First-Order Queries over One Unary Function

Arnaud Durand[1] and Frédéric Olive[2]

[1] Equipe de Logique Mathématique, CNRS UMR 7056 - Université Denis Diderot, 2 place Jussieu, 75251 Paris Cedex 05, France - durand@logique.jussieu.fr
[2] LIF, CNRS UMR 6166 - Université de Provence, CMI, 39 rue Joliot Curie F-13453 Marseille Cedex 13, France - olive@lif.univ-mrs.fr

Abstract. This paper investigates the complexity of query problem for first-order formulas on quasi-unary signatures, that is, on vocabularies made of a single unary function and any number of monadic predicates. We first prove a form of quantifier elimination result: any query defined by a quasi-unary first-order formula can be equivalently defined, up to a suitable linear-time reduction, by a quantifier-free formula. We then strengthen this result by showing that first-order queries on quasi-unary signatures can be computed with constant delay i.e. by an algorithm that has a precomputation part whose complexity is linear in the size of the structure followed by an enumeration of all solutions (i.e. the tuples that satisfy the formula) with a constant delay (i.e. depending on the formula size only) between each solution. Among other things, this reproves (see [7]) that such queries can be computed in total time $f(|\varphi|).(|S| + |\varphi(S)|)$ where S is the structure, φ is the formula, $\varphi(S)$ is the result of the query and f is some fixed function.
The main method of this paper involves basic combinatorics and can be easily automatized. Also, since a forest of (colored) unranked tree is a quasi-unary structure, all our results apply immediately to queries over that later kind of structures.
Finally, we investigate the special case of conjunctive queries over quasi-unary structures and show that their combined complexity is not prohibitive, even from a dynamical (enumeration) point of view.

1 Introduction

The complexity of logical query languages is a well-studied field of theoretical computer science and database theory. Understanding the complexity of query evaluation for a given language is a good way to measure its expressive power. In this context, first-order logic and its fragments are among the most interesting and studied such languages.

In full generality, the data complexity of first-order queries is in AC^0 hence in polynomial time [15] (see also [11]). However, the size of the formula appears as a major ingredient in the exponent of the polynomial. Hence, taking into account the sizes of both the structure and the formula, the combined complexity becomes highly intractable, even for small formulas. Nevertheless, tractability results have been obtained for natural query problems defined by restricting either the logic or the set of structures. This is the case, for example, for acyclic conjunctive queries [16, 13], or for full first-order queries on relational structures of bounded degree [14, 5, 12] or on tree-decomposable structures [8] (see also [7]).

A quasi-unary signature consists of one unary function and any number of monadic predicates. First-order logic over quasi-unary structures has been often studied and some of its aspects are quite well understood. The satisfiability problem for this kind of first-order formulas is decidable, while by adding just one more unary function symbol in the vocabulary we can interpret graphs, and hence all the first-order logic. In particular, first-order logic over two unary functions structures is undecidable (even for formulas with one variable [9]). In [3], it is proved that the spectrum (i.e. the set of cardinalities of the finite models) of formulas over one unary function are ultimately periodic. This result has been generalized in [10] even to the case of spectra of monadic second-order formulas.

In this paper, we continue the study of first-order logic on quasi-unary vocabularies and show some new structural properties that have interesting consequences on the complexity of query problems for such languages. Our first result shows that it is possible to eliminate variables in first-order formulas on quasi-unary vocabularies at reasonable cost while preserving the answers of the queries. More precisely, given a quasi-unary structure S and a first-order formula φ, one can construct a quantifier-free formula φ' and, in linear time in the size S, a new quasi-unary structure S' such that the results of the queries $\varphi(S)$ and $\varphi'(S')$ are the same. The method used to prove this result is mainly based on combinatorial arguments related to covering problems.

Then, as in [5], we explore the complexity of query evaluation from a dynamical point of view: queries are seen as enumeration problems and the complexity is measured in terms of delay between two successive outputs (i.e. tuples that satisfy the formula). Such an approach provides very precise information on the complexity of query languages: by adding up the delays, it makes it possible to obtain tight complexity bounds on complexity evaluation (in the classical sense) but also to measure how regular this process is. This latter kind of information is useful, for example, for query answering "on demand".

Taking as a starting point the quantifier elimination method, we show that a first-order query φ on a quasi-unary structure S can be computed with constant delay i.e. by an algorithm that has a precomputation part whose complexity is linear in the size of the structure, followed by an enumeration of all the tuples that satisfy the formula with a constant delay (i.e. depending on the formula size only) between two of them. Among other things, this gives an alternative proof that such queries can be computed in total time $f(|\varphi|).(|S| + |\varphi(S)|)$ where f is some fixed function, hence the complexity is linear (see [7] remarking that the graph of one unary function is of tree-width two). One nice feature of quasi-unary structures is their proximity to trees: any forest of (colored) ranked or unranked tree is a quasi-unary structure, hence, all our results immediately apply to queries over trees. Several recent papers have investigated query answering from an enumeration point-of-view for monadic second-order logic on trees (see [1, 2]).

They mainly prove that the results of MSO queries on binary trees or on tree-like structures can be enumerated with a linear delay (in the size of the next output) between two consecutive solutions. The methods used in these papers rely on tree-automata techniques and some of these results apply to our context. However, our goal in this paper, is to prove strong structural properties of logical formulas (such as quantifier elimination) in this language and show how these properties influence the complexity of query answering.

The paper is organized as follows. In Sect. 2, main definitions about query problems, reductions and enumeration complexity are given. A normal form for formula over quasi-unary vocabularies is also stated. In Sect. 3 the combinatorial material to prove the main results are introduced. Then, in Sect. 4, the variable elimination theorem for first-order formula on quasi-unary vocabulary is proved. Section 5, is devoted to query evaluation and our result about enumeration of query result is proved. Finally, in Sect. 6, the special case of conjunctive queries is investigated.

2 Preliminaries

Definitions. All formulas considered in this paper belong to first-order logic, denoted by FO. The FO-formulas written over a same signature σ are gathered in the class FO^{σ}. The *arity* of a first-order formula φ, denoted by $\mathrm{arity}(\varphi)$, is the number of free variables occuring in φ. We denote by $\mathrm{FO}(d)$ the class of FO-formulas of arity d. When the prenex form of a FO-formula φ involves at most q quantifiers, we say that it belongs to FO_q. Combining these notations, we get for instance that $\mathrm{FO}_0^E(d)$ is the class of quantifier-free formulas of arity d and of signature $\{E\}$.

Given a signature σ, we denote by $\textsc{struc}(\sigma)$ the class of *finite* σ-structures. The domain of a structure S is denoted by $\mathrm{dom}(S)$. For each $S \in \textsc{struc}(\sigma)$ of domain D and each $\varphi \in \mathrm{FO}^{\sigma}(d)$, we set:

$$\varphi(S) = \{(a_1, \ldots, a_d) \in D^d : (S, a_1, \ldots, a_d) \models \varphi(x_1, \ldots, x_d)\}.$$

Two σ-formulas φ, ψ of same arity are said *equivalent* if $\varphi(S) = \psi(S)$ for any σ-structure S. We then write $\varphi \sim \psi$.

Let $\mathcal{C} \subseteq \textsc{struc}(\sigma)$ and $\mathcal{L} \subseteq \mathrm{FO}^{\sigma}$. The *query problem* for $\mathcal{C}$ and $\mathcal{L}$ is the following:

$\textsc{Query}(\mathcal{C}, \mathcal{L})$

input: A structure $S \in \mathcal{C}$ and a formula $\varphi(\overline{x}) \in \mathcal{L}$ with free-variables $\overline{x}$;
parameter: the size $|\varphi|$ of $\varphi(\overline{x})$;
output: $\varphi(S)$.

Instances of such a problem are thus pairs $(S, \varphi) \in \mathcal{C} \times \mathcal{L}$.

Most of the complexity results of this paper will be expressed in terms of the structure size, considering the size of the formula as a parameter. This explain the definition of a query problem as a parameter problem.

When φ has no free variable then the boolean query problem is also known as the *model-checking problem* for $\mathcal{L}$, often denoted by $\mathrm{MC}(\mathcal{L})$. Most of the time, $\mathcal{C}$ will simply be the class of all σ-structures $\text{STRUC}(\sigma)$ for a given signature σ and restriction will only concern formulas. In this case, the query problem is denoted $\text{Query}(\mathcal{L})$ and its instances are called *$\mathcal{L}$-queries.*

Reductions, complexity and enumeration problems. The basic model of computation used in this paper is standard Random Access Machine with addition. In the rest of the paper, the *big-O* notation $O_k(m)$ stands for $O(f(k).m)$ for some function f. Such a notation is used to shorten statements of results, especially when k is a parameter whose exact value is difficult to obtain. However, when this value can be made precise, we use the classical *big-O* notation. The definition below specifies the notion of reductions between query problems.

Definition 1. *We say that* $\text{Query}(\mathcal{C}, \mathcal{L})$ linearly reduces *to* $\text{Query}(\mathcal{C}', \mathcal{L}')$ *if there exist two recursive functions $f : \mathcal{C} \times \mathcal{L} \to \mathcal{C}'$ and $g : \mathcal{L} \to \mathcal{L}'$ such that:*

- *if $(S, \varphi) \in \mathcal{C} \times \mathcal{L}$ and $(S', \varphi') = (f(S, \varphi), g(\varphi))$, then $dom(S) = dom(S')$ and $arity(\varphi) = arity(\varphi')$;*
- *the results of the queries are the same: $\varphi(S) = \varphi'(S')$;*
- *the function $f : (S, \varphi) \mapsto S'$ is computable in time $O_{|\varphi|}(|S|)$.*

(There is no requirement on g, but its computability.)

This reduction is a kind of fixed-parameter linear reduction with some additional constraints. Notice that this definition differs from the usual notion of interpretation: here, the interpretive structure S' depends both on the structure S *and on the formula* φ to be interpreted (notice also that, in the sequel, S' will always be an extension of S with only new additional monadic predicates).

In Sect. 5, query evaluation is considered as an enumeration problem. A non boolean query problem $\text{Query}(\mathcal{C}, \mathcal{L})$ is *enumerable with constant delay* if one can compute all its solutions in such a way that the delay beween the beginning of the computation and the first solution, between two successive solutions, and between the last solution and the end of the computation are all constant. We denote by Constant-Delay the class of query problems which are enumerable with constant delay. This class is not robust: a problem which is in Constant-Delay when represented with a given data-structure may not be in this class anymore for another presentation. Nevertheless, this definition will be relevant to forthcoming technical tasks. The complexity class that is of real interest was first introduced in [5]: the class Constant-Delay(*lin*) collects query problems that can be enumerated as previously described after a step of preproccessing which costs a time at most linear in the size of the input. This class is robust. A more formal definition of it is given in the following.

Definition 2. *An query problem* $\text{Query}(\mathcal{C}, \mathcal{L})$ *is computable* within constant delay and with linear precomputation *if there exists a RAM algorithm $\mathcal{A}$ which, for any input (S, φ), enumerates the set $\varphi(S)$ in the following way:*

1. $\mathcal{A}$ uses linear space
2. $\mathcal{A}$ can be decomposed into the two following successive steps
   (a) PRECOMP($\mathcal{A}$) *which performs some precomputations in time $O(f(\varphi).|S|)$ for some function f, and*
   (b) ENUM($\mathcal{A}$) *which outputs all elements in $\varphi(S)$ without repetition within a delay bounded by some constant* DELAY($\mathcal{A}$) *which depends only on $|\varphi|$ (and not on S). This delay applies between two consecutive solutions and after the last one.*

The complexity class thus defined is denoted by CONSTANT-DELAY*(lin).*

The following fact is immediate.

Fact 3. *If* QUERY($\mathcal{C}, \mathcal{L}$) $\in$ CONSTANT-DELAY*(lin), then each query problem that linearly reduces to* QUERY($\mathcal{C}, \mathcal{L}$) *also belongs to* CONSTANT-DELAY*(lin).*

Logical normalization. We now specify the kind of structures and formulas that are considered in this paper. A *unary* signature contains only unary relation symbols. A *quasi-unary* signature is obtained by enriching a unary signature with a single unary function symbol. That is, quasi-unary signatures have the shape $\{f, \overline{U}\}$, where f is a unary function symbol and $\overline{U}$ is a tuple of unary relation symbols.

For any signature σ, a *unary enrichment* of σ is a signature obtained from σ by adding some new unary relation symbols. Therefore, unary enrichments of quasi-unary signatures are also quasi-unary signatures. Structures over quasi-unary (resp. unary) signatures are called *quasi-unary* (resp. *unary*) structures.

Notice that structures over one unary function are disjoint collections of "whirlpools": upside down trees whose roots come together at a cycle (see Fig. 1). Therefore, considering the case where all the cycles consist of a single node, we see that the set of quasi-unary structures includes that of (colored) forests and, *a fortiori*, that of (colored) trees.

Let us consider a quantifier-free disjunctive formula φ over a quasi-unary signature. For any $y \in \mathrm{var}(\varphi)$, we can "break" φ into two pieces ψ and θ, in such a way that y does not occur in ψ while it occurs in θ in a very uniform way. More precisely, each quantifier-free disjunctive formulas over a quasi-unary signature can be written, up to linear reduction, under the form (1) below. This normalization is formally stated in the next proposition. It will provide us with a key tool to eliminate quantified variables in any first-order formulas of quasi-unary signature.

Proposition 4. *Let $\mathrm{FO}_0^{qu}[\vee]$ be the class of quantifier-free disjunctive formulas over a quasi-unary signature. Then,* QUERY($\mathrm{FO}_0^{qu}[\vee]$) *linearly reduces to the problem* QUERY($\mathcal{L}_0$)*, where $\mathcal{L}_0$ is the class of $\mathrm{FO}_0^{qu}[\vee]$-formulas that fit the shape:*

$$\psi(\overline{x}) \vee [(f^m y = f^n z \wedge U(y)) \to \bigvee_i f^{m_i}(y) = f^{n_i}(\overline{x})], \tag{1}$$

where ψ is a quantifier-free disjunctive formula, $0 \leq m_1 \leq \cdots \leq m_k \leq m$, $z \in \overline{x}$ and U is a unary relation.

Note that an essential consequence of this normalization is that on the left-hand side of the implication, only one atomic formula involving function f and variable y appears. This will make the elimination of variable y easier.

3 Acyclic representation

Because of the normal form stated in Proposition 4, a corner-stone of our oncoming results lies on the way we handle formulas of the type:

$$\varphi(\overline{x}) \equiv \forall y : (f^m y = a \wedge U(y)) \rightarrow \bigvee_{i\in[k]} f^{m_i} y = c_i \tag{2}$$

where $0 \leq m_1 \leq \cdots \leq m_k \leq m$, U is a unary relation and $a, c_1, \dots, c_k$ are terms in $\overline{x}$. In order to deal efficiently (from a complexity point of view) with such formulas, we introduce in this section a data structure related to the tuple $\overline{f} = (f^{m_1}, \dots, f^{m_k})$ and the set $X = f^{-m}(a) \cap U$. Then we show that this structure allows for a fast computation of some combinatorial objects – the samples of $\overline{f}$ over X – which are the combinatorial counterparts of the logical assertion (2).

Until the end of this section, f denotes a unary function on a finite domain D, X is a subset of D and $0 \leq m_1 \leq m_2 \leq \cdots \leq m_k$ are nonnegative integers. Furthermore, $\overline{f}$ denotes the tuple $(f^{m_1}, \dots, f^{m_k})$. We associate a labelled forest to $\overline{f}$ and X in the following way (and we right now refer the reader to Fig. 1 and 2 to follow this definition):

- The set of vertices of the forest is partitioned into $k+1$ sets $L_0, L_1, \dots, L_k$ corresponding to the sets $X, f^{m_1}(X), \dots, f^{m_k}(X)$.
- For each i, the label function ℓ is a bijection from L_i to $f^{m_i}(X)$
- There is an edge from y to x if and only if there exists $i \in [0..k]$ such that: $x \in L_i$, $y \in L_{i+1}$ and $f^{m_{i+1}-m_i}\ell(x) = \ell(y)$.

Then we enrich this labelled forest with the data $\mathsf{root}, \mathsf{next}, \mathsf{back}, \mathsf{height}$ defined from some depth-first traversal of the forest:

- root is the first root of the forest visited by the depth-first traversal.
- $\mathsf{next}(s)$ is the first descendent of s visited after s, if such a node exists. Otherwise, $\mathsf{next}(s) = s$.
- $\mathsf{back}(s)$ is the first node visited after s which is neither a parent nor a descendant of s, provided that such a node exists. Otherwise, $\mathsf{back}(s) = s$.
- $\mathsf{height}(s)$ is the height of s in $\mathcal{F}$. That is, $\mathsf{height}(s)$ is the distance from s to a leaf.

The resulting structure is called the *acyclic representation of* $\overline{f}$ *over* X. We denote it by $\mathcal{F}(\overline{f}, X)$. Besides, we denote by $\mathsf{height}(\mathcal{F})$ the height of $\mathcal{F}(\overline{f}, X)$, that is the maximum of the values $\mathsf{height}(s)$, for s in $\mathcal{F}$, and we call *branch* of the forest any path that connects a root to a leaf (i.e. any maximal path of a tree of $\mathcal{F}$).

Example 5. A function $f : D \to D$, where $D = \{1, \ldots, 16\}$, is displayed in Fig. 1. The acyclic representation of $\overline{f} = (f, f, f^2, f^4, f^5, f^7)$ over the set $D \setminus \{3, 8, 10, 14, 16\}$ is given in Fig. 2.

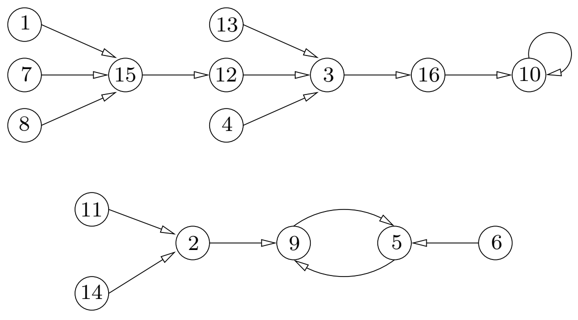

Fig. 1. A function $f : [16] \to [16]$

We let the reader check that the following Lemma is true (the acyclic representation can be easily built by sorting and running through image sets $f^{m_i}(X)$, for all $i \leq k$).

Lemma 6. *The acyclic representation of* $\overline{f} = (f^{m_1}, \ldots, f^{m_k})$ *over* X *can be computed in time* $O(k.m_k.|X|)$.

Let us come back to the combinatorial objects (the samples - this terminology comes from the earlier papers [6, 4]) mentioned at the beginning of this section. For each $m \geq 0$ and each $c \in D$, we denote by $f^{-m}(c)$ the set of pre-images of c by f^m. That is: $f^{-m}(c) = \{x \in D \mid f^m x = c\}$. We set:

Definition 7. *Let* $P \subseteq [k]$ *and* $(c_i)_{i \in P} \in D^P$. *The tuple* $(c_i)_{i \in P}$ *is a* sample of $\overline{f}$ over X *if*

$$X \subseteq \bigcup_{i \in P} f^{-m_i}(c_i).$$

This sample is minimal *if, moreover, for all* $j \in P$*:*

$$X \not\subseteq \bigcup_{i \in P \setminus \{j\}} f^{-m_i}(c_i).$$

Notice that each sample contains a minimal sample: if $(c_i)_{i \in P}$ is a sample of $\overline{f}$ over X, then $(c_i)_{i \in P'}$ is a minimal sample of $\overline{f}$ over X, where P' is obtained from P by iteratively eliminating the j's such that $X \subseteq \bigcup_{i \in P \setminus \{j\}} f^{-m_i}(c_i)$.

Samples provide a combinatorial interpretation of some logical assertions. It is easily seen that the assertion $\forall y \in X : \bigvee_{i \in [k]} f^{m_i} y = c_i$ exactly means that

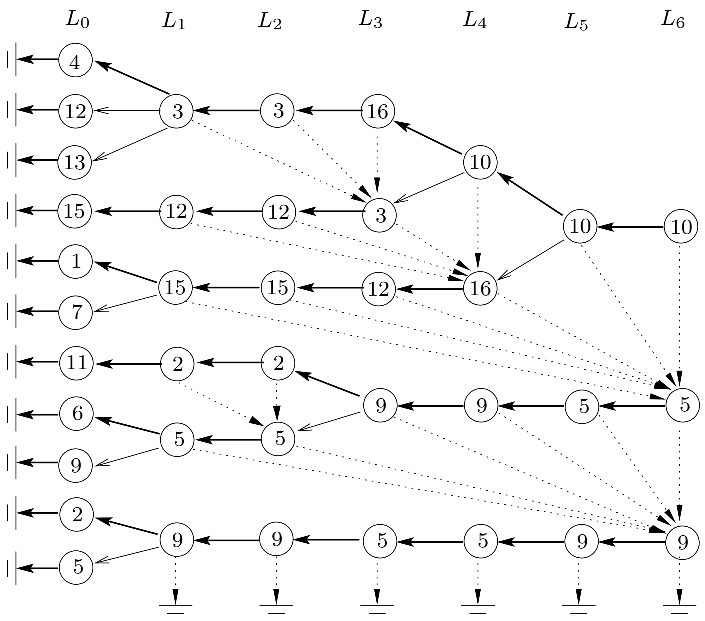

Fig. 2. The acyclic representation of a tuple $\overline{f} = (f^{m_1}, \ldots, f^{m_k})$ over a set X. Here, f is the function drawn in Fig. 1, $\overline{f}$ is the tuple $(f, f, f^2, f^4, f^5, f^7)$ and X is the set $[16] \setminus \{3, 8, 10, 14, 16\}$. Each L_i at the top of the figure corresponds both to the set $f^{m_i}(X)$ and to the set of nodes of height i in the forest.

$(c_i)_{i\in[k]}$ is a sample of $\overline{f}$ over X. In particular, assertion (2) holds if, and only if, $(c_i)_{i\in[k]}$ is a sample of $(f^{m_i})_{i\in[k]}$ over $f^{-m}(a)\cap U$. This equivalence will yields the variable elimination result of Sect. 4.

Another characterization of samples connects this notion to that of acyclic representation: Let $(x_i)_{i\in P}$ be a sequence of pairwise distinct nodes in $\mathcal{F}(\overline{f}, X)$, where $P \subseteq [k]$. We say that $(x_i)_{i\in P}$ is a *minimal branch marking* of $\mathcal{F}(\overline{f}, X)$ if each branch of the forest contains a unique x_i and if, furthermore, each x_i lies on the level L_i of the forest. Clearly, $(x_i)_{i\in P}$ is a minimal branch marking of $\mathcal{F}(\overline{f}, X)$ iff the tuple $(\ell(x_i))_{i\in P}$ is a minimal sample of $\overline{f}$ over X (recall $\ell(x)$ is the label of x in $\mathcal{F}(\overline{f}, X)$). Roughly speaking: minimal samples of $\overline{f}$ over X are exactly sequences of values that label minimal branch markings of $\mathcal{F}(\overline{f}, X)$.

The next lemma states that both the total number of minimal samples and the time required to compute all of them can be precisely controlled. According to the equivalence above, evaluating all minimal samples of $\overline{f}$ over X amounts to compute all minimal branch markings of $\mathcal{F}(\overline{f}, X)$. This underlies an efficient procedure to carry out such an evaluation. We will not give the proof of the next Lemma: it has already been proved in a more general context in [6, 4].

Lemma 8. *There are at most $k!$ minimal samples of $\overline{f}$ over X and their set can be computed in time $O_k(|X|)$.*

We now slightly modify the presentation of minimal samples in order to manipulate them more conveniently inside formulas. The levels (with a few other information) of the acyclic representation are introduced explicitly.

Definition 9. *Let m be an integer, $a \in D$ and $X_a \subseteq f^{-m}(a)$. The sets $L_i^{P,h}$, with $P \subseteq [k]$, $h \leq k!$ and $i \in P$, are defined as the minimal sets such that, for all $a \in f^m(D)$:*

if $(s_i)_{i\in P}$ is the h^{th} minimal sample of $\overline{f}$ over X_a then, for all $i \in P$, $s_i \in L_i^{P,h}$.

Note that if the image set $f^m(D)$ is reduced to a single element i.e. $f^m(D) = \{a\}$, then each $L_i^{P,h}$ contains *at most* one element. We will use this property in the sequel. Note also that each $x \in L_i^{P,h}$ belongs to the level L_i of the acyclic representation of $\overline{f}$: for fixed i, sets $L_i^{P,h}$ are refinements of level L_i.

Not all sets $L_i^{P,h}$ for $P \subseteq [k]$ and $h \leq k!$ are non empty and hence need to be defined. We denote by $\mathrm{P\scriptstyle{H}}$ the set of such (P, h) for which the sets $L_i^{P,h}$ are not empty ($i \in P$). The following lemma details how to compute the sets $L_i^{P,h}$.

Lemma 10. *With the notation of Definition 9, the collection of sets $L_i^{P,h}$ for $i \in P$ and $(P, h) \in \mathrm{P\scriptstyle{H}}$ can be computed in time $O_k(|D|)$.*

Proof. We first set $L_i^{P,h} = \emptyset$ for all $P \subseteq [k], h \leq k!$ and $i \in P$. By Lemma 8, for each $a \in f^m(D)$, one can compute the set of the at most $k!$ minimal samples of X_a in time $O_k(|X_a|)$ and assign a different number $h \leq k!$ to each. Now, running through these samples, if $(s_i)_{i\in P}$ is the h^{th} of them, one add each s_i to set $L_i^{P,h}$. This step needs to be repeated for each $a \in f^m(D)$. Since all sets X_a are pairwise disjoints and since their union is included in D, the whole process requires time $O_k(|D|)$ which is the expected time bound. Note that, for indices $(P,h) \notin \text{P}\textsc{h}$, the sets $L_i^{P,h}$ remain empty. □

4 Variable elimination

We are now in a position to eliminate quantifiers in a first-order formula to be evaluated on a quasi-unary structure. As mentioned in Sect. 3, a first step must be to eliminate y in a formula φ of the following form:

$$\varphi(\overline{x}) \equiv \forall y : (f^m y = a \wedge U(y)) \rightarrow \bigvee_{i\in[k]} f^{m_i} y = c_i$$

where $0 \leq m_1 \leq \dots \leq m_k \leq m$, U is a unary relation and $a, c_1, \dots, c_k$ are terms in $\overline{x}$. To deal with this task, we understand $\varphi(\overline{x})$ as meaning:

$$(c_i)_{i\in[k]} \text{ is a sample of } \overline{f} \text{ over } f^{-p}(a) \cap U.$$

That is, since every sample contains a minimal sample:

$$(c_i)_{i\in[k]} \text{ contains a minimal sample of } \overline{f} \text{ over } f^{-m}(a) \cap U$$

or, equivalently:

$$\exists P \subseteq [k] \text{ such that } (c_i)_{i\in P} \text{ is a minimal sample of } \overline{f} \text{ over } f^{-m}(a) \cap U$$

From the previous section, this assertion can now be stated as:

$$\exists h \leq k!\ \exists P \subseteq [k] \text{ such that } \forall i \in P\text{: } L_i^{P,h}(c_i) \text{ and } f^{m-m_i} c_i = a$$

This is more formally written:

$$\bigvee_{(P,h)\in \text{P}\textsc{h}} \bigwedge_{i\in P} (L_i^{P,h}(c_i) \wedge f^{m-m_i} c_i = a).$$

All predicates in this last assertion can be computed in linear time and the disjunction and conjunction deal with a constant number (depending on k) of objects. Variable y is now eliminated. This provides a general framework to iteratively eliminate variables.

Lemma 11. *Every* FO_1*-query of the form* $(\forall y \varphi(\overline{x}, y), S)$*, where* $\varphi(\overline{x}, y)$ *is a quantifier-free disjunctive formula over a quasi-unary signature* σ*, linearly reduces to an* FO_0*-query over a quasi-unary signature.*

Proof. Let S be a quasi-unary structure and $\varphi(\overline{x}, y)$ be a quantifier-free disjunctive formula. Thanks to Proposition 4, $\forall y \varphi(\overline{x}, y)$ may be considered, up to linear reduction, as fitting the form:

$$\psi(\overline{x}) \vee \forall y[\,(f^p y = f^q z \wedge U(y)) \rightarrow \bigvee_{i\in[k]} f^{m_i}(y) = f^{n_i}(\overline{x})\,]$$

where $z \in \overline{x} \cup \{y\}$. Assume $z \neq y$ (otherwise the proof gets simpler). Denoting $\overline{f} = (f^{m_1}, \ldots, f^{m_k})$, the second disjunct simply means that $(f^{n_i}(\overline{x}))_{i\in[k]}$ is a sample of $\overline{f}$ over $f^{-p}(f^q(z)) \cap U$. From what has been said before, this implies that there exists $P \subseteq [k]$ such that $(f^{n_i}(\overline{x}))_{i\in P}$ is a minimal sample of $\overline{f}$ over $f^{-p}(f^q(z))$. Recalling Definition 9 and the discussion that followed, one can write:

$$\langle S, (L_i^{P,h})_{\mathrm{PH}}\rangle \models \psi(\overline{x}) \vee \bigvee_{(P,h)\in \mathrm{PH}} \bigwedge_{i\in P} L_i^{P,h}(f^{n_i}(\overline{x})) \wedge f^{p-m_i+n_i}(\overline{x}) = f^q(z)$$

Variable y is well eliminated. From Lemma 10, the collection of sets $L_i^{P,h}$ are linear time computable from structure S. This concludes the proof. □

Theorem 12. *Each non-Boolean (resp. Boolean)* FO*-query over a quasi-unary signature linearly reduces to a* FO_0*-query (resp.* FO_1*-query) over a quasi-unary signature.*

Proof. The proof is by induction on the number k of quantified variables of the query. Most of the difficulty already appears for the case $k = 1$. Let $(\varphi(\overline{z}), S) \in \mathrm{FO}_1^\sigma \times \mathrm{STRUC}(\sigma)$ and $\overline{z}$ be the nonempty tuple of free variables of φ. Since $\exists y\ \psi$ is equivalent to $\neg(\forall x\ \neg\psi)$, one may suppose that φ is of the form $\pm\forall y \varphi(\overline{z}, y)$ (where $\pm$ means that one negation $\neg$ may occur). The conjunctive normal form for the matrix of φ must be computed, and, since universal quantification and conjunction commute, one can put the formula in the form $\varphi(\overline{z}) \equiv \pm \bigwedge_\alpha \forall y \varphi_\alpha(\overline{z}, \overline{x}, y)$, where each φ_α is a quantifier-free disjunctive formula. By Lemma 11, we know that the query $(\bigwedge_\alpha \forall y \varphi_\alpha(\overline{z}, y), S)$ linearly reduces to a quantifier-free query $(\psi(\overline{z}, \overline{x}), S')$ over a quasi-unary signature. This concludes this case.

Assume the result for $k \geq 1$ and prove it for $k + 1$. For the same reason as above, $\varphi(\overline{z})$ can be put under the form $\varphi(\overline{z}) \equiv \overline{Q}\overline{x} \pm \bigwedge_\alpha \forall y \varphi_\alpha(\overline{z}, \overline{x}, y)$, where each φ_α is a quantifier-free disjunctive formula and $\overline{x}$ is a tuple of k quantified variables. Again, from Lemma 11 query $(\bigwedge_\alpha \forall y \varphi_\alpha(\overline{z}, \overline{x}, y), S)$ linearly reduces to a quantifier-free query $(\psi(\overline{z}, \overline{x}), S')$ and then, $(\varphi(\overline{z}), S)$ linearly reduces to the k variable query $(\overline{Q}\overline{x}\psi(\overline{z}, \overline{x}), S')$. □

5 Query enumeration

In this section we prove that each first-order query over a quasi-unary structure $\langle D, f, \overline{U}\rangle$ can be enumerated with constant delay after a linear time preprocessing step. The proof involves the cost of the enumeration of all the elements of D whose images by different powers of f (i.e. by functions of the form f^i)

avoid a fixed set of values. It appears that the elements thus defined can be enumerated with constant delay, provided the inputs are presented with the appropriate data structure. Let us formalize the problem, before stating the related enumeration result.

Authorized Values

input:	the acyclic representation of a k-tuple $\overline{f} = (f^{m_1}, \ldots, f^{m_k})$ over a set $X \subseteq D$, and a set of *forbidden pairs* $\mathsf{F} \subset [k] \times D$;
parameter:	k, $\lvert\mathsf{F}\rvert$;
output:	the set $\mathcal{A} = \{y \in X \mid \bigwedge_{(i,c)\in\mathsf{F}} f^{m_i}y \neq c\}$.

Lemma 13. Authorized Values $\in$ Constant-Delay. *Moreover, the delay between two consecutive solution is* $O(k.\lvert\mathsf{F}\rvert)$.

Proof. Each forbidden pair $(i, c) \in \mathsf{F}$ is either inconsistent (i.e. $f^{-m_i}(c) = \emptyset$) or corresponds to a node s of $\mathcal{F}(\overline{f}, X)$ such that $\mathsf{height}(s) = i$ and $\ell(s) = c$. If we denote by forbid those nodes s of $\mathcal{F}(\overline{f}, X)$ for which $(\mathsf{height}(s), \ell(s)) \in \mathsf{F}$, the set $\mathcal{A}$ to be constructed is exactly the set of values $y \in D$ that label leaves of the forest whose branches avoid all forbidden nodes. Therefore, computing $\mathcal{A}$ amounts to finding all the leaves described above.

This can be done by a depth-first search of the forest, discarding those leaves whose visit led to a forbidden node. Furthermore, the search can be notably sped up by backtracking on each forbidden node: indeed, such a node is the root of a subtree whose all leaves disagree with our criteria. This algorithm clearly runs in linear time. Let us show it enumerates solutions with constant delay:

Consider a sequence of p nodes $s_1, \ldots, s_p$ successively visited by the algorithm. If $p > k$, the sequence must contain a leaf or a forbidden node. Indeed, if it does not contain any node of forbid, the algorithm behaves as a usual DFS between s_1 and s_p. Therefore, one node among $s_2, \ldots, s_{k+1}$ has to be a leaf since $\mathsf{height}(\mathcal{F}(\overline{f}, X)) = k$.

Now, if s, s' are two leaves successively returned by the algorithm and if $f_1, \ldots, f_p$ are the forbidden nodes encountered between these two solutions, then the previous remark ensures that the algorithm did not visit more than k nodes between s and f_1, between f_p and s' or between two successive f_i's. The delay between the outputs s and s' is hence in $O(pk)$ and therefore, in $O(k\lvert\mathsf{forbid}\rvert)$. Furthermore, this reasoning easily extends to the delay between the start of the algorithm and the first solution, or between the last solution and the end of the algorithm. This concludes the proof. □

Now we can prove the main result of this section.

Theorem 14. Query(FO$^\sigma$) $\in$ Constant-Delay*(lin) for any quasi-unary signature* σ.

Proof. Because of Theorem 12 and Fact 3, we just have to prove the result for the quantifier-free restriction of FO^σ. We prove by induction on d that every $\mathrm{FO}_0^\sigma(d)$-query can be enumerated with constant delay and linear time precomputation. (Recall σ is a quasi-unary signature.)

The result is clear for $d = 1$: Given an instance $(S, \varphi(x))$ of $\textsc{Query}(\mathrm{FO}_0^\sigma(1))$, the set $\varphi(S)$ can be evaluated in time $O(|S|)$ since φ is quantifier-free. Therefore, the following procedure results in a $\textsc{Constant-Delay}(lin)$-enumeration of $\varphi(S)$:

Algorithm 1 $\textsc{Enumeration}(1, \varphi(y))$

```
1: compute $\varphi(S)$
2: enumerate all the values of $\varphi(S)$
```

Let us now suppose the induction hypothesis is true for $d \geq 1$ and examine the case $d+1$. This case is divided in several steps. Let $(S, \varphi(\overline{x}, y))$ be a $\mathrm{FO}_0^\sigma(d+1)$-query.

Step 1. By standard logical techniques, $\varphi(\overline{x}, y)$ can be written in disjunctive normal form as $\bigvee_\alpha \theta_\alpha(\overline{x}, y)$, where the θ_α's are *disjoint* conjunctive (quantifier-free) formulas (i.e., $\theta_\alpha(S) \cap \theta_\beta(S) = \emptyset$ for $\alpha \neq \beta$). And this normalization can be managed in linear time. But it is proved in [5] that the class $\textsc{Constant-Delay}(lin)$ is stable by disjoint union. Therefore, we do not loose generality when focusing on the $\textsc{Constant-Delay}(lin)$-enumeration of a query of the form $(S, \theta_\alpha(\overline{x}, y))$.

Step 2. One can separate parts of formulas that depends exclusively on $\overline{x}$, then a conjunctive $\mathrm{FO}_0(d+1)$-formula $\theta(\overline{x}, y)$ can be written $\theta(\overline{x}, y) \equiv \psi(\overline{x}) \wedge \delta(\overline{x}, y)$. It is essential to consider tuples $\overline{x}$ that satisfy $\psi(\overline{x})$ but that can also be completed by some y such that $\delta(\overline{x}, y)$ holds. Hence, one considers the equivalent formulation :

$$\theta(\overline{x}, y) \equiv \psi(\overline{x}) \wedge \exists y \delta(\overline{x}, y) \wedge \delta(\overline{x}, y)$$

If we set $\psi_1(\overline{x}) \equiv \psi(\overline{x}) \wedge \exists y \delta(\overline{x}, y)$, formula $\theta(\overline{x}, y)$ can now be written, thanks to Proposition 4:

$$\theta(\overline{x}, y) \equiv \psi_1(\overline{x}) \wedge \delta(\overline{x}, y), \tag{3}$$

where $\psi_1(\overline{x})$ is a conjunctive $\mathrm{FO}_1(d)$-formula and $\delta(\overline{x}, y)$ is a conjunctive $\mathrm{FO}_0(d+1)$-formula of the form $f^m y = f^n \overline{x} \wedge Uy \wedge \bigwedge_{i \in [k]} f^{m_i} y \neq f^{n_i} \overline{x}$.

The idea is to enumerate the $(d+1)$-tuples $(\overline{x}, y)$ satisfying θ by enumerating the d-tuples $\overline{x}$ satisfying $\psi_1(\overline{x})$ and, for each such $\overline{x}$, by enumerating the values y fulfilling $\delta(\overline{x}, y)$. Both these enumerations can be done with constant delay: the first, by inductive hypothesis ; the second by Lemma 13. As we made sure that any tuple satisfying $\psi_1(\overline{x})$ can be completed by at least one convenient y, our enumeration procedure will not explore bad directions in depth. Next step makes this precise.

Step 3. By induction hypothesis, there exists a CONSTANT-DELAY(lin)-enumeration algorithm ENUMERATION(d, φ) for formulas φ of arity d. Then, we get the following CONSTANT-DELAY(lin)-enumeration algorithm for conjunctive formulas of arity $d+1$, together with its linear time precomputation procedure:

Algorithm 2 PRECOMP($d+1, \theta(\overline{x}, y))$)

```
1: write $\theta$ under the form (3)
2: build the complete acyclic representation of $\overline{f} = (f^{m_1}, \dots, f^{m_k})$ over $X = f^{-m}(f^n\overline{x}) \cap U$
3: PRECOMP($d, \psi_1(\overline{x})$)
```

Notice that items 1 and 2 of the above algorithm are carried out in time $O(|X|)$, thanks to Proposition 4 and Lemma 6.

Algorithm 3 ENUMERATION($d+1, \theta(\overline{x}, y)$)

```
1: PRECOMP($d+1, \theta(\overline{x}, y))$)
2: for all $\overline{x}$ in ENUM($d, \psi_1(\overline{x})$) do
3:     for all $y$ in AUTHORIZED VALUES ($\overline{f}, X, (i, f^{n_i}\overline{x})_{i\in[k]}$) do
4:         return $(\overline{x}, y)$
```

Finally, we get a complete CONSTANT-DELAY(lin)-enumeration algorithm for $\mathrm{FO}_0(d+1)$-formula by running successively the algorithms for the disjoint conjunctive formulas $\theta_\alpha(\overline{x}, y)$ obtained in the first step. □

6 Conjunctive queries

So far, we proved that the first-order query problem over quasi-unary signature is "linear-time" computable. However, in full generality the size of the constant (depending on the formula) may be huge: this is essentially due to the variable elimination process that, at each step, may produce an exponentially larger formula. Notice that, once variables are eliminated in formulas, query answering become tractable both in terms of formula and of structure sizes. However, it is not straightforward to know in advance which kind of query admit equivalent formulation in terms of a quantifier-free query of polynomially related size. This amounts to determine the number of expected minimal samples in different situations. This will be the object of further investigation in the extended version of this paper.

In what follows, we examine the very easy particular case of conjunctive queries.

Definition 15. *A first-order formula is existential conjunctive if it uses conjunction and existential quantification only. Conjunctive queries are queries defined by existential conjunctive formulas.*

Conjunctive queries (even union of conjunctive queries) over quasi-unary structures are more tractable from the point of view of answer enumeration. Enumeration of query result is tractable even from the point of view of the constant size. The following result is easy to see by a direct algorithm.

Proposition 16. *The conjunctive query problem over quasi-unary structures belongs to the class* CONSTANT-DELAY *with a delay between two consecutive tuples in* $O(|\varphi|)$.

References

1. G. Bagan. MSO queries on tree decomposable structures are computable with linear delay. Computer Science Logic'06 (this volume), 2006.
2. B. Courcelle. Linear delay enumeration and monadic second-order logic. submitted, 2006.
3. A. Durand, R. Fagin, and B. Loescher. Spectra with only unary function symbols. In M. Nielsen and W. Thomas, editors, *Computer Science Logic, selected paper of CSL'97*, LNCS 1414, pages 111–126. Springer, 1998.
4. A. Durand and E. Grandjean. The complexity of acyclic conjunctive queries revisited. Draft, 2005.
5. A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic*, 2006. To appear.
6. A. Durand, E. Grandjean, and F. Olive. New results on arity vs. number of variables. Research report 20-2004, LIF, Marseille, France, April 2004.
7. J. Flum, M. Frick, and M. Grohe. Query evaluation via tree decompositions. *Journal of the ACM*, 49(6):716–752, 2002.
8. M. Frick and M. Grohe. Deciding first-order properties of locally tree decomposable structures. *Journal of the ACM*, 48:1184–1206, 2001.
9. Y. Gurevich. The decision problem for standard classes. *J. Symb. Logic*, 41(2):pp.460–464, 1976.
10. Y. Gurevich and S. Shelah. On spectra of one unary function. In *18th IEEE Conference in Logic in Computer Science*, IEEE Computer Society, pages 291–300, 2003.
11. L. Libkin. *Elements of finite model theory.* EATCS Series. Springer, 2004.
12. S. Lindell. A normal form for first-order logic over doubly-linked data structures. *Submitted*, 2006.
13. C. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.
14. D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, December 1996.
15. M. Y. Vardi. On the complexity of bounded-variable queries. In *ACM Symposium on Principles of Database Systems*, pages 266–276. ACM Press, 1995.
16. M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Databases*, pages 82–94, 1981.