ and $\mathbf{B}_2$ are structures such that $A \cap B_1 = A \cap B_2 =: \boldsymbol{u}$ and the MSO_q-types of $\boldsymbol{u}$ in $\mathbf{B}_1$ and $\mathbf{B}_2$ are the same, then $\mathbf{A} \cup \mathbf{B}_1 \equiv^q \mathbf{A} \cup \mathbf{B}_2$.

Least Fixed-Point Logic. We first present a brief introduction to least fixed-point logic. For a detailed exposition see [8]. Let σ be a signature and let $\varphi(R, \boldsymbol{x})$ be a formula of signature σ which is *positive* in the k-ary relation variable R, i.e. every atom of the form $R\boldsymbol{t}$ in φ occurs within the scope of an *even* number of negation symbols. For every σ-structure $\mathbf{A}$, φ defines a monotone operator[1] $F_{\mathbf{A},\varphi} : \text{Pow}(A^k) \to \text{Pow}(A^k)$ via $F_{\mathbf{A},\varphi}(P) := \{\boldsymbol{a} \in A^k : (\mathbf{A}, P) \models \varphi[\boldsymbol{a}]\}$, for every $P \subseteq A^k$. A theorem due to Knaster and Tarski shows that on every structure $\mathbf{A}$ every monotone operator $F_{\mathbf{A},\varphi}$ has a least fixed point which we denote by $\mathbf{lfp}(F_{\mathbf{A},\varphi})$.

The logic LFP(σ) is the extension of FO(σ) by least fixed-point operators. To be precise: LFP(σ) contains FO(σ) and is closed under Boolean connectives and first-order quantification; and if $\varphi(R, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{Q})$ is an LFP(σ)-formula which is positive in the k-ary relation variable R then for every k-tuple $\boldsymbol{t}$ of terms $[\mathbf{lfp}_{R,\boldsymbol{x}}\ \varphi](\boldsymbol{t})$ is an LFP(σ)-formula such that for every $\big(\sigma \dot\cup \{\boldsymbol{z}, \boldsymbol{Q}\}\big)$-structure $\mathbf{A}$ and every tuple $\boldsymbol{a} \in A^k$ we have $\mathbf{A} \models [\mathbf{lfp}_{R,\boldsymbol{x}}\ \varphi](\boldsymbol{a})$ if, and only if, $\boldsymbol{a} \in \mathbf{lfp}(F_{\mathbf{A},\varphi})$.

[1] An operator $F : \text{Pow}(M) \to \text{Pow}(M)$ is *monotone* iff $F(A) \subseteq F(B)$ for all $A \subseteq B \subseteq M$.

The *alternation depth* of an LFP formula φ is defined as the maximal number of alternations between fixed-point operators and negations inside φ. We write LFP^k for the class of LFP formulas of alternation depth at most k.

Datalog. Datalog is a database query language which could be defined as the collection of formulas of LFP which do not use negation or universal quantification. However, the usual presentation of the language is in terms of function-free Horn clauses, and we follow this presentation below as the structure of the program in terms of rules is useful for our proof of the pumping lemma in Section 5.

A *Datalog program* is a finite set of rules of the form $T_0 \leftarrow T_1, \ldots, T_m$, where each T_i is an atomic formula. T_0 is called the *head* of the rule, while the right-hand side is called the *body*. The relation symbols that occur in the heads are the *intensional* database predicates (IDBs), while all others are the *extensional* database predicates (EDBs). Note that IDBs may occur in the bodies too, thus, a Datalog program is a recursive specification of the IDBs with semantics obtained via least fixed-points of monotone operators. The collection of EDB predicates occurring in π constitute its *signature* σ, and a Datalog program of signature σ is interpreted in σ-structures. One IDB predicate is distinguished as the *goal predicate*. In general, we will assume that the goal predicate is a 0-ary predicate, so that the program defines a Boolean query. In the interests of space, we will not give a formal definition of the semantics of the program, which can be found in standard textbooks such as [8]. A key parameter in analysing Datalog programs is the number of variables used. We write k-Datalog for the collection of all Datalog programs with at most k distinct variables in total.

A formula of first-order logic is said to be a *conjunctive query* if it is obtained from atomic formulas using only conjunctions and existential quantification. Every finite structure $\mathbf{A}$ with n elements gives rise to a *canonical conjunctive query* $\varphi_{\mathbf{A}}$, which is obtained by first associating a different variable x_i with every element a_i of $\mathbf{A}$, $1 \leq i \leq n$, then forming the conjunction of all atomic facts true in $\mathbf{A}$, and finally existentially quantifying all variables x_i, $1 \leq i \leq n$. In other words, the formula $\varphi_{\mathbf{A}}$ is the existential closure of the *positive diagram* of $\mathbf{A}$ (see [12]). The significance of these queries lies in the fact (first noted by Chandra and Merlin [7]) that for any structure $\mathbf{B}$, $\mathbf{B} \models \varphi_{\mathbf{A}}$ if, and only if, there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$.

For every positive integer k, let CQ^k be the collection of conjunctive queries that have at most k distinct variables. Note that each variable may be reused, so its number of occurrences may be arbitrarily large. The significance of CQ^k lies in that the number of variables required to express $\varphi_{\mathbf{A}}$ is closely related to the *tree width* of $\mathbf{A}$. We first review the definition of tree width and then state its relationship with CQ^k.

Let $\mathbf{A}$ be a σ-structure. A *tree-decomposition* of $\mathbf{A}$ is a pair $(\mathbf{T}, B)$ where $\mathbf{T}$ is a directed tree oriented from the root to the leaves and B is a labelling that associates to each node t of $\mathbf{T}$ a non-empty set of elements $B_t \subseteq A$ such that

1. for every tuple $\boldsymbol{a}$ in some relation R of $\mathbf{A}$, there is a node $t \in T$ such that $\boldsymbol{a}$ is contained in B_t; and
2. for every $a \in A$, the set $\{t \in T : a \in B_t\}$ forms a connected subtree of $\mathbf{T}$.

The *width* of a tree-decomposition is the maximum cardinality of a set B_t minus one. The *treewidth* of $\mathbf{A}$ is the smallest k for which $\mathbf{A}$ has a tree-decomposition of width k.

The connection between the number of variables in $\varphi_{\mathbf{A}}$ and the tree width of $\mathbf{A}$ can now be summarised as follows (see [14,6]).

Lemma 2.4. *If* $\mathbf{A}$ *has tree width less than* k*, then* $\varphi_{\mathbf{A}}$ *is equivalent to a formula of* CQ^k*. For any satisfiable formula* φ *in* CQ^k*, there is a structure* $\mathbf{A}$ *with tree width less than* k*, such that* $\varphi_{\mathbf{A}}$ *is logically equivalent to* φ*.*

A Datalog program π can be *unfolded* into a conjunctive query, by repeatedly expanding the rules. There are infinitely many such unfoldings for a recursive program. We are interested in the structures, called *expansions* of π, for which these unfoldings are the canonical conjunctive queries.

Definition 2.5. Given a Datalog program π, a *partial unfolding* of π is any conjunctive query obtained using the following rules:

- The goal predicate G of π is a partial unfolding of π;
- If ϑ is a partial unfolding of π; R is an IDB predicate of π; $R(\boldsymbol{x})$ is an atomic formula occurring in ϑ; and $R(\boldsymbol{y}) \leftarrow T_1(\boldsymbol{z_1}), \ldots, T_m(\boldsymbol{z_m})$ is a rule of π, let $\varphi(\boldsymbol{x})$ be the formula obtained from $\exists \boldsymbol{z}(T_1(\boldsymbol{z_1}) \wedge \cdots \wedge T_m(\boldsymbol{z_m}))$ (where $\boldsymbol{z}$ includes all variables occurring in the rule except for those in $\boldsymbol{y}$) by replacing the variables in $\boldsymbol{y}$ by $\boldsymbol{x}$. Then, the formula ϑ' obtained from ϑ by replacing the occurrence $R(\boldsymbol{x})$ by $\varphi(\boldsymbol{x})$ is also a partial unfolding of π.

An *unfolding* of π is a partial unfolding in which no IDB predicate occurs.

It is not difficult to see that any unfolding of a Datalog program is a conjunctive query, and more particularly, if π is a k-Datalog program, then any unfolding of π is in CQ^k. It is also easily established that a structure $\mathbf{A}$ is in the query defined by π if, and only if, there is some unfolding ϑ of π such that $\mathbf{A} \models \vartheta$.

Definition 2.6. An *expansion* of a k-Datalog program π is a structure $\mathbf{A}$ of tree width less than k such that the canonical conjunctive query $\varphi_{\mathbf{A}}$ is logically equivalent to an unfolding of π.

Now, it is clear, by Lemma 2.4, that $\mathbf{B} \models \pi$ if, and only if, $\mathbf{A} \rightarrow \mathbf{B}$ for some expansion $\mathbf{A}$ of π. Indeed, the models of π are generated from expansions whose tree decompositions are given by the unfolding of π.

Definition 2.7. A *decorated expansion* of the k-Datalog program π is a tree decomposition $(\mathbf{T}, B)$ of an expansion $\mathbf{A}$ of π along with a labelling L that associates to each node t of $\mathbf{T}$ a pair (r, ρ) where r is either a rule of π or an atomic formula $R(\boldsymbol{x})$ (for an EDB predicate R); and ρ is an injective mapping from the variables of r to B_t.

The labelling L must satisfy the following conditions:

1. If $L(t) = (r, \rho)$ and r is an atomic formula, then t is a leaf of $\mathbf{T}$.
2. If $L(t) = (r, \rho)$ and r is a rule $R(\boldsymbol{x}) \leftarrow T_1(\boldsymbol{z_1}), \ldots, T_m(\boldsymbol{z_m})$, then t has exactly m children $t_1, \ldots, t_m$ where for each i, if $L(t_i) = (r_i, \rho_i)$ then r_i is either an atomic formula $T_i(\boldsymbol{y})$ or a rule whose head is $T_i(\boldsymbol{y})$. Further $\rho_i(\boldsymbol{y}) = \rho(\boldsymbol{z_i})$.

3 LFP Definable Classes Closed Under Homomorphisms

In this section we introduce the classes of structures which we will use to separate LFP from Datalog, and show that they are LFP definable, though proofs are omitted for lack of space.

A *source-target graph* is a (finite or infinite) directed graph $\mathbf{G}$ with two distinguished vertices s and t, i.e. a structure over the signature $\{E, s, t\}$ where E is a binary relation symbol and s, t are constant symbols. For a source-target graph $\mathbf{A} = (\mathbf{G}, s, t)$, let $n^{\mathbf{A}}$ denote $\sup\{p : \mathbf{G}$ contains a simple path of length p starting at $s\}$. Note that $n^{\mathbf{A}}$ is either a finite ordinal or ω. In the sequel, when we speak about a graph, we mean a source-target graph.

Fix a set $S \subseteq \omega$ of natural numbers. We define the following classes of graphs.

- Cyc – the class of graphs that contain a cycle.
- Unb – the class of graphs $\mathbf{A}$ for which $n^{\mathbf{A}} = \omega$.
- P_S – the class of graphs $\mathbf{A}$ that contain a path from s to t of length p for some $p \in S$.
- $C_S := P_S \cup$ Cyc.
- $C_S^\infty = (P_S \cap \text{Unb}) \cup \text{Cyc}$.

It is the classes C_S and C_S^∞ (for suitable choices of the set S) which we show separate LFP from Datalog. Note that all acyclic graphs in C_S^∞ are infinite, while C_S may contain finite as well as infinite acyclic graphs. We begin first by noting that these classes are closed under homomorphisms.

Lemma 3.1. *The classes C_S and C_S^∞ are closed under homomorphisms.*

It can be shown that even the classes P_S are closed under homomorphisms. The reason we work with the classes C_S and C_S^∞ is for the sake of definability in LFP. It is difficult to use LFP to determine the lengths of paths in the presence of cycles. In fact, the longest path problem is NP-complete and hence unlikely to be definable in LFP. By including all graphs with cycles, we make the problem easier, as then we only have to consider the longest path in acyclic digraphs. We now aim to show that if the set S is definable in LFP, in some sense, then the classes C_S and C_S^∞ are also definable.

For an ordinal $\alpha \in [0, \omega]$, we write (α, succ) to denote the structure whose universe is $\{\beta : \beta < \alpha\}$ and where *succ* is interpreted as the binary successor relation.

Lemma 3.2. *There is a uniform* LFP *interpretation of $(n^{\mathbf{A}}, \mathit{succ})$ in acyclic source-target graphs $\mathbf{A}$.*

The proof of Lemma 3.2 relies on the use of *stage comparison relations*, see [16]. We remark that the interpretation in Lemma 3.2 is already definable in LFP^1, the alternation free fragment of LFP.

Lemma 3.3. *There is a formula φ_{unb} of* LFP *that defines Unb on acyclic graphs.*

This is used to show the definability of the classes C_S and C_S^∞.

Lemma 3.4. *If $S \subseteq \omega$ is definable in the structure (ω, succ) by a formula of LFP^k, then the class C_S^∞ is defined by a sentence of LFP^{k+1}.*

Note that the class of sets S that are definable by LFP formulas in (ω, succ) is very rich. In particular, it includes all Π_1^1-definable sets of numbers.

Lemma 3.5. *If the class of finite structures $\mathcal{S} = \{(n, \mathit{succ}) : n \in S\}$ is definable in* LFP*, then C_S is defined by a sentence of* LFP.

Note that $\{(n, \mathit{succ}) : n \in S\}$ is definable in LFP if, and only if, the set S, represented in unary, is decidable in polynomial time.

4 The Diagonalisation Method

The main result of this section is the following theorem.

Theorem 4.1. *There is a sentence of* LFP *that is preserved under homomorphisms on the class of all structures but which is not equivalent to any* Datalog *program.*

Since Datalog is in the negation-free fragment of LFP, it is clear that every Datalog program is equivalent to a formula in LFP^1. Using diagonalisation methods, one can show that for each k there is a subset $S_k \subset \omega$ such that S_k can be defined in the structure $(\omega, succ)$ by an LFP^{k+1} formula $\varphi_k(x)$ but not by any formula in LFP^k, where *succ* denotes the successor relation on ω. See e.g. [16]. Thus, we can choose a set S of natural numbers which is definable in LFP on the structure $(\omega, succ)$ but not in Datalog. Our aim is to show that the class C_S^∞ is not definable in Datalog.

Lemma 4.2. *For any set S, if there is a* Datalog *program defining C_S^∞, then S is definable in $(\omega, succ)$ by a* Datalog *program.*

This allows us to prove Theorem 4.1, as we can choose a set S that is definable in LFP but not in LFP^1. Then, Lemma 3.4, 3.1 and 4.2 together imply the theorem. The proof actually implies a somewhat stronger result.

Corollary 4.3. *For every k, there is an LFP^{k+2}-definable class of structures which is closed under homomorphisms but which cannot be defined in LFP^k.*

5 The Pumping Method

The result in the previous section relies crucially on infinite structures. In particular, the class C_S^∞ restricted to finite structures is just the class of all graphs containing a cycle, and this is definable in Datalog. Moreover, the stronger Corollary 4.3 cannot hold on finite structures since it is known that every formula of LFP is equivalent, in the finite, to a formula of LFP^1 (see [13]). Still, in this section we establish that the homomorphism preservation property fails even when we restrict ourselves to finite structures.

Theorem 5.1. *There is an* LFP *sentence φ which is preserved under homomorphisms on the class of all finite structures such that there is no* Datalog *program equivalent to φ on finite structures.*

Specifically, we show that there are sets of numbers S, which are polynomial-time decidable when written in unary, such that there is no Datalog program whose finite models are exactly the ones in C_S. This is established by showing the following pumping lemma.

Lemma 5.2. *Let $S \subseteq \omega$ be an infinite set of numbers and π a* Datalog *program which accepts a directed acyclic graph $(\mathbf{G}, s, t)$ if, and only if, $\mathbf{G}$ contains a path from s to t of length p for some $p \in S$. Then, there is a constant c and an increasing sequence $(a_i)_{i\in\omega}$ of numbers such that:*

1. *$a_{i+1} < a_i^c$ for all i; and*
2. *$S \cap [a_i, a_{i+1}] \neq \varnothing$ for all i.*

Before we give a proof, a few remarks are in order. Recall that a Datalog program π determines a collection $\mathcal{C}$ of expansions of bounded tree width such that a structure $\mathbf{B}$ is accepted by π if, and only if, $\mathbf{A} \to \mathbf{B}$, for some $\mathbf{A} \in \mathcal{C}$. If π is as in Lemma 5.2, then it accepts a structure $(\mathbf{G}, s, t)$ where $\mathbf{G}$ is a simple path of length $p \in S$. The expansion $\mathbf{A}$ that maps to this structure must be an acyclic graph in which all paths from s to t are of length p. To prove the lemma, we proceed from a decorated expansion for $\mathbf{A}$ to "pump" a portion of the tree decomposition and obtain a sequence of expansions $\mathbf{A}_i$ which are all acyclic and such that the lengths of all paths in $\mathbf{A}_i$ from s to t are in the interval $[a_i, a_{i+1}]$ for a suitably defined sequence $(a_i)_{i\in\omega}$. This establishes the result.

It should be noted that a similar pumping lemma is stated by Afrati et al. [2], and proved by similar means. Indeed, their statement is apparently stronger in that condition (1) can be replaced by $a_{i+1} < c + a_i$, which is to say that the sequence $(a_i)_{i\in\omega}$ can be chosen to grow linearly in i rather than exponentially. However, their statement is not confined to acyclic graphs, which is an essential restriction for our result. It would suffice for our purposes if, in the proof of the pumping lemma of Afrati et al., it could be shown that when an acyclic expansion is pumped, we always obtain an acyclic expansion, but we are unable to recover this fact from their proof. To be precise, they present the proof in detail only for the case when the expansion $\mathbf{A}$ is itself a simple path. In this case, the proof below can also be used to yield a linear sequence $(a_i)_{i\in\omega}$. They then state that the general case can be handled similarly, by choosing in the decorated expansion of $\mathbf{A}$ a collection of pairs of points to pump such that each simple path crosses exactly one such pair. We are unable to determine how such a collection could be chosen and, if the points at which we pump an expansion are crossed by more than one path, it is quite possible that pumping may create shortcuts. This is the reason why, in the proof below, we have to pump each pair of points multiple times, forcing an exponential growth in the sequence $(a_i)_{i\in\omega}$. However, this is still sufficient to establish Theorem 5.1, which is our aim here. We now proceed to a proof of Lemma 5.2.

Proof of Lemma 5.2. Let π be a Datalog-program that accepts a directed acyclic graph $(\mathbf{G}, s, t)$ if, and only if, $\mathbf{G}$ contains a path from s to t of length p for some $p \in S$ and let k be the number of variables in π. Then, for any such $(\mathbf{G}, s, t)$, there is an expansion $\mathbf{A}$ of π such that $\mathbf{A} \to (\mathbf{G}, s, t)$, and there is a corresponding decorated expansion $(\mathbf{T}, B, L)$ where $(\mathbf{T}, B)$ is a tree decomposition of $\mathbf{A}$ of width $k - 1$. We can assume, without loss of generality that each B_u, $u \in T$, has exactly k elements. It will be clear how to adapt the construction to the case where this is not so. Since $\mathbf{A}$ is acyclic (otherwise there would be no homomorphism $\mathbf{A} \to \mathbf{G}$), we let $<$ be the (partial) order on vertices of $\mathbf{A}$ induced by distance from $s^{\mathbf{A}}$ (where vertices that are not reachable from s have distance ∞).

We now represent the decorated expansion $(\mathbf{T}, B, L)$ as a relational structure $\mathbf{D}$ as follows:

- the universe of $\mathbf{D}$ is $D := T \dot\cup A$, the disjoint union of T and A;
- the constants s and t are interpreted in $\mathbf{D}$ by $s^{\mathbf{A}}$ and $t^{\mathbf{A}}$;
- $\mathbf{D}$ has a $k + 1$-ary relation B such that for each $u \in T$ there is exactly one tuple $(u, a_1, \ldots, a_k) \in B$, and it satisfies: $B_u = \{a_1, \ldots, a_k\}$ and $a_1 \leq a_2 \cdots \leq a_k$; and

– for every rule r of π and every mapping ρ from the variables of π to $\{1, \ldots, k\}$, there is a unary relation $L_{r,\rho}$ interpreted in $\mathbf{D}$ by $\{u \in T : L(u) = (r, \rho')\}$, where ρ' is the map that takes x to $a_{\rho(x)}$ where $(u, a_1, \ldots, a_k) \in B$.

We will not distinguish notationally between $(\mathbf{T}, B, L)$ and $\mathbf{D}$ in the sequel, as it will always be clear from the context in which presentation we formally work. It is easily seen that we can write a formula φ of MSO such that $\mathbf{D} \models \varphi$ if, and only if, $\mathbf{D}$ is a decorated expansion of π and the underlying expansion $\mathbf{A}$ is acyclic. Let q be the quantifier rank of φ and let Q be the number of distinct MSO-types of quantifier rank at most q. Note that the values of q and Q are determined by π and do not depend on the choice of the expansion $\mathbf{A}$.

For $x \in T$, we write $\mathbf{D}_x$ for the substructure of $\mathbf{D}$ induced by the subtree of $\mathbf{T}$ rooted at x, and the elements related to nodes of this subtree by B. Note that the only elements that $\mathbf{D}_x$ shares with the rest of $\mathbf{D}$ are in B_x. We write $\mathbf{D}[x/\mathbf{D}']$ for the structure obtained from $\mathbf{D}$ by replacing $\mathbf{D}_x$ by $\mathbf{D}'$. That is, it is the disjoint union of the structure $\mathbf{D} \setminus \mathbf{D}_x$, obtained by removing $\mathbf{D}_x$ from $\mathbf{D}$, with the structure $\mathbf{D}'$ while identifying the elements in $B_r^{\mathbf{D}'}$ (where r is the root of $\mathbf{D}'$) with $B_x^{\mathbf{D}}$. It is then an easy consequence of Lemma 2.3 that $\mathbf{D} \equiv^q \mathbf{D}[x/\mathbf{D}']$ if $\mathbf{D}_x \equiv^q \mathbf{D}'$. In particular this implies that if $\mathbf{D}$ is an acyclic decorated expansion then $\mathbf{D}[x/\mathbf{D}']$ is also an acyclic expansion. For $x, y \in T$, we write $y \prec x$ to denote that y is an ancestor of x in $\mathbf{T}$.

We begin with an informal account of the proof of Lemma 5.2. The idea is to start with an acyclic expansion $\mathbf{D}$ that maps homomorphically to a simple path of length N, for some large enough N. This enables us to find a pair $x, y \in T$ such that $\mathbf{D}_x \equiv^q \mathbf{D}_y$ and $y \prec x$. We can then *pump*, i.e. consider the expansions $\mathbf{D}' := \mathbf{D}[x/\mathbf{D}_y]$ and $\mathbf{D}'' := \mathbf{D}[x/\mathbf{D}'_y]$, etc. in order to obtain larger acyclic expansions with longer s-t-paths. If $\mathbf{D}$ itself consisted of a single path, x and y could be chosen so that the pumped expansions themselves consisted of simple paths and we would obtain a set of such paths growing linearly in length. However, if $\mathbf{D}$ contains multiple intersecting paths, the process of pumping may create new paths, including ones shorter than N. Moreover, in order to ensure that *all* paths in the new expansion are affected by pumping, it is not sufficient to choose one pumping pair (x, y), rather we need pairs intersecting (in a suitable way) all s-t-paths in $\mathbf{D}$. Unfortunately, these pairs may overlap and we need to define the process of pumping carefully.

The difficult part of the proof is therefore to choose the set of pairs (x, y) we want to use, and to define the process of pumping carefully. In the construction outlined below, we show how such a set of pairs can be found such that after repeating the pumping process n times, every s-t-path has length at least n and at most n^c, for some $c \in \mathbb{N}$ that depends on $\mathbf{D}$ but not on n. This is enough to prove the lemma. We begin by giving a definition of pumping for a set C of pairs (x, y) which form an *anti-chain* in $\mathbf{D}$ (in a sense we make precise below). We then use this to inductively define the pumped expansions for more general sets C.

Pumping at an antichain: Let $\mathbf{D} = (\mathbf{T}, B, L)$ be a decorated expansion and $C \subseteq T^2$ a set of pairs (x, y) such that $y \prec x$ and if $(x, y), (x', y') \in C$ then $x \neq x'$ and $y \not\preceq y'$. We define the expansions $\mathbf{D}_n^C$ by induction on n: $\mathbf{D}_0^C := \mathbf{D}$ and $\mathbf{D}_{n+1}^C := \mathbf{D}[x/(\mathbf{D}_n^C)_y : (x, y) \in C]$.

In other words, $\mathbf{D}_n^C$ is obtained from $\mathbf{D}$ by pumping each pair (x, y) in C simultaneously n times. Since, for distinct pairs (x, y) and (x', y'), y and y' (and hence also x and x') are incomparable, this is well-defined. We now use this to define pumping for sets of pairs C which are not necessarily incomparable. To be specific, suppose that $C \subseteq T^2$ is a set of pairs (x, y) with $y \prec x$ and $x \neq x'$ for distinct pairs (x, y) and (x', y'). We define a partial order on C by letting $(x, y) \sqsubset (x', y')$ just in case $y' \prec y$ and let $\mathrm{ht}(x, y)$ denote the length of the maximal $\sqsubset$-chain below the pair (x, y). Let m be the maximal value of $\mathrm{ht}(x, y)$ among all pairs in C. Write C^p for the set $\{(x, y) \in C : \mathrm{ht}(x, y) = p\}$.

Pumping: We define the pumped expansions by induction on p: $\mathbf{D}_n^0 = \mathbf{D}_n^{C^0}$ and $\mathbf{D}_n^{p+1} = (\mathbf{D}_n^p)_n^{C^{p+1}}$. Finally, let $\mathbf{D}_n^C$ denote $\mathbf{D}_n^m$.

Intuitively, given $\mathbf{D}$ and C we pump $\mathbf{D}$ by working bottom-up in $\mathbf{D}$ and replacing recursively for each pair $(x, y) \in C$ the tree rooted at x by the tree rooted at y and repeat n times. Note that if C is chosen so that for each $(x, y) \in C$, $\mathbf{D}_x \equiv^q \mathbf{D}_y$, then we also have $\mathbf{D}_n^C \equiv^q \mathbf{D}$. In particular, $\mathbf{D}_n^C$ is an acyclic expansion of π. The following claim is easily established by induction on p.

Claim. Every s-t-path in $\mathbf{D}_n^C$ is of length at most $n^m \cdot N$.

Let b be the maximal branching degree in any decorated expansion of π (note that this depends only on π) and choose $N \in S$ with $N > b^{Q \cdot K}$, where $K := 2^{k^2} \cdot k^4$. Let $\mathbf{A}$ be an expansion witnessing that a simple path of length N is accepted by π and $\mathbf{D} = (\mathbf{T}, B, L)$ be the corresponding decorated expansion. By the choice of N, every s-t-path P in $\mathbf{A}$ must contain K distinct internal vertices $v_1, \ldots v_K$ such that there is a chain $\boldsymbol{x}_P := x_1 \prec \cdots \prec x_K$ in $\mathbf{T}$ with $\mathbf{D}_{x_i} \equiv^q \mathbf{D}_{x_j}$ for all i, j and $v_i \in B_{x_i}$. Choose for each s-t-path such a chain and let $\Gamma := \{\boldsymbol{x}_P : P \text{ is an } s\text{-}t\text{-path in } \mathbf{A}\}$.

If B_x consists of the elements $a_1, \ldots, a_k$ in order, we say that a path P crosses x at (α, β) (for $1 \leq \alpha < \beta \leq k$) if P contains a_α and a_β and no intermediate element of B_x. For a fixed $\boldsymbol{x}_P$, by the choice of K, we can find a pair (α, β) and a subsequence $\boldsymbol{x}'_P$ of $\boldsymbol{x}_P$ of length at least $2^{k^2} \cdot k^2$ such that for each x in $\boldsymbol{x}'_P$, P crosses x at (α, β). Let Γ' be the collection of the pairs $(\boldsymbol{x}'_P, (\alpha, \beta))$ for $\boldsymbol{x}_P \in \Gamma$.

Distant: Say a pair $(u, v) \subseteq B_y$, for some $y \in T$, is *distant* if for every path P from u to v in $\mathbf{A}$ there is some $(\boldsymbol{x}, (\alpha, \beta)) \in \Gamma'$ such that P crosses each $x \in \boldsymbol{x}$ at (α, β).

By construction, (s, t) is distant. For each $(\boldsymbol{x}, (\alpha, \beta)) \in \Gamma'$ we can choose a pair $x, y \in \boldsymbol{x}$ with $(x, a_1, \ldots, a_k) \in B$ and $(y, b_1, \ldots, b_k) \in B$ such that $y \prec x$ and (a_i, a_j) is distant if, and only if, (b_i, b_j) is distant for all i, j. Indeed, as $\boldsymbol{x}$ has at least $2^{k^2} \cdot k^2$ elements, we have at least k^2 distinct choices for x. This ensures that we can choose C to be a collection of such pairs (x, y), including one from each $(\boldsymbol{x}, (\alpha, \beta)) \in \Gamma'$ such that no two pairs in C share the same first component.

For $u, v \in B_x$, for some $x \in \mathbf{T}$, define the *pumping height* of (u, v) to be the length of the maximal chain (with respect to the order $\sqsubset$) in C below x. The following claim is the key to the pumping argument.

Claim. For all p, n, if the pumping height of (u, v) is at most p and (u, v) is distant then the distance of u and v in $\mathbf{D}_n^p$ is at least n.

In particular, the claim implies that for $n \in \mathbb{N}$, every s-t-path in $\mathbf{D}_n^C$ is of length at least n. As C, and hence m, only depend on the initial choice of $\mathbf{D}$ and not on n, we

have that every s-t-path in $\mathbf{D}_n^C$ is of length at most $n^m \cdot N$. To complete the proof of Lemma 5.2, take $a_1 = N + 1$ and $a_{i+1} = a_i^{m+1}$. □

To complete the proof of Theorem 5.1, consider the class C_S where $S = \{2^{2^{n^2}} : n \in \mathbb{N}\}$ which is clearly decidable in polynomial time. It is easily verified that there is no sequence $(a_i)_{i\in\omega}$ that satisfies the conditions of Lemma 5.2 for this set. Finally, we note also that the restriction of the class C_S to finite structures can be characterised as $\{\mathbf{A} : \mathbf{A}$ finite and $\mathbf{A} \not\to \mathbf{B}\}$ for a fixed infinite structure $\mathbf{B}$. Simply take $\mathbf{B}$ to be the structure formed from the disjoint union of all finite $\mathbf{A} \notin C_S$ by identifying all copies of s and t.

References

1. Open Problems List for the MathsCSP Workshop, Oxford (2006), http://www.cs.rhul.ac.uk/home/green/mathscsp/
2. Afrati, F., Cosmadakis, S., Yannakakis, M.: On Datalog vs. Polynomial Time. Journal of Computer and System Sciences 51, 177–196 (1995)
3. Atserias, A.: The homomorphism preservation property. In: Talk at International Workshop on Mathematics of Constraint Satisfaction, Oxford (2006)
4. Atserias, A., Bulatov, A., Dawar, A.: Affine systems of equations and counting infinitary logic. In: Proc. 34th International Colloquium on Automata, Languages and Programming. LNCS, vol. 4596, pp. 558–570. Springer, Heidelberg (2007)
5. Atserias, A., Dawar, A., Grohe, M.: Preservation under extensions on well-behaved finite structures. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1437–1449. Springer, Heidelberg (2005)
6. Atserias, A., Dawar, A., Kolaitis, P.G.: On preservation under homomorphisms and unions of conjunctive queries. Journal of the ACM 53, 208–237 (2006)
7. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational databases. In: Proc. 9th ACM Symp. on Theory of Computing, pp. 77–90 (1977)
8. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer, Heidelberg (1999)
9. Feder, T., Vardi, M.Y.: Computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. SIAM Journal of Computing 28, 57–104 (1998)
10. Feder, T., Vardi, M.Y.: Homomorphism closed vs existential positive. In: Proc. of the 18th IEEE Symp. on Logic in Computer Science, pp. 311–320 (2003)
11. Grohe, M.: Logic, graphs, and algorithms. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata History and Perspectives, Amsterdam University Press (2007)
12. Hodges, W.: Model Theory. Cambridge University Press, Cambridge (1993)
13. Immerman, N.: Relational queries computable in polynomial time. Information and Control 68, 86–104 (1986)
14. Kolaitis, P.G., Vardi, M.Y.: Conjunctive query containment and constraint satisfaction. Journal of Computer and System Sciences 61, 302–332 (2000)
15. Makowsky, J.A.: Algorithmic uses of the Feferman-Vaught theorem. Annals of Pure and Applied Logic 126, 159–213 (2004)
16. Moschovakis, Y.N.: Elementary Induction on Abstract Structures. North Holland, Amsterdam (1974)
17. Rossman, B.: Existential positive types and preservation under homomorphisisms. In: 20th IEEE Symposium on Logic in Computer Science, pp. 467–476 (2005)
18. Vardi, M.Y.: The complexity of relational query languages. In: Proc. of the 14th ACM Symp. on the Theory of Computing, pp. 137–146 (1982)