The complexity of acyclic conjunctive queries revisited

Arnaud Durand, Etienne Grandjean

The complexity of acyclic conjunctive queries revisited

Arnaud Durand [*] Etienne Grandjean [†]

Abstract

In this paper, we consider first-order logic over unary functions and study the complexity of the evaluation problem for conjunctive queries described by such kind of formulas.

A natural notion of query acyclicity for this language is introduced and we study the complexity of a large number of variants or generalizations of acyclic query problems in that context (Boolean or not Boolean, with or without inequalities, comparisons, etc...). Our main results show that all those problems are *fixed-parameter linear* i.e. they can be evaluated in time $f(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|$ where $|Q|$ is the size of the query Q, $|\mathbf{db}|$ the database size, $|Q(\mathbf{db})|$ is the size of the output and f is some function whose value depends on the specific variant of the query problem (in some cases, f is the identity function).

Our results have two kinds of consequences. First, they can be easily translated in the relational (i.e., classical) setting. Previously known bounds for some query problems are improved and new tractable cases are then exhibited. Among others, as an immediate corollary, we improve a result of [PY99] by showing that any (relational) acyclic conjunctive query with inequalities can be evaluated in time $f(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|$.

A second consequence of our method is that it provides a very natural descriptive approach to the complexity of well-known algorithmic problems. A number of examples (such as acyclic subgraph problems, multidimensional matching, etc...) are considered for which new insights of their complexity are given.

[*]Équipe de Logique Mathématique - CNRS UMR 7056. Université Denis Diderot - Paris 7, 2 place jussieu, 75251 Paris cedex 05, France. Email : `durand@logique.jussieu.fr`

[†]GREYC - CNRS UMR 6072. Université de Caen, 14032 Caen, France Email : `grandjean@info.unicaen.fr`

Contents

1 Introduction

The complexity of relational query problems is an important and well-studied field of database theory. In particular, the class of conjunctive queries (equivalent to select-project-join queries) which are among the most simple, the most natural and the most frequent type of queries have received much attention.

A query problem takes as input a database **db** and a query Q and outputs $Q(\mathbf{db})$ the result of the evaluation of Q against **db** (when the query is Boolean, $Q(\mathbf{db})$ is simply *yes* or *no*). There exist mainly two ways to investigate the complexity of such a problem. In the *combined complexity* setting, one expresses the complexity of the problem in terms both of the database size $|\mathbf{db}|$ and of the query size $|Q|$ (and of the output size $|Q(\mathbf{db})|$ if necessary). It is well-known that, in that context, the Boolean conjunctive query problem is NP-complete ([CM77, AHV95]). However, it is natural to consider that the database size is incomparably bigger than the query size and to express the complexity of the problem in terms of the database size only. In that case, the complexity of the conjunctive query problem falls down to P (and even less). However, as discussed by [PY99], that point of view is not completely satisfactory because although the problem becomes polynomial time decidable, the formula size may inherently occur in the exponent of the polynomial. Even for small values of this parameter, this may lead to non tractable cases.

An interesting notion from parameterized complexity ([DF99]) that appears to be very useful in the context of query evaluation (see [PY99]) is fixed parameter tractability. A (query) problem is said to be *fixed-parameter (f.p.) tractable (resp. linear)* if its time complexity is $f(|Q|).P(|\mathbf{db}|, |Q(\mathbf{db})|)$ for some function f and some polynomial P (resp. linear polynomial P). In that case, the formula size influences the complexity of the problem by a multiplicative factor only. Identifying the fragments of relational queries that are f.p. tractable for small polynomials P is then an important but difficult task. Surprisingly, a very broad and well-studied set of queries appears to lie within this class: as shown in [Yan81] (see also [FFG02] for a precise bound), **ACQ**, the *acyclic* conjunctive query problem (we refer to the standard notion of acyclicity in databases; for precise definitions see section 2.1) can be solved in polynomial time $O(|Q|.|\mathbf{db}|.|Q(\mathbf{db})|)$. Besides, it has been proved that evaluating an acyclic conjunctive query is not only polynomial for sequential time but also highly parallelizable (see [GLS01]).

A natural extension $\mathbf{ACQ}^{\neq}$ of **ACQ** allows inequalities between variables, i.e., atoms of the form $x \neq y$. In [PY99], it is shown that this latter class of queries is also f.p. tractable and can be evaluated in time $g(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|. \log^2 |\mathbf{db}|$ where g is an exponential function. Despite of these results, a lot of query problems including the extension of acyclic queries obtained by allowing comparisons of the form $x < y$ are likely f.p. intractable as shown again by [PY99].

In this paper, we revisit the complexity of acyclic conjunctive queries under a different angle. First, a class of so-called *unary functional queries* based on first-order logic over unary functions is introduced. Focusing on the existential fragment of this language, we introduce a very natural graph-based notion of query acyclicity. We then show that various classes of relational conjunctive query problems can be easily interpreted in *linear time* by corresponding (unary) functional conjunctive query problems (see section 3): this is done by switching from the classical language describing relations between elements of some domain D (i.e., the relational setting) to a functional one over the universe of tuples: unary functions basically describe attribute values. In this context, unary functional formulas can be seen as a logical embodiment of the well-known *tuple calculus*. A nice feature of the reduction is that it preserves acyclicity of queries in the two different contexts. The main part of the paper (section 5) is devoted to the analyze of the complexity of the query problem for a wide range of syntactically defined functional formulas. More precisely, whether inequalities ($\neq$) are allowed or not, whether the query is Boolean or not or whether a restricted use of comparisons ($<$) is allowed are considered. In each case, we show that such queries can be evaluated in time $f(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|$ (in time $f(|Q|).|\mathbf{db}|$ for the Boolean case) where the value of function f depends on the precise (functional) query problem under consideration.

Coming back to the relational setting, as immediate corollaries, we obtain a substantial (and optimal) improvement of the bound proved in [PY99] for the $\mathbf{ACQ}^{\neq}$ problem and a new proof of the complexity of the **ACQ** problem. Moreover, we generalize the complexity bound for **ACQ** to a slightly larger class of queries denoted by $\mathbf{ACQ}^{+}$ that allow comparisons ($<$, $\leq$, $\neq$) in a restricted way. This should be compared with the result of [PY99] which shows that an unrestricted use of comparisons inside formulas leads to an intractable query problem. The results of this paper implies that, regardless of the query size, **ACQ**, $\mathbf{ACQ}^{+}$ and $\mathbf{ACQ}^{\neq}$ are inherently of the same *data* complexity.

One can easily describe algorithmic problems by queries written in some language. This allows to reduce the complexity of these problems to the complexity of query evaluations for the language. In section 8, this well-known descriptive approach is used for a number of algorithmic problems (like acyclic subgraph isomorphism, multidimensional matching, etc..). They are considered as well in their decision version as in their function or enumeration (of solutions) version. The variety of languages considered in the paper permits to express easily (i.e. without encoding) a large kind of properties (on graphs, sets, functions, etc...). Our results provides new insight on the complexity of these problems. In all cases, the best known (data) complexity bounds is at least reproved and sometimes improved.

The methods we use to prove the main results of this paper are, as far as we know, original and quite different from those used so far in this context. They are essentially a refinement of the methods introduced in a recent technical report by Frédéric Olive and the present authors (see [DGO04]): that paper essentially deals with hierarchies of definability inside existential second order logic in connection with nondeterministic linear time. As [DGO04] did before, we introduce here a simple combinatorial notion on unary functions called *minimal sample* (see section 4) and develop over this notion some technics of quantifier elimination in formulas that can be performed in linear time. Considering unary functions in the language permits the introduction of simple but powerful new logico-combinatorial methods (based on graphs mainly). Arguments for this are given here through the consequences on the complexity of relational acyclic conjunctive queries. There are possible other applications of the methods and the language; they are discussed in the conclusion.

2 Preliminaries

The reader is expected to be familiar with first-order logic (see e.g. [EF99, Lib04]) but we briefly give some basic definitions on signatures, first-order structures and formulas.

A *signature* (or *vocabulary*) σ is a finite set of relation and function symbols, each of which has a fixed arity which can be zero (0-ary function symbols are constant symbols and 0-ary relation symbols are Boolean variables). The *arity* of σ is the maximal arity of its symbols. A signature whose arity is at most one is said to be *unary*.

A (finite) *structure* $\mathcal{S}$ of vocabulary σ, or σ-structure, consists of a finite domain D of cardinality $n > 1$, and, for any symbol $s \in \sigma$, an interpretation of s over D (often denoted also by s, for simplicity).

We will often deal with *tuples* of objects. We denote them by bold letters: for example, $\mathbf{x} = (x_1, \ldots, x_k)$. If $\mathbf{f}$ is a k-tuple of functions $(f_1, \ldots, f_k)$, then $\mathbf{f}(x)$ stands for $(f_1(x), \ldots, f_k(x))$. Analogously, if $\mathbf{f}$ and $\mathbf{g}$ are two k-tuples of functions, $\mathbf{f}(x) = \mathbf{g}(y)$ stands for the logical statement: $f_1(x) = g_1(y) \wedge \ldots \wedge f_k(x) = g_k(y)$.

Let $\varphi \equiv \varphi(x_1, \ldots, x_k)$ be a first-order formula of signature σ and free variables among $x_1, \ldots, x_k$. Let $\mathrm{var}(\varphi)$ denote the set of variables of φ . Let $\mathcal{L}$ be a class of first-order formulas (also called a query language). The *query* problem associated to $\mathcal{L}$ (and also denoted by $\mathcal{L}$) is defined as follows:

Input: A signature σ, a σ-structure $\mathcal{S}$ of domain D and a first-order σ-formula $\varphi(x_1, \ldots, x_k)$ of $\mathcal{L}$.
Output: The set $\varphi(\mathcal{S}) =_{def} \{(a_1, \ldots, a_k) \in D^k : (\mathcal{S}, a_1, \ldots, a_k) \models \varphi(x_1, \ldots, x_k)\}$.

In the following, query languages $\mathcal{L}$ are always specified by fragments of first-order logic. The query $\mathcal{S} \mapsto \varphi(\mathcal{S})$ is often identified with the formula φ itself.

In this paper, we consider two different kinds of signature σ: either σ contains relation symbols only or it contains relation and function symbols of arity at most one. In the first case, σ is said to be *relational*, a σ-structure will often be denoted by $\mathbf{db}$ and a σ-query by Q. In the second case, σ is said to be *unary functional* or, for short, *functional*, a σ-structure is often denoted by $\mathcal{F}$ and a σ-query by φ.

By making syntactic restrictions on the formula φ, one may define a number of query problems. As we will see, the choice of the kind of signature has some influence also and we will define both relational query problems and the associated functional query problems. In what follows we briefly recall the basics about "classical" conjunctive queries and revisit this notion by introducing a new kind of functional query problem.

2.1 Conjunctive queries

Conjunctive queries can be seen as select-join-project queries (with renaming of variables). Logically speaking, they are equivalent to queries expressed by first-order relational formulas with existential quantification and conjunction, i.e., of the form:

$$Q(y_1, \ldots, y_b) \equiv \exists x_1 \ldots \exists x_a \ \phi(x_1, \ldots, x_a, y_1, \ldots, y_b)$$

where ϕ is a conjunction of atoms over some relational signature σ and variables among $\mathbf{x}, \mathbf{y}$. If Q has no free variable the query is said to be *Boolean*.

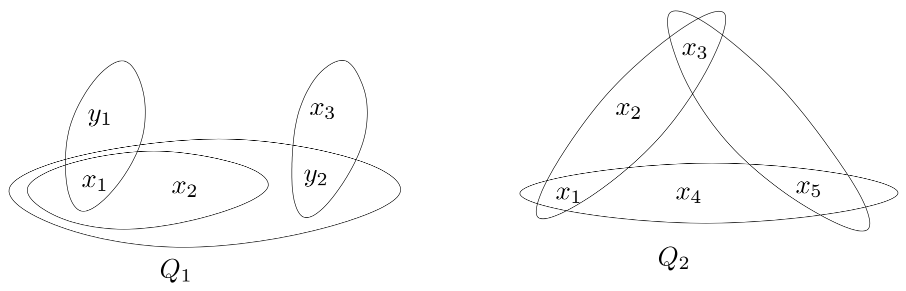

Figure 1: The Hypergraphs of queries Q_1 and Q_2

Example 1 *The two queries below are examples of conjunctive queries.*

$$Q_1(y_1, y_2) \equiv \exists x_1 \exists x_2 \exists x_3 : \; R(x_1, y_1) \wedge S(x_1, y_2, x_2) \wedge T(y_2, x_3) \wedge R(x_1, x_2)$$

$$Q_2 \equiv \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 : \; S(x_1, x_2, x_3) \wedge S(x_1, x_4, x_5) \wedge R(x_3, x_5)$$

Query Q_2 is boolean.

An important and well-studied class of conjunctive queries are the so-called *acyclic* conjunctive queries. To each conjunctive query Q one associates the following hypergraph $\mathcal{H}_Q = (V, E)$: its set of variables is $V = \mathit{var}(Q)$ and its set of hyperedges is $E = \{\mathit{var}(\alpha) : \alpha \; is \; an \; atom \; of \; Q\}$. There exist various notions of acyclicity related to hypergraphs. We have to use the most general one that is defined as follows. A hypergraph is *acyclic* if one can obtain the empty set by repeating the following two rules (known as GYO rules, see [Gra79, YO79]) until no change occurs:

1. Remove hyperedges contained in other hyperedges;
2. Remove vertices that appear in at most one hyperedge.

As usual (see [Fag83]), a query is said to be *acyclic* if its associated hypergraph is acyclic. Denote by **ACQ** the class of acyclic conjunctive queries.

Example 2 *The hypergraphs associated to queries Q_1 and Q_2 of Example 1 are shown in Figure 1. Applying GYO rules shows that Q_1 is acyclic and Q_2 is cyclic.*

Conserving the same notion of acyclicity, one can enlarge this class of queries by allowing inequalities between variables (as defined in [PY99]). This defines the larger class of so-called $\mathbf{ACQ}^{\neq}$ queries.

Example 3 *Query Q_3 below is an example of an $\mathbf{ACQ}^{\neq}$ query.*

$$\begin{aligned} Q_3(y_1, y_2) \equiv \exists x_1 \exists x_2 \exists x_3 : \quad & R(x_1, y_1) \wedge S(x_1, y_2, x_2) \wedge T(y_2, x_3) \wedge R(x_1, x_2) \\ & \wedge y_1 \neq x_3 \wedge x_2 \neq x_1. \end{aligned}$$

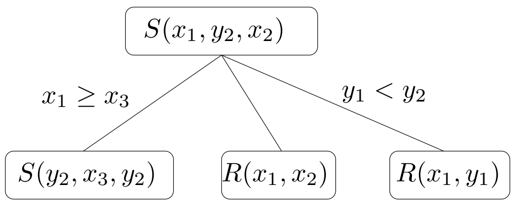

Figure 2: Tree decomposition of query Q_4

Alternatively, it is well-known that a conjunctive query $Q(\mathbf{y})$ is acyclic if and only if it has a *join forest* (called *join tree* in case the forest is connected), that is an acyclic graph $G_Q = (V, E)$ whose set of vertices V is the set of atoms of Q and such that, for each variable x that occurs in Q, the set A_x of relational atoms where x occurs is connected (is a subtree) in G_Q. Similarly, a conjunctive query $Q(\mathbf{y})$ with inequalities is in $\mathbf{ACQ}^{\neq}$ if it has a join forest G_Q. Note that G_Q relies upon the relational atoms but does not take into account the inequalities.

One obtains another natural generalization of acyclic queries by allowing comparison atoms $x < y$. As proved by [PY99] the evaluation problem of such queries is as difficult with respect to parameterized complexity as the clique problem (both are $W[1]$-complete problems) and hence is similarly conjectured to be f.p. intractable. Surprisingly, we will show that for the following class of acyclic queries with (restricted use of) comparisons, denoted by $\mathbf{ACQ}^+$, the evaluation problem is exactly as difficult, with respect to time complexity, as that of $\mathbf{ACQ}$. A conjunctive query Q with comparisons, i.e., atoms of the form $x\theta y$ where x, y are variables and $\theta \in \{\neq, <, \leq, >, \geq\}$ is in $\mathbf{ACQ}^+$ if

1. it has a join forest $G_Q = (V, E)$ (defined as usual),
2. for each comparison $x\theta y$ of Q, either $\{x, y\} \subseteq \mathit{var}(\alpha)$, for some relational atom α of Q, or there is some edge $(\alpha, \beta) \in E$ in G_Q such that $x \in \mathit{var}(\alpha)$ and $y \in \mathit{var}(\beta)$, and
3. for each edge $(\alpha, \beta) \in E$, there is at most one comparison $x\theta y$ in Q such that $x \in \mathit{var}(\alpha)$ and $y \in \mathit{var}(\beta)$.

In other words, a conjunctive query with comparisons is in $\mathbf{ACQ}^+$ if it has a join forest G_Q and if each comparison of Q relates two variables inside the same vertex of G_Q or along an edge of G_Q, with globally at most one comparison per edge. The reason for authorizing only one comparison per edge of the tree will be explain later in Remark 6.

Example 4 *Query Q_4 below is in $\mathbf{ACQ}^+$.*

$$\begin{aligned} Q_4(y_1, y_2) \equiv \exists x_1 \exists x_2 \exists x_3 : \quad & R(x_1, y_1) \wedge S(x_1, y_2, x_2) \wedge S(y_2, x_3, y_2) \wedge R(x_1, x_2) \\ & \wedge y_1 < y_2 \wedge x_1 \geq x_3. \end{aligned}$$

Its join tree is shown in Figure 2.

Finally, as defined in [FFG02], a query $Q(\mathbf{y})$ is said to be *strict* if there exists a relational atom α in Q such that $\mathbf{y} \subseteq var(\alpha)$. We denote by $\mathbf{ACQ}_1$, $\mathbf{ACQ}_1^+$ and $\mathbf{ACQ}_1^{\neq}$ the restrictions of the classes of queries $\mathbf{ACQ}$, $\mathbf{ACQ}^+$, $\mathbf{ACQ}^{\neq}$, respectively, to strict queries.

2.2 Conjunctive functional queries.

In all this part, σ is a unary functional signature. In full generality, a conjunctive functional query is a conjunctive query over some unary functional signature σ. More precisely, it is of the form:

$$\varphi(\mathbf{y}) \equiv \exists x_1 \ldots \exists x_b : \bigwedge_{i=1}^{h} \tau_i(z_i) = \tau_i'(t_i) \wedge \bigwedge_{i=1}^{k} U_i(\tau_i''(v_i))$$

with $z_i, t_i, v_i \in \mathrm{var}(\varphi)$, and τ, τ', τ'' are terms made of compositions of unary function symbols of σ. For example, $\tau(x) = f_1 f_2 \ldots f_k(x)$. Formulas are then interpreted on functional structures with *totally* defined unary functions.

In this paper, formulas over a functional language are viewed as an analog of the well-knowm "tuple calculus". Then, for sake of clarity, we will adopt the following choices in the presentation (these choices do not restrict the applicability of our results to queries of the most general form. See also Remark 1). In what follows, structures are considered as multisorted unary algebras i.e. as a collection of partially defined unary functions. Let $\sigma = \sigma_{rel} \cup \sigma_{fun}$ where σ_{rel} contains unary relation symbols only and σ_{fun} contains unary function symbols. A σ-structure $\mathcal{F}$ will verify :

- Its finite domain D is such that D is the union of all sets $T \in \sigma_{rel}$. Also, for all $T_1, T_2 \in \sigma_{rel}$, $T_1 \cap T_2 = \emptyset$.
- For each function $f \in \sigma_{fun}$, there is a collection $T_{j_1}, \ldots, T_{j_k}$ of sets in σ_{rel}, such that f is defined over $\bigcup_{i \leq k} T_{j_i}$ (and undefined elsewhere) and has value in D.

This definition reflects the fact that each $T \in \sigma_{rel}$ is seen as a set of tuples with each function $f \in \sigma_{fun}$ being a projection function from tuples to the domain D. The number of functions defined over T is equal to the arity of the underlying relation that T represents.

For what concerns σ-formulas two restrictions will be adopted in this paper.

- Quantifications will always be relativized to some universe $X \in \sigma_{rel}$ i.e. formulas are of the form $(\exists x \in T)\varphi$ which is equivalent to $\exists x\, T(x) \wedge \varphi$.
- All atoms are of the form $x \in T$ for $T \in \sigma_{rel}$ or $f(x) = g(y)$ for $f, g \in \sigma_{fun} \cup \{Id\}$ where Id is the identity function. Note that composition of functions is not allowed here.

Example 5 *Formula φ_1 below defines a functional conjunctive query.*

$$\begin{aligned}\varphi_1(x) \equiv \quad & \exists y \in T_1, \exists z \in T_2 : \\ & \qquad f_1(x) = g_1(y) \wedge f_2(x) = h_1(z) \wedge \\ & \qquad f_1(y) = g_2(z) \wedge f_1(z) \neq h_1(y) \wedge \\ & \qquad x \in T_1\end{aligned}$$

As in the relational setting, one can define a notion of acyclic (unary) functional queries. The definition is even more natural and simpler since it relies upon graphs instead of hypergraphs.

Definition 1 *Let φ be a conjunctive functional query. The undirected graph $G_\varphi = (V, E)$ associated to φ is defined by: $V = var(\varphi)$ and for all distinct $x, y \in V$, $(x, y) \in E$ iff φ contains at least one atom of the form $f(x) = g(y)$ for some $f, g \in \sigma \cup \{Id\}$. The query φ is acyclic if its graph G_φ is acyclic.*

We denote by **F-ACQ** the class of acyclic (conjunctive) functional queries. Again, one may authorize the use of negation inside queries. We then denote by **F-ACQ**$^{\neq}$ the class of acyclic functional queries φ whose atoms are of one of the three forms $f(x) = g(y)$, $f(x) \neq g(y)$, or $T(x)$, for $f, g \in \sigma \cup \{Id\}$ and $T \in \sigma$ (recall that the notion of acyclicity relies upon equalities only).

Example 6 *The following query φ_2 belongs to **F-ACQ**$^{\neq}$.*

$$\begin{aligned}\varphi_2(y_1, y_2) \equiv \quad & \exists x_1 \in T_1, \exists x_2 \in T_2, \exists x_3 \in T_2 : \\ & f(x_1) = f(y_1) \wedge g(x_1) = x_2 \wedge g(x_1) = f(y_2) \wedge g(y_2) = x_3 \wedge \\ & \wedge x_3 \neq f(x_1) \wedge g(y_1) \neq f(y_2).\end{aligned}$$

The associated graph of φ_2 is given in Figure 3.

Similarly, let **F-ACQ**$^{+}$ denote the class of acyclic functional queries φ whose atoms are of the form $f(x) = g(y)$ or $f(x)\theta g(y)$ or $U(x)$, for $f, g \in \sigma \cup \{Id\}$, $U \in \sigma$, and $\theta \in \{\neq, <, \leq, >, \geq\}$, whose associated graph G_φ defined at Definition 1) is acyclic and for which the following holds: if $f(x)\theta g(y)$ is a comparison atom of Q for two distinct variables x and y then $(x, y) \in E$ and, conversely, for each edge $(x, y) \in E$, there is at most one comparison $f(x)\theta g(y)$ in Q.

Example 7 *Here is an example of **F-ACQ**$^{+}$ query.*

$$\begin{aligned}\varphi_3(y_1, y_2) \equiv \quad & \exists x_1 \in T_1, \exists x_2 \in T_2, \exists x_3 \in T_2 : \\ & f(x_1) = f(y_1) \wedge g(x_1) = x_2 \wedge g(x_1) = f(y_2) \wedge g(y_2) = x_3 \wedge \\ & \wedge f(x_1) < g(y_2) \wedge f(y_2) \geq g(x_3).\end{aligned}$$

Its associated graph is given in Figure 3.

In analogy with the notion defined above in the relational setting, a functional query is said to be *strict* if it contains *at most one free variable*. We denote by **F-ACQ**$_1$, **F-ACQ**$_1^{\neq}$ and **F-ACQ**$_1^{+}$ the restrictions of the three above defined classes of queries to strict queries.

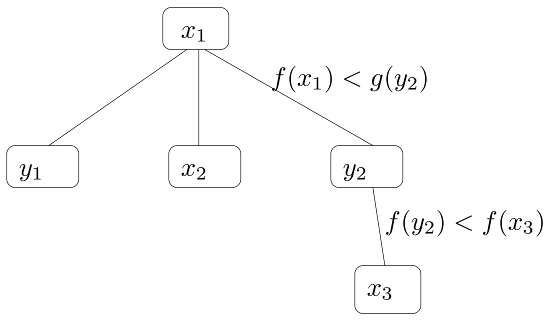

Figure 3: Graph of queries φ_2 (without comparisons) and φ_3

In this paper, we will make extensive use of a class of queries defined, roughly speaking, as the complement of acyclic functional queries. Let **F-FO** be the class of first-order queries defined by universal formulas in conjunctive normal form over some (unary) functional signature σ, i.e., formulas of the form:

$$\varphi(\mathbf{y}) \equiv \forall \mathbf{x} : \bigwedge_{i \leq k} C_i(\mathbf{x}, \mathbf{y})$$

where each C_i is a clause, i.e., a disjunction of literals of the form $(\neg)f(z) = g(t)$ or $U(z)$ for $f, g \in \sigma \cup \{Id\}$ and $U \in \sigma$.

The negation of an **F-FO** formula is clearly a disjunction of conjunctive functional queries $\exists \mathbf{x}\ \neg C_i$. An **F-FO** query φ is said to be *acyclic* if each query $\exists \mathbf{x}\ \neg C_i$ is acyclic. By definition, the acyclicity of an **F-FO** query can be read directly on each clause of the query by looking at inequalities $f(z) \neq g(t)$ of the clause. The class of **F-FO** acyclic queries is denoted by **F-AFO**; its restriction to strict queries is obviously denoted by **F-AFO**$_1$.

Remark 1 *In the formulas we consider, terms made of composition of functions are not autohrized at first sight. However, our results easily applies to this more general kind of formulas: for each term $\tau(x) = f_1 \ldots f_k(x)$, one may add τ as a new unary function sombol in the signature and pre-computes $\tau(x)$, for eaxh $x \in D$, from $f_1, \ldots, f_k$ in linear time. In this way, one can obtain an equivalent query problem but without composed terms. Also, obviously, relativization and the use of partially defined functions do not play an essential role for what concerns the complexity results presented here.*

2.3 Basic notions of complexity

The model of computation used in this paper is the Random Access Machine (RAM) with uniform cost measure (see [AHU74, GS02, GO04, FFG02]). Basically, our inputs are first-order structures and first-order formulas.

Let E be a finite set or relation. We denote by *card*(E) the *cardinality* of E. Let $[n]$ be the set $\{1, \ldots, n\}$. A set of cardinality n is often identified with the set $[n]$.

The *size* $|I|$ of an object I is the number of registers used to store I in the RAM. If E is the set $[n]$, $|E| = card(E) = n$. If $R \subseteq D^k$ is a k-ary relation over domain D, with

$|D| = card(D)$, then $|R| = k.card(R)$: all the tuples $(x_1, \ldots, x_k)$ for which $R(x_1, \ldots, x_k)$ holds must be stored, each in a k-tuple of registers. Similarly, if f is a k-ary function from D^k to D, all values $f(x_1, \ldots, x_k)$ must be stored and $|f| = |D|^k$.

If φ is a first-order formula, $|\varphi|$ is the number of occurrences of variables, relation or function symbols and syntactic symbols: $\exists, \forall, \wedge, \vee, \neg, =, "(", ")", ","$. For example, if $\varphi \equiv \exists x \exists y \; R(x, y) \wedge \neg(x = y)$ then $|\varphi| = 17$.

All the problems we consider in this paper are parameterized problems: each takes as input a list of objects I (e.g., a σ-structure $\mathcal{S}$ and a formula φ) together with a parameter k (e.g., the size of φ) and outputs an object S (e.g. the result of the query $\varphi(\mathcal{S})$).

A problem **P** is computable in time $f(k).T(|I|, |S|)$ for some function $f : \mathbb{N} \to \mathbb{R}^+$ if there exists a RAM that computes **P** in time (i.e., the number of instructions performed) bounded by $f(k).T(|I|, |S|)$ using space i.e., addresses and register contents also bounded by $f(k).T(|I|, |S|)$ [1]. The notation $O_k(T(|I|, |S|))$ is used when one does not want to make precise the value of function f.

Definition 2 *Let T be a polynomial function. A property P is fixed-parameter tractable if it is computable in time $f(k).T(|I|, |S|)$. When T is of the form $T(n, p) = (n \times p)$, P is said to be fixed-parameter linear.*

It is easy to see that one obtains the same complexity measure if instead of the uniform cost the logarithmic cost is adopted, i.e., if the time of each instruction is the number of bits of the objects it manipulates. E.g., if the "uniform" time (and space) complexity is $O_k(|I|, |S|)$ then the corresponding "logarithmic" time complexity is $O_k(|I|.|S|. \log(|I|.|S|))$ which is at most (and in fact less than) $O_k(|I|. \log |I|.|S|. \log |S|) = O_k(|I|_{bit}.|S|_{bit})$ where $|I|_{bit} = \Theta(|I|. \log |I|)$ denotes the number of bits, i.e. the size in the logarithmic cost view, of the input I.

3 Translating relational queries into functional queries

The transformation of (acyclic) queries to be constructed in this section is very similar to the translation of the domain relational calculus into the tuple relational calculus in the classical framework of database theory. Although one needs to examine carefully all the details, the idea is very simple.

We want to transform each input $(\sigma, Q, \mathbf{db})$ of a relational query problem into an input $(\sigma', \varphi_Q, \mathcal{F}_{\mathbf{db}})$ of a (unary) functional query problem so that, among other things, $Q(\mathbf{db})$ is some projection of the relation $\varphi_Q(\mathcal{F}_{\mathbf{db}})$. Let us describe successively the transformation of the structure and the corresponding transformation of the query.

3.1 Transformation of the structure

Let **db** be a relational σ-structure $\mathbf{db} = \langle D; R_1, \ldots, R_q \rangle$ with each R_i of arity m_i. For convenience and simplicity (but w.l.o.g.), assume that there is no isolated element in D,

[1] This last restriction on addresses and register contents forces the RAM to use its memory in a "compact" way with space not greater than time.

i.e., for each $x \in D$, there exists $i \leq q$ and some tuple t in R_i to which x belongs. Let $m = max_{i \leq q} m_i$ be the maximal arity among relations R_is. The associated functional σ'-structure is defined as follows:

$$\mathcal{F}_{\mathbf{db}} = \langle D'; D, T_1, \ldots, T_q, f_1, \ldots, f_m \rangle$$

where the domain D' is the disjoint union of $q+1$ sets $D' = D \cup T_1 \cup \ldots \cup T_q$ where D is the domain of **db** and each T_i, $1 \leq i \leq q$, is a set of elements identified to the tuples of R_i ($card(T_i) = card(R_i)$); each of D, T_1, ..., T_q is a unary relation of $\mathcal{F}_{\mathbf{db}}$; each f_j, $1 \leq j \leq m$, is a unary function.

Functions f_j are defined as follows. For each R_i of arity $m_i \leq m$ ($1 \leq i \leq q$) and for each $t \in T_i$ that represents the tuple $(e_1, \ldots, e_{m_i})$ of R_i, set $f_1(t) = e_1, \ldots, f_{m_i}(t) = e_{m_i}$. Intuitively, each f_j is the j^{th} projection for each tuple, it is obviously defined on sets T_i that represents relations R_i with $m_i \geq j$, else it is undefined. Clearly, the functional structure $\mathcal{F}_{db}$ encodes the whole database structure **db**. We first have to prove the following result.

Proposition 1 *The transformation* $\mathbf{db} \mapsto \mathcal{F}_{\mathbf{db}}$ *is computable in linear time* $O(|\mathbf{db}|)$.

Proof. Since the transformation is immediate, we only have to prove that $|\mathcal{F}_{\mathbf{db}}| = O(|\mathbf{db}|)$. It is essential to notice that each f_j is defined and described on some subset of $\bigcup_{i \leq q} T_i$ so that $|f_j| = O(\sum_{i \leq q, j \leq m_i} |T_i|) = O(\sum_{i \leq q, j \leq m_i} card(R_i))$ and hence $\sum_{j \leq m} |f_j| = O(\sum_{i \leq q} m_i.card(R_i)) = O(\sum_{i \leq q} |R_i|) = O(|\mathbf{db}|)$. Finally, $\mathcal{F}_{\mathbf{db}} = |D'| + |D| + \sum_{i \leq q} |T_i| + \sum_{j \leq m} |f_j| = O(|\mathbf{db}|)$. □

3.2 Transformation of the query

The transformation is essentially the same for all the variants ($\mathbf{ACQ}, \mathbf{ACQ}^{\neq}$, etc) of acyclic queries. We present it here for $\mathbf{ACQ}_1^{\neq}$. Let $Q(y_1, \ldots, y_b)$ denote an $\mathbf{ACQ}_1^{\neq}$ query, i.e., a strict acyclic query with inequalities of the form:

$$Q(y_1, \ldots, y_b) \equiv \exists x_1 \ldots \exists x_a \Psi(x_1, \ldots, x_a, y_1, \ldots, y_b)$$

with $\Psi(\mathbf{x}, \mathbf{y}) \equiv \bigwedge_{1 \leq i \leq k} A_i \wedge I$ where the A_i's are relational atoms and I is a conjunction of variable inequalities $v \neq v'$ for $v, v' \in \{\mathbf{x}, \mathbf{y}\}$.

By definition of the strict acyclicity, Q has a join forest $F = (V, E)$ whose set of vertices is $V = \{A_1, \ldots, A_k\}$ so that $\mathbf{y} \subseteq \text{var}(A_1)$. We want to construct a conjunctive functional σ'-formula φ_Q whose graph G_{φ_Q} is exactly the acyclic graph F. Roughly, the idea is to replace the k atoms $A_1, \ldots, A_k$ by k variables $t_1, \ldots, t_k$ that represent the corresponding tuples. For a relational atom A_u, $1 \leq u \leq k$, let $var_i(A_u)$ denote the i^{th} variable of A_u: e.g., if A_u is the atom $S(y_2, x_3, y_2)$ then $var_1(A_u) = var_3(A_u) = y_2$. As defined before, each function f_j, $j \leq m$, of the functional structure $\mathcal{F}_{\mathbf{db}}$ gives for each tuple t (of a relation of **db**) its j^{th} field $f_j(t)$. E.g., the above equality $var_1(A_u) = var_3(A_u)$ for $A_u = S(y_2, x_3, y_2)$ is expressed by the formula $f_1(t_u) = f_3(t_u)$. The following functional formula essentially mimics the description of formula Q and of its forest $F = (V, E)$:

$$\varphi_Q(\mathbf{t}) \equiv \Psi_{rel}(\mathbf{t}) \wedge \Psi_V(\mathbf{t}) \wedge \Psi_E(\mathbf{t}) \wedge \Psi_I(\mathbf{t}).$$

Each conjunct of φ_Q is described precisely as follows.

- $\Psi_{rel}(\mathbf{t})$ is $\bigwedge_{u\le k} T_{v_u}(t_u)$ if the atom A_u is of the form $R_{v_u}(\ldots)$.
- $\Psi_V(\mathbf{t})$ is $\bigwedge_{u\le k} \Psi_u$ where Ψ_u is nonempty if A_u has at least one repeated variable and contains for each (repeated) variable that occurs at successive indices $j_1,\ldots,j_r$ of A_u the conjunction $\bigwedge_{i<r} f_{j_i}(t_u) = f_{j_{i+1}}(t_u)$.
- $\Psi_E(\mathbf{t})$ is $\bigwedge_{(A_u,A_v)\in E} \Psi_{u,v}$ where $\Psi_{u,v}$ contains, for each variable w that occurs both in A_u and A_v with $(A_u,A_v)\in E$, one equality of the form $f_i(t_u)=f_j(t_v)$ for two arbitrarily chosen indices i,j such that $var_i(A_u)=w=var_j(A_v)$.
- $\Psi_I(\mathbf{t})$ is constructed as follows. For each inequality $w\neq w'$ of I, choose (arbitrarily, again) two atoms A_u and A_v so that w (resp. w') occurs in A_u (resp. A_v) at index i (resp. j). Replace $w\neq w'$ by the inequality $f_i(t_u)\neq f_j(t_v)$. Let Ψ_I be the conjunction of all those inequalities.

Due to formula $\Psi_{rel}(\mathbf{t})$, each quantified variable is relativized to some domain T_i.

Example 8 *The following query:*

$$Q(y_1,y_2)\equiv \exists x_1\exists x_2\exists x_3 : R_1(x_1,y_1,y_2)\wedge R_2(x_2,x_1,x_2)\wedge R_1(x_2,x_2,x_3)\wedge y_1\neq x_2,$$

is translated into the formula $\varphi_Q(\mathbf{t})$, with $\mathbf{t}=(t_1,t_2,t_3)$, that is the conjunction of the following formulas:

$$\begin{aligned}\Psi_{rel}(\mathbf{t})\equiv&\quad T_1(t_1)\wedge T_2(t_2)\wedge T_1(t_3)\\ \Psi_V(\mathbf{t})\equiv&\quad f_1(t_2)=f_3(t_2)\wedge f_1(t_3)=f_2(t_3)\\ \Psi_E(\mathbf{t})\equiv&\quad f_1(t_1)=f_2(t_2)\wedge f_1(t_2)=f_1(t_3)\\ \Psi_I(\mathbf{t})\equiv&\quad f_2(t_1)\neq f_1(t_2)\end{aligned}$$

Finally, it is easy to check that the following equality holds:

$$Q(\boldsymbol{db})=\{(f_2(t_1),f_3(t_1)) : t_1\in D' \text{ and } (\mathcal{F}_{\boldsymbol{db}},t_1)\models \exists t_2\exists t_3\varphi_Q(\mathbf{t})\}.$$

In other words, $Q(\boldsymbol{db})$ is the result of the projection $\mathbf{t}\mapsto(f_2(t_1),f_3(t_1))$ applied to the relation $\varphi_Q(\mathcal{F}_{\boldsymbol{db}})$. Obviously, formula $\exists t_2\exists t_3\varphi_Q(\mathbf{t})$ is equivalent to the relativized formula:

$$\exists t_1\in T_1,\exists t_2\in T_2 : T_1(t_3)\wedge\Psi_V(\mathbf{t})\wedge\Psi_E(\mathbf{t})\wedge\Psi_I(\mathbf{t}).$$

More generally, the transformation process described before yields the following properties.

Lemma 2 *Let $Q(y_1,\ldots,y_b)$ be a query in $\boldsymbol{ACQ}_1^{\neq}$ (resp. $\boldsymbol{ACQ}^{\neq}$, $\boldsymbol{ACQ}_1$, $\boldsymbol{ACQ}$, $\boldsymbol{ACQ}_1^{+}$, $\boldsymbol{ACQ}^{+}$). The following properties hold:*

1. *$\varphi_Q\in$ **F-ACQ**$_1^{\neq}$ (resp. **F-ACQ**$^{\neq}$, **F-ACQ**$_1$, **F-ACQ**, **F-ACQ**$_1^{+}$, **F-ACQ**$^{+}$).*

2. *For each relational σ-structure **db**, the result $Q(\boldsymbol{db})$ (of query Q over **db**) is obtained by some "projection" of the relation $\varphi_Q(\mathcal{F}_{\boldsymbol{db}})$. More precisely, there are two lists of indices $i_1, \ldots, i_b$ and $j_1, \ldots, j_b$ such that* [2]

$$\begin{aligned} Q(\boldsymbol{db}) = \quad & \{(f_{i_1}(t_{j_1}), \ldots, f_{i_b}(t_{j_b})) : \\ & \textit{there exist } t_1, \ldots, t_k \in D' \textit{ such that } (\mathcal{F}_{\boldsymbol{db}}, \mathbf{t}) \models \varphi_Q(\mathbf{t})\}. \end{aligned}$$

   where $y_h = var_{i_h}(A_{j_h})$ for $h = 1, \ldots, b$.

3. $|\varphi_Q| = O(|Q|)$.

Proof. For simplicity of notation, let us still assume that Q belongs to $\mathbf{ACQ}_1^{\neq}$.

1. By construction, the graph G_{φ_Q} is (up to isomorphism) the join forest F (associated to Q); this corresponds to the conjunct Ψ_E. See also Remark 2

2. By definition of the join forest F, the set of atoms where any fixed variable of Q occurs is connected in F. This implies that the conjunct $\Psi_V \wedge \Psi_E$ exactly expresses which variables the relational atoms $A_1, \ldots, A_k$ of Q share. Moreover, Ψ_I correctly expresses the conjunction I of inequalities of Q. This proves that for each relational σ-structure $\mathbf{db} = \langle D; R_1, \ldots, R_q \rangle$ where $\mathcal{F}_{\mathbf{db}} = \langle D'; D, T_1, \ldots, T_q, f_1, \ldots, f_m \rangle$ and for all $\mathbf{y} \in D^b$, it holds:

$$\begin{gathered} (\mathbf{db}, \mathbf{y}) \models Q(\mathbf{y}) \textit{ iff} \\ \text{there exists } t_1, \ldots, t_k \in D' \text{ such that} \\ (\mathcal{F}_{\mathbf{db}}, \mathbf{t}) \models \varphi_Q(\mathbf{t}) \text{ and } f_{i_h}(t_1) = y_h \text{ for each } h = 1, \ldots, b. \end{gathered}$$

3. Let $NbOcc$ denote the number of occurrences of variables in Q. It is easy to see that:

$$|\Psi_V| + |\Psi_E| = O(NbOcc).$$

   Clearly, we also have $|\Psi_I| = O(|I|)$ and $|\Psi_{rel}| = O(k)$. That implies $|\varphi_Q| = O(|Q|)$.

□

Remark 2 *Having a join forest F for Q is not necessary to construct the acyclic formula Ψ_E in φ_Q. There is an alternative way to obtain an equivalent Ψ_E using the GYO rules ([Gra79, YO79]). Let $\mathcal{H}_Q$ be the hypergraph associated to query Q. For each application of rule 2 ("remove vertices that appear in at most one hyperedge") nothing as to be done. However, each time rule 1 ("remove hyperedges contained in other hyperedge") is applied to some atom A_u and A_v, one proceed as for the original construction of Ψ_E: for each variable w that occurs in A_u and A_v, one equality of the form $f_i(t_u) = f_j(t_v)$ for two arbitrarily chosen indices i, j such that $var_i(A_u) = w = var_j(A_v)$ is constructed.*

Applying these rules till $\mathcal{H}$ is empty will result in a new acyclic formula Ψ_E.

[2]In case $Q(\mathbf{y})$ is a strict query with $\mathbf{y} \subseteq var(A_1)$ then $j_1 = \ldots = j_b = 1$.

4 Samples of unary functions

In this section, some simple combinatorial notions about unary functions are defined. They may be seen as some kind of set/table covering problem. They will be essential for proving the main results of this paper.

Definition 3 *Let E, F be two finite sets and $\mathbf{g} = (g_1, \ldots, g_k)$ be a tuple of unary functions from E to F. Let $P \subseteq [k]$ and $(c_i)_{i \in P}$ be a family of elements of F. $(P, (c_i)_{i \in P})$ is said to be a* sample *of* $\mathbf{g}$ *(indexed by P) over E if*

$$E = \bigcup_{i \in P} g_i^{-1}(c_i).$$

where $g_i^{-1}(c_i)$ is the set of preimages of c_i by function g_i. A sample is said to be minimal *if, moreover, for all $j \in P$:*

$$E \neq \bigcup_{i \in P \setminus \{j\}} g_i^{-1}(c_i).$$

Finally, if $(P, (c_i)_{i \in P})$ is a sample (resp. minimal sample) of $\mathbf{g}$ *over E, the family of sets $(g_i^{-1}(c_i))_{i \in P}$ is called a* covering *(resp.* minimal covering*) of E by* $\mathbf{g}$.

Samples will often simply be denoted $(c_1, \ldots, c_k)$ with $c_i = '-'$ when $i \notin P$.

Example 9 *Let $\mathbf{g} = (g_1, g_2, g_3)$ be the following tuple of unary functions over some domain/table T with tuples a, b, c, d, e.*

	g_1	g_2	g_3
a	1	2	4
b	1	5	1
c	3	2	4
d	3	5	3
e	5	2	4

It is easily seen that the tuples $(1, 2, 3), (1, 5, 4), (3, 2, 1)$ and $(-, 5, 4)$ are the samples of $\mathbf{g}$ *over T. Among them, $(1, 2, 3), (3, 2, 1)$ and $(-, 5, 4)$ are minimal.*

Remark 3 *Let $P \subseteq [k]$ and $(c_i)_{i \in P}$ be a sample of* $\mathbf{g}$ *over E. Then, there exists $P' \subseteq P$ such that $(c_i)_{i \in P'}$ is a minimal sample of* $\mathbf{g}$ *over E. Informally, it is obtained by repeating the following steps as long as possible:*

- pick a j from P such that $E = \bigcup_{i \in P \setminus \{j\}} g_i^{-1}(c_i)$

- set $P \leftarrow P \setminus \{j\}$.

Note that the only minimal sample of $\mathbf{g}$ *over the empty set is $(-, -, \ldots, -)$*

In the rest of this section, problems about minimal samples are defined and their complexities are studied. Those problems will play a key role in the paper.

MIN-SAMPLES
Input: two finite sets E and F and a k-tuple of unary functions $\mathbf{g} = (g_1, \ldots, g_k)$ from E to F.
Parameter: integer k.
Output: the set of minimal samples of $\mathbf{g}$ over E.

Lemma 3 *Let E, F, $\mathbf{g}$ be an input of* MIN-SAMPLES.

1. *There are at most $k!$ minimal samples of $\mathbf{g}$ over E.*
2. *Problem* MIN-SAMPLES *can be solved in time $O_k(|E|)$.*

Proof. Let us identify E with the set $\{1, \ldots, n\}$. We describe the construction of a tree T of depth n with at most $k!$ leaves and hence at most $k!|E| = O_k(n)$ nodes. The leaves represent all the minimal samples of $\mathbf{g}$ over E. Level i of the tree corresponds to element i of E. Each node x of level i is labelled by a subset $P^x \subseteq [k]$ and by a sample $(c_j^x)_{j \in P^x}$ of $\mathbf{g}$ over $\{1, \ldots, i\}$. The root r of the tree is labelled by $P^r = \emptyset$.

Let $i = 1$. There are at most k possibilities to cover element 1 with $g_h(1) = c_h$ $(h = 1, \ldots, k)$. Then, the root of T has k children x_h each labelled by $(P^{x_h} = \{h\}, c_h)$.

At each level i $(i = 2, \ldots, n)$, the same strategy is used. Let x be a node of level $i-1$ labelled by $(P^x, (c_j^x)_{j \in P^x})$. The set of children y of x labelled by $(P^y, (c_j^y)_{j \in P^y})$ will correspond to all the possibilities to extend the covering of $\{1, \ldots, i-1\}$ by $(P^x, (c_j^x)_{j \in P^x})$ in a minimal way in order to cover node i (if i is not already covered).

Testing whether i is already covered, i.e., if $i \in \bigcup_{j \in P^x} g_j^{-1}(c_j^x)$ can be done in constant time $O_k(1)$: it suffices to test the disjunction $\bigvee_{j \in P^x} g_j(i) = c_j^x$. Two cases may occur:

- Either $i \in \bigcup_{j \in P^x} g_j^{-1}(c_j^x)$. In this case, node x has a unique child node y of level i with $P^y = P^x$ and $\forall j \in P^x$, $c_j^y = c_j^x$.
- Or $i \notin \bigcup_{j \in P^x} g_j^{-1}(c_j^x)$. Two subcases may hold.
  - Either $P^x = [k]$. Then, it is not possible to cover element i, the construction fails and stops here for that branch.
  - Or $P^x \subsetneq [k]$. Then, for each $h \in [k] \backslash P^x$, one constructs a child node y for x such that: $P^y = P^x \cup \{h\}$, $c_j^y = c_j^x$ for $j \in P^x$ and $c_h^y = g_h(i)$. Node y and its label can be constructed in constant time.

That process ends after $O_k(|E|)$ steps with a tree of size $O_k(|E|)$ whose leaves represent the (up to) $k!$ minimal samples of $\mathbf{g}$ over E: $(c_1, \ldots, c_k)$ with $c_i = \; '-'$ when $i \notin P$. [3]. $\square$

[3] To be completely rigorous, we should mention that some of the (up to) $k!$ samples that our algorithm constructs may be not minimal. The essential property is that, by construction, each minimal sample (of $\mathbf{g}$ over E) is included in (at least) one of the constructed samples. The algorithm above should be completed by a variant of the algorithm of Remark 3 that extracts from a sample all the minimal samples that it contains. Note that the whole additional time required is $O_k(1)$ and that some minimal samples may be repeated.

We will also need a more elaborate problem about samples. Let l be an integer and v be a function from E to E^l. The image set of v is denoted by $v(E)$. It is clear that the collection of sets $v^{-1}(\mathbf{a})$ for $\mathbf{a} = (a_1, \ldots, a_l) \in v(E) \subseteq E^l$ forms a partition of E. Let us define the following problem.

MIN-SAMPLES-PARTITION

Input: two finite sets E and F, two integers k and l, a k-tuple of unary functions $\mathbf{g} = (g_1, \ldots, g_k)$ from E to F and a function v from E to E^l.

Parameter: integers k and l.

Output: for each $\mathbf{a} \in v(E) \subseteq E^l$, the set $M(\mathbf{a})$ of minimal samples of $\mathbf{g}$ over $v^{-1}(\mathbf{a})$.

Lemma 4 *Problem* MIN-SAMPLES-PARTITION *can be solved in time $O_{k,l}(|E|)$.*

Proof. The algorithm is the following.

1. Compute the set $S = \{(v(x), x) : x \in E\}$ in time $O_l(|E|)$.
2. Sort S by values of $v(x)$: this computes the partition $(v^{-1}(\mathbf{a}))_{\mathbf{a} \in v(E)}$ of E in time $O_l(|E|)$.
3. For each $\mathbf{a} \in v(E)$, compute the set $M(\mathbf{a})$ of minimal samples of $\mathbf{g}$ over $v^{-1}(\mathbf{a})$ in time $O_k(|v^{-1}(\mathbf{a})|)$ (by Lemma 3). The total time required for this last step is $O_k(\sum_{\mathbf{a} \in v(E)} |v^{-1}(\mathbf{a})|) = O_k(|E|)$.

□

Remark 4 *The "sampling" problems and their algorithms that are involved in Lemmas 3 and 4 can be seen as generalizations of the well-known k-VERTEX COVER problem in graphs and its algorithm of parameterized linear complexity $O_k(|G|)$ (see [DF99]).*

Let $G = \langle V; E \rangle$ be a graph and $\mathcal{F}_G = \langle D; f_1, f_2 \rangle$ be its functional representation : $D = V \cup E$ and for each $e \in E$, $f_1(e)$ and $f_2(e)$ describe the endpoints of e. Then $C = \{c_1, \ldots, c_k\} \subseteq V$ is a k-VERTEX COVER of G if:

$$\forall x \in E, f_1(x) = c_1 \vee \ldots \vee f_1(x) = c_k \vee f_2(x) = c_1 \vee \ldots \vee f_2(x) = c_k.$$

In other words, $(c_1, \ldots, c_k, c_1, \ldots, c_k)$ is a $(f_1, \ldots, f_1, f_2, \ldots, f_2)$-sampling of V.

5 The complexity of functional acyclic queries

Roughly, the main technic of this paper shows among other things that it is possible to eliminate quantified variables in an acyclic conjunctive query (by transforming both the query and the structure)without overhead in the query evaluation process, i.e., so that evaluating the query so simplified is just as hard as evaluating the original query.

For the sake of clarity, the main result will first be stated in the context of **F-AFO** queries. Let us explain the method on a very simple example. Let φ be the following Boolean **F-AFO** query (without negation and only two variables):

$$\varphi \equiv \forall x \forall y : f_1(x) = g_1(y) \vee \ldots \vee f_k(x) = g_k(y)$$

A first naive approach for evaluating φ against a given unary functional structure $\mathcal{F} = \langle D; \mathbf{f}, \mathbf{g} \rangle$ consists in testing the truth value of the matrix for any possible value of (x, y): that requires a time $O_k(|D|^2)$. Alternatively, φ can be interpreted as follows: for each value of x, the family of sets $g_i^{-1}(f_i(x))$, for $i \in \{1, \ldots, k\}$, is a covering of D. In other words, for each x, there exists a sample $(P, (c_i)_{i \in P})$ of $\mathbf{g}$ over D (with initially $P = [k]$) that "agrees" with values of $\mathbf{f}(x)$, i.e., such that:

$$\bigwedge_{i \in P} f_i(x) = c_i \text{ holds.} \tag{1}$$

Such a sample can be chosen among minimal ones (recall Remark 3). Then, evaluating φ against $\mathcal{F}$ can be done as follows. First, the set of the (up to) $k!$ minimal samples of $\mathbf{g}$ over D is computed. Then, for each x, it is looked for one of these minimal samples that satisfy Property 1. Because of Lemma 3, the whole process requires $O_k(|D|)$ steps.

With some more work, this basic idea can be extended to the general case of (non necessarily Boolean) acyclic queries where both equalities and inequalities are allowed. This is achieved through the main result that follows.

Theorem 5 *The **F-AFO**$_1$ query problem can be solved in time $f(|\varphi|).|D|$ where D is the domain of the input structure $\mathcal{F}$, φ is the input formula and f is a fixed function from $\mathbb{N}$ to $\mathbb{R}^+$.*

Proof. Let a unary functional structure $\mathcal{F}$ of domain D and a formula $\varphi(x) \equiv \forall \mathbf{y} \phi(x, \mathbf{y})$ with $\mathbf{y} = (y_1, \ldots, y_d)$ be the inputs of an **F-AFO**$_1$ query problem. Since clauses may be reduced independently, it can be supposed that $\varphi(x)$ contains only one clause ϕ. The proof is done by induction on the number of variables of φ.

Let G_φ denote the acyclic graph associated to φ whose set of vertices is *var*(φ). Recall that G_φ only takes into account the *inequalities* of the clause ϕ. Without loss of generality, assume that G_φ is connected, i.e., is a tree T and choose x as the root of T. Order the nodes of T, i.e., the variables of φ, by increasing levels from the root to the leaves, as $y_0 = x, y_1, \ldots, y_d$. Note that the restriction of T to the subset of variables $\{y_0, y_1, \ldots, y_i\}$ for $i \leq d$ is a subtree T_i where y_i is a leaf. The variables of φ except $y_0 = x$ will be eliminated one by one according to this ordering. Let y_{i_0}, $i_0 \leq d-1$, be the parent of leaf y_d in T. W.l.o.g., assume that φ is of the form [4]:

$$\varphi(y_0) \equiv \forall y_1 \ldots \forall y_d : \bigvee_{j \leq l} v_j(y_d) \neq u_j(y_{i_0}) \vee \psi(\mathbf{y}) \vee \bigvee_{j \leq k} g_j(y_d) = f_j(y_{p_j})$$

where $\mathbf{y}$ is now the d-tuple of variables $(y_0, y_1, \ldots, y_{d-1})$, the u_j, v_j, for $1 \leq j \leq l$, and f_j, g_j, for $1 \leq j \leq k$ are unary function symbols, $\psi(\mathbf{y})$ is an acyclic clause over

[4]In case φ contains one-variable atoms of the form $(\neg)u(y_d) = v(y_d)$ or $(\neg)U(y_d)$, we can easily replace them by two-variable positive atoms by expanding the signature and the structure by new unary functions computable in linear time

$\mathbf{y}$ of associated graph $G_\psi = T_{d-1}$, and for each $j \leq k$, $0 \leq p_j \leq d-1$. Replacing our disjunction of negated atoms by an implication, one obtains:

$$\varphi(y_0) \equiv \forall y_1 \dots \forall y_{d-1} \forall y_d : \mathbf{v}(y_d) = \mathbf{u}(y_{i_0}) \to (\psi(\mathbf{y}) \vee \bigvee_{j \leq k} g_j(y_d) = f_j(y_{p_j}))$$

where $\mathbf{v}(y_d) = \mathbf{u}(y_{i_0})$ stands for $\bigwedge_{j \leq l} v_j(y_d) = u_j(y_{i_0})$. Formula φ can be equivalently written as:

$$\varphi(y_0) \equiv \forall y_1 \dots \forall y_{d-1} : \psi(\mathbf{y}) \vee [\forall y_d \in \mathbf{v}^{-1}(\mathbf{u}(y_{i_0})) \bigvee_{j \leq k} g_j(y_d) = f_j(y_{p_j})].$$

The second disjunct states that $(f_j(y_{p_j}))_{j \leq k}$ is a sample of $\mathbf{g}$ over $\mathbf{v}^{-1}(\mathbf{u}(y_{i_0}))$ and hence contains such a minimal sample.

The family $M = \{(b, M(\mathbf{u}(b))) : b \in D\}$ of the sets of minimal samples $M(\mathbf{u}(b)) = \{(c_1^h(b), \dots, c_k^h(b)) : 1 \leq h \leq k!\}$ of $\mathbf{g}$ over $\mathbf{v}^{-1}(\mathbf{u}(b))$ is computed by Algorithm A below (since the number of minimal samples is only bounded by $k!$, there may be repetitions of identical samples).

Algorithm A:

1. Compute the family $A = \{(\mathbf{a}, M(\mathbf{a})) : \mathbf{a} \in \mathbf{v}(D)\}$ where $M(\mathbf{a})$ is the set of minimal samples of $\mathbf{g}$ over $\mathbf{v}^{-1}(\mathbf{a})$: this amounts to solve the problem MIN-SAMPLES-PARTITION in time $O_{k,l}(|D|)$ by Lemma 4 for $E = F = D$ and $v = \mathbf{v}$.

2. Sort A in lexicographic order according to $\mathbf{a}$.

3. Compute and lexicographically sort the set $B = \{(\mathbf{u}(b), b) : b \in D\}$ in time $O_l(|D|)$.

4. Merge the sorted lists A and B (in time $O_{k,l}(|D|)$) to compute the set
$$C = \{(\mathbf{u}(b), M(\mathbf{u}(b)), b) : b \in D\}$$

5. Return the family of sets (of minimal samples) $M = \{(b, M(\mathbf{u}(b))) : b \in D\}$

Hence, the time complexity of Algorithm A is $O_{k,l}(|D|)$. In the set M, we are interested, of course, by the elements b for which $M(\mathbf{u}(b))$ is not empty. Let:

$$K = \{b : M(\mathbf{u}(b)) \neq \emptyset\}.$$

We now eliminate variable y_d by expanding the signature of the query and the structure (a classical method in quantifier elimination). New unary relations K, S_j^h and functions c_j^h (for $h \leq k!$ and $j \leq k$) are introduced.

Functions c_j^h are those that appear in the description of the minimal samples. Predicates S_j^h are defined from P^h as follows: for all j, h and $y \in D$,

$$S_j^h(y) \leftrightarrow y \in K \text{ and } c_j^h \neq \text{'}-\text{'}.$$

Let $\mathcal{F}'$ be the expansion of structure $\mathcal{F}$ defined as $\mathcal{F}' = (\mathcal{F}, (S_j^h, c_j^h)_{h\leq k!, j\leq k})$. Let now φ' be the following formula having d variables $y_0, \dots, y_{d-1}$ (recall $i_0 < d$):

$$\varphi'(y_0) \equiv \forall y_1 \dots \forall y_{d-1} : \psi(\mathbf{y}) \vee [K(y_{i_0}) \wedge \bigvee_{h\leq k!} \bigwedge_{j\leq k} (S_j^h(y_{i_0}) \to f_j(y_{p_j}) = c_j^h(y_{i_0}))]$$

The last part of formula φ' simply asserts that if j belongs to the index set of the h-th minimal sample of $\mathbf{g}$ over $\mathbf{v}^{-1}(\mathbf{u}(y_{i_0}))$ then, $f_j(y_{p_j})$ must be equal to the j-th value of the h-th minimal sample. That means that $(f_j(y_{p_j}))_{j\leq k}$ contains a minimal sample of $\mathbf{g}$ over $\mathbf{v}^{-1}(\mathbf{u}(y_{i_0}))$. It is then clear that, for each possible value a of $x = y_0$, it holds $(\mathcal{F}, a) \models \varphi(x) \leftrightarrow (\mathcal{F}', a) \models \varphi'(x)$, that means $\varphi(\mathcal{F}) = \varphi'(\mathcal{F}')$. Notice that the last part of formula φ' does not really introduce negative atoms: it can be rephrased as $\bigvee_{h\leq k!} \bigwedge_{j\leq k} (S_j^h(y_{i_0}) = 0 \vee f_j(y_{p_j}) = c_j^h(y_{i_0}))$ where S_j^h is now regarded as a unary function from D to $\{0, 1\}$. From the previous paragraphs, the two following facts also clearly hold.

Fact 1 *The expansion $\mathcal{F}'$ of structure $\mathcal{F}$ can be computed in time $O_{k,l}(|D|)$.*

Fact 2 *Formula $\varphi'(x)$ can be easily transformed into a conjunction of acyclic clauses each having d variables and associated tree T_{d-1}.*

By iterating this process d times, i.e., eliminating successively variables $y_d, y_{d-1}, \dots, y_1$, one obtains in time $O_\varphi(|D|)$ an expansion $\mathcal{F}'$ of $\mathcal{F}$ and a quantifier-free formula $\varphi'(x)$ with only one variable $x = y_0$. It is clear that the final query $\varphi'(\mathcal{F}') = \{a \in D : (\mathcal{F}', a) \models \varphi'(x)\} = \varphi(\mathcal{F})$ can be computed in linear time $O(|\varphi'|.|D|)$. □

Remark 5 (On the constant value $f(|\varphi|)$) *In the worst case, the value of $f(|\varphi|)$ may be huge: each elimination step may introduce a number of new atoms bounded by $k!$ (and requires to put the new formula in conjunctive normal form for the next step).*

*A very interesting particular case concerns **F-AFO**$_1$-queries without positive atoms (closely related to the **ACQ**$_1$ problem). In that case, formula $\varphi(x)$ is of the following form, for some $i_0 \leq d$:*

$$\begin{aligned}\varphi(x) &\equiv \forall y_1 \dots \forall y_{d-1} \forall y_d : \textstyle\bigvee_{j\leq l} v_j(y_d) \neq u_j(y_{i_0}) \vee \psi(\mathbf{y}) \\ &\equiv \forall y_1 \dots \forall y_{d-1} : \psi(\mathbf{y}) \vee \neg \exists y_d (\mathbf{v}(y_d) = \mathbf{u}(y_{i_0}))\end{aligned}$$

with $\mathbf{y} = (y_0, \dots, y_{d-1})$. It is easy to see that one can compute, in time $O(l.|D|)$, the set D_0 of elements $y \in D$ such that $\mathbf{u}(y) \in \mathbf{v}(D)$ (i.e., such that there exists y_d with $\mathbf{v}(y_d) = \mathbf{u}(y)$). By enlarging the signature, the formula φ can be transformed into an equivalent formula without variable y_d (also denoted by φ for convenience):

$$\varphi(x) \equiv \forall y_1 \dots \forall y_{d-1} : \psi(\mathbf{y}) \vee \neg D_0(y_{i_0})$$

*Note that, although a new atom $D_0(y_{i_0})$ has been introduced, the sum of the number of quantifiers plus the number of literals of φ has been decreased by l. Summing up the costs of all the steps, it yields that **F-AFO**$_1$-queries without positive atoms can be evaluated in time $O(|\varphi|.|D|)$.*

We are now able to state the consequences of our results, first in the context of acyclic conjunctive functional queries.

Theorem 6 *The query problem **F-ACQ**$_1^{\neq}$ (resp. **F-ACQ**$_1$) can be solved in time $f(|\varphi|).|D|$ (resp. $O(|\varphi|.|D|)$).*

Proof. Let $\mathcal{F}$ and $\varphi(x)$ be inputs of the **F-ACQ**$_1^{\neq}$ (resp. **F-ACQ**$_1$) problem. By definition, $\neg\varphi(x)$ defines the **F-AFO**$_1$ query (resp. **F-AFO**$_1$ query without positive atoms) whose output is $D \setminus \varphi(\mathcal{F})$. By Theorem 5, Remark 5 and the fact that $\varphi(\mathcal{F})$ can be computed from $D \setminus \varphi(\mathcal{F})$ is time $O(|D|)$, we are done. $\square$

For what concerns **F-ACQ**$_1^{+}$ queries, the following result can be proved.

Theorem 7 *The query problem **F-ACQ**$_1^{+}$ can be solved in time $O(|\varphi|.|D|)$*

Proof. The proof, that is a generalization of the proof for **F-ACQ**$_1$, is similar and, in several aspects, is simpler than that of the similar result for **F-ACQ**$_1^{\neq}$. Let us mention essentially the differences. W.l.o.g., let $\varphi \in$ **F-ACQ**$_1^{+}$ be a formula of the form

$$\varphi(x) \equiv \exists y_1 \dots \exists y_{d-1} \exists y_d : \Psi(y_0, y_1, \dots, y_{d-1}) \wedge \mathbf{u}(y_{i_0}) = \mathbf{v}(y_d) \wedge f(y_{i_0}) \theta g(y_d) \wedge \gamma(y_d)$$

where y_0 is x, $\theta \in \{\neq, <, \leq, >, \geq\}$, $0 \leq i_0 \leq d-1$, $\gamma(y_d)$ is a quantifier-free formula on the unique variable y_d, and $\mathbf{u}(y_{i_0}) = \mathbf{v}(y_d)$ stands for $\bigwedge_{j \leq l} u_j(y_{i_0}) = v_j(y_d)$. Formula φ can be equivalently written as:

$$\varphi(x) \equiv \exists y_1 \dots \exists y_{d-1} : \Psi(\mathbf{y}) \wedge \delta(y_{i_0})$$

where $\mathbf{y} = (y_0, y_1, \dots, y_{d-1})$ and δ is the following two-variable formula:

$$\delta(y) \equiv \exists z : \gamma(z) \wedge \mathbf{u}(y) = \mathbf{v}(z) \wedge f(y) \theta g(z)$$

The key point is the following:

Lemma 8 *The set $D_0 = \delta(\mathcal{F}) = \{a \in D : (\mathcal{F}, a) \models \delta(y)\}$ is computable in time $O(|\delta|.|D|)$*

Proof. The set $B = \gamma(\mathcal{F}) = \{b \in D : (\mathcal{F}, b) \models \gamma(z)\}$ is obviously computable in time $O(|\gamma|.|D|)$. Assume that the comparison symbol θ is $<$ (the other cases are variants of this case). Now, compute and lexicographically sort the following lists of $(l+3)$-tuples (in time $O(l.|D|)$):

$$Y = \{(\mathbf{u}(y), f(y), 1, y) : y \in D\}, \text{ and}$$

$$Z = \{(\mathbf{v}(z), g(z), 0, z) : z \in B\}.$$

Then, merge the sorted lists Y, Z into the sorted list L. It is easy to see that the following fact holds:

Fact 3 *$\delta(\mathcal{F})$ is the set of elements $y \in D$ such that there exists $z \in B$ such that $\mathbf{u}(y) = \mathbf{v}(z)$ and $(\mathbf{u}(y), f(y), 1, y)$ occurs before $(\mathbf{v}(z), g(z), 0, z)$ in L.*

Using this fact, the following algorithm computes $\delta(\mathcal{F})$ (knowing set B) in time $O(l.|D|)$.

- Partition the sorted list L into nonempty (sorted) sublists $L(\mathbf{a})$, for $\mathbf{a} \in \mathbf{u}(D) \cup \mathbf{v}(B)$, according to the first l-tuple $\mathbf{u}(y) = \mathbf{a}$ or $\mathbf{v}(z) = \mathbf{a}$ of each tuple.
- In each sorted list $L(\mathbf{a})$, compute the last tuple, denoted by $Max_B(\mathbf{a})$, of the form $(\mathbf{v}(z) = \mathbf{a}, g(z), 0, z)$, $z \in B$, with $g(z)$ maximal if such element exists. Otherwise, set $Max_B(\mathbf{a}) = -\infty$.
- In each $L(\mathbf{a})$, compute the list $L^<(\mathbf{a})$ of the tuples of the form $(\mathbf{u}(y) = \mathbf{a}, f(y), 1, y)$ that occur before $Max_B(\mathbf{a})$ in $L(\mathbf{a})$. By convention, $L^<(\mathbf{a})$ is empty in case $Max_B(\mathbf{a}) = -\infty$.
- Return the set of elements y that appear in the lists $L^<(\mathbf{a})$. By Fact 3, this is clearly the required set $\delta(\mathcal{F})$.

Globally, $\delta(\mathcal{F})$ is computed in time $O((|\gamma| + l).|D|) = O(|\delta|.|D|)$. This proves the lemma. □

End of proof of Theorem 7: Let $\mathcal{F}'$ be the expansion of structure $\mathcal{F}$ defined as $\mathcal{F}' = (\mathcal{F}, D_0)$ where D_0 is the unary predicate defined as $D_0 = \delta(\mathcal{F})$. Let φ' denote the following formula, of signature expanded with D_0:

$$\varphi'(x) \equiv \exists y_1 \dots \exists y_{d-1} : \Psi(x, y_1, \dots, y_{d-1}) \wedge D(y_{i_0})$$

where $y_{i_0} \in \{x, y_1, \dots, y_{d-1}\}$. By construction, we have:

Fact 4 $\varphi(\mathcal{F}) = \varphi'(\mathcal{F}')$.

In order to simply compare the lengths of φ and φ', let us introduce a simplified notion of formula length: let $|\varphi|_s$ denote the number of quantifiers of φ plus its number of occurrences of atoms. Clearly, it holds: $|\varphi| = \Theta(|\varphi|_s)$. By construction, we get the following fact:

Fact 5 $|\varphi'|_s = |\varphi|_s - |\delta|_s + 1$.

Lemma 8 immediately yields the following:

Fact 6 *The expansion $\mathcal{F} \mapsto \mathcal{F}'$, i.e., the computation of the added unary relation $D_0 = \delta(\mathcal{F})$ is computed in time* $0(|\delta|_s.|D|)$.

Iterating the transformation $(\mathcal{F}, \varphi) \mapsto (\mathcal{F}', \varphi')$ d times allows to eliminate successively the quantifed variables $y_d, y_{d-1}, \dots, y_1$; this can be performed in total time $O((|\varphi|_s + d).|D|)$ by Facts 5 and 6, and hence in time $O(|\varphi|.|D|)$ as required. This completes the proof of the theorem. □

Remark 6 *Allowing more than one comparison along the edges of the tree decomposition leads to a class of queries that seems intrinsically non-linear. Let's consider the very simple following formula with two comparisons:*

$$\exists x \exists y : \ f_1(x) \leq g_1(y) \wedge f_2(x) \leq g_2(y).$$

Finding two satisfying witnesses x and y, amounts to find lexicographically ordered pairs $(f_1(x), f_2(x))$ and $(g_1(y), g_2(y))$ which seems not doable in linear time (even if "tables" (f_1, f_2) and (g_1, g_2) are already sorted).

The following theorem states the complexity of our (functional) acyclic queries in the general case.

Theorem 9 *The **F-ACQ**$^{\neq}$ (resp. **F-ACQ**, **F-ACQ**$^{+}$) query problem can be solved in time $f(|\varphi|).|D|.|\varphi(\mathcal{F})|$ (resp. $O(|\varphi|.|D|.|\varphi(\mathcal{F})|)$) for some function f.*

Proof. We prove that, for any function $f : \mathbb{N} \mapsto \mathbb{R}^+$, if problem **F-ACQ**$_1^{+}$ (resp. **F-ACQ**$_1^{\neq}$) can be solved in time $f(|\varphi|).|D|$ then problem **F-ACQ**$^{+}$ (resp. **F-ACQ**$^{\neq}$) can be solved in time $f(|\varphi|).|D|.|\varphi(\mathcal{F})|$ for the same function f. Combined with Theorem 6 and 7, this yields the desired result.

Let $\mathcal{F}$ be a functional structure and $\varphi(x_1, \ldots, x_k)$ be a formula for the query problem **F-ACQ**$^{+}$ or **F-ACQ**$^{\neq}$. For $i = 1, \ldots, k$, let:

$$E_i = \{(x_1, \ldots, x_i) \in D^i : \ (\mathcal{F}, x_1, \ldots, x_i) \models \exists x_{i+1} \ldots \exists x_k \varphi\}$$

Obviously, $\varphi(\mathcal{F}) = E_k$. Sets E_1, ..., E_k are computed inductively by the following algorithm that only evaluates strict acyclic queries as subroutines.

```
$E_1 \leftarrow \{x_1 \in D : (\mathcal{F}, x_1) \models \exists x_2 \ldots \exists x_k \varphi\}$
For $i$ from $2$ to $k$ do
    $E_i \leftarrow \emptyset$
    For all $(x_1, \ldots, x_{i-1}) \in E_{i-1}$ do
        $S \leftarrow \{x_i \in D : \ (\mathcal{F}, x_1, \ldots, x_i) \models \exists x_{i+1} \ldots \exists x_k \varphi\}$        (*)
        $E_i \leftarrow E_i \cup \{(x_1, \ldots, x_{i-1}, x_i) : \ x_i \in S\}$
    End
End
$\varphi(\mathcal{F}) \leftarrow E_k$
```

The main step of the algorithm, that is step (*), requires time $f(|\varphi|).|D|$. It is repeated, a number of times bounded by:

$$card(E_1) + card(E_2) + \ldots + card(E_k) \leq k.card(E_k) = |E_k| = |\varphi(\mathcal{F})|$$

This yields total time $f(|\varphi|).|D|.|\varphi(\mathcal{F})|$. □

Finally, let us give another consequence of our results in the functional setting. Any two-variable quantifier-free (CNF) functional formula $\psi(x, y)$ is acyclic because any undirected graph with at most two vertices is acyclic. Let $\textbf{F-FO}^{var2}$ denote the set of functional first-order formulas (not necessarily in prenex form) with only two variables x, y which may be quantified several times. Denote by $\textbf{F-FO}_1^{var2}$ its restriction to strict queries.

Corollary 10 *The $\textbf{F-FO}_1^{var2}$ query problem is computable in time $O_\varphi(|\mathcal{F}|)$.*

Proof. The proof is done by induction on the structure (i.e., subformulas) of the input formula φ by using Theorem 5. □

6 Application to the complexity of relational acyclic queries

In the context of "classical", i.e., relational conjunctive queries, Theorem 9 immediately yields the following improvement of the time bound $g(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|.\log^2|\mathbf{db}|$ (for some function g) proved by [PY99] for the complexity of acyclic queries with inequalities.

Corollary 11 *The $\textbf{ACQ}^{\neq}$ (resp. $\textbf{ACQ}_1^{\neq}$) query problem can be solved in time $f(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|$ (resp. $f(|Q|).|\mathbf{db}|$) where Q is the input query and $\mathbf{db}$ is the input database.*

Proof. This comes from Theorem 9 and from the fact that the class $\textbf{ACQ}^{\neq}$ can be linearly interpreted by the class $\textbf{F-ACQ}^{\neq}$ (see section 3). □

Another consequence of Theorems 6 and 9 is an alternative proof of the following well-known result of [Yan81] (see also [FFG02]) that we slightly generalize since now also restricted comparisons are allowed.

Corollary 12 *The $\textbf{ACQ}$ and $\textbf{ACQ}^+$ (resp. $\textbf{ACQ}_1$ and $\textbf{ACQ}_1^+$) query problems can be solved in time $O(|Q|.|\mathbf{db}|.|Q(\mathbf{db})|)$ (resp. $O(|Q|.|\mathbf{db}|)$).*

In a two-atom query each database predicate appears at most two times. These kind of queries have been studied in [KV00, Sar91] mainly in the context of query-containment. A consequence of Corollary 10, is the following.

Corollary 13 *Any two-atom conjunctive query with inequalities can be evaluated in time $O_\varphi(|\mathbf{db}|)$ i.e. in time $O_\varphi(|T_1| + |T_2|)$ where T_1 and T_2 are the two input tables.*

7 Enumeration of query results

For all kind of queries considered in this paper, the complexity of the evaluation process can be done in time $f(|Q|).|\mathbf{db}|.|Q(\mathbf{db})|$. In other words, coming back to data complexity, this is equivalent to say that there exists a polynomial total time algorithm (in the size of the input and the output) that generates the output tuples. It is natural to ask whether one can say more on the efficiency of this enumeration process. This could be justified, for example, in situation where only parts of the results are really needed quickly or when having solutions one by one but regularly is required (e.g., in order to be tested by an other procedure that runs in parallel). Some remarks on this subject are sketched in this section.

One of the most widely accepted notion of tractability in the context of generation of solutions is the following. A problem P is said to be solvable within a *Polynomial (resp. linear) Delay* if there exists an algorithm that outputs a first solution in polynomial (resp. linear) time (in the size of the input only) and generates all solutions of P with a polynomial (resp. linear) delay between two consecutives ones (see [JYP88] for an introduction to complexity measures for enumeration problems). Of course, a *Polynomial Delay* algorithm is polynomial total time (but, unless surprise, the converse is not true).

Not too surprisingly, our complexity results can be adapted to obtain polynomial, even linear, delay algorithms for acyclic queries as shown by the following corollary.

Corollary 14 *Generating all results of a* **F-ACQ**$^{\neq}$ *(resp.* **F-ACQ**, **F-ACQ**$^{+}$, **ACQ**$^{\neq}$, **ACQ**, **ACQ**$^{+}$*) query can be done with a linear delay (and with linear space also).*

Proof. We proceed in a similar way as for Theorem 9. Results for relational query classes are obtained by reduction. Let $\mathcal{F}$ be a functional structure and $\varphi(x_1, \dots, x_k)$ be a functional query in **F-ACQ**$^{\neq}$, **F-ACQ** or **F-ACQ**$^{+}$.

The simple (recursive) algorithm below outputs all satisfying tuples of $\varphi(x_1, \dots, x_k)$.

Algorithm 1 Eval$(i, \varphi(x_i, \dots, x_k), \mathcal{F}, sol)$

```
if $i = k + 1$ then
    Output $sol$
end if
$E_i \leftarrow \{x_i \in D : (\mathcal{F}, x_i) \models \exists x_{i+1} \dots \exists x_k \varphi\}$
for $a \in E_i$ do
    $sol \leftarrow (sol, a)$
    $\varphi \leftarrow \varphi(x_i/a, x_{i+1}, \dots, x_k)$
    Eval$(i + 1, \varphi, \mathcal{F}, sol)$
end for
```

Due to results of the preceding sections, computing E_i can be done, in all cases, in time $f(|\varphi|.|D|)$. Then, running *Eval*$(1, \varphi(x_1, \dots, x_k), \mathcal{F}, \emptyset)$ generates all solutions sol in a depth-first manner with a linear delay between each of them. It can be easily rewritten in a sequential way to use linear space. □

8 Fixed-parameter linearity of some natural problems

In this part of the paper, the different kind of formulas introduced so far are used to define classical algorithm properties as query problems. This method provides a simple and uniform method to cope with the complexity of these problems. In all cases, the complexity bound found with this method reaches or improve the best bound known so far (at least in terms of data complexity). However, some of these problems have been the object of intensive researches and recent optimize ad-hoc algorithms (against which a general and uniform method can not compete) have better constant values.

8.1 Acyclic Subgraph problems

Given two graphs $G = \langle V; E\rangle$ and $H = \langle V_H; E_H\rangle$, H is said to be a *subgraph* (resp. *induced subgraph*) of G if there is a one-to-one function g from V_H to V such that, for all $u, v \in V_H$, $E(g(u), g(v))$ if (resp. if and only if) $E(u, v)$. Also, a graph G is of maximum degree d if none of its vertex belongs to more than d edges. This gives rise to the two following problems.

ACYCLIC SUBGRAPH ISOMORPHISM (A.S.I.)

Input:	an acyclic graph H and a graph G
Parameter:	$\vert H\vert$.
Question:	is H a subgraph of G ?

ACYCLIC INDUCED SUBGRAPH ISOMORPHISM (A.I.S.I.)

Input:	an acyclic graph H and a graph G of maximum degree d
Parameter:	$\vert H\vert, d$.
Question:	is H an induced subgraph of G ?

The treewidth of a graph G is the maximal size of a node in a tree decomposition of G. In [PV90] it is proved that for graphs H of treewidth at most w, testing is H is a subgraph (resp. induced subgraph) of G can be done in time $f(|H|).|G|^{w+1}$ (resp. $f(|H|, d).|G|^{w+1}$). For the particular case of acyclic graphs (which have tree width 1), the bounds given in [PV90] can be improved. The following corollary is easily obtained from our results.

Corollary 15 *The two following results hold:*

- *Problem* A.S.I. *can be solved in time* $f(|H|).|G|$.
- *Problem* A.I.S.I. *can be solved in time* $f(|H|, d).|G|$.

For the two problems, generating all satisfying subgraphs can be done with a linear delay.

Proof. We will express problem A.S.I. as a boolean $\mathbf{ACQ}^{\neq}$ query. Let $G = \langle V; E\rangle$, $H = \langle V_H = \{h_1, \ldots, h_k\}; E_H\rangle$ be the two input graphs. Let Q be the following formula:

$$Q \equiv \exists x_1 \ldots \exists x_k : \bigwedge_{i,j \le k} x_i \neq x_j \wedge \bigwedge_{E_H(h_i,h_j)} E(x_i, x_j)$$

Since H is acyclic, formula Q defines an $\mathbf{ACQ}^{\neq}$ query whose size is linear in the size of the graph H. It is easily seen that Q is true in G if and only if it admits H as a subgraph. The complexity bound follows from Corollary 11.

For problem A.I.S.I., let again G and $H = \langle V_H = \{x_1, \ldots, x_k\}; E_H\rangle$ be the two inputs of the problem. Since ^{G} is of maximum degree d, we partition its vertex set V into d sets $V^1, \ldots, V^d$ where each V^α contains vertex of degree α. This can be done in linear time from G. We proceed the same for graph H and obtain the sets $V_H^1, \ldots, V_H^d$. In case there exists a vertex in H of degree greater than d, it can be concluded immediately that the problem has no solution. Now, let Q be the following formula:

$$Q \equiv \exists x_1 \ldots \exists x_k : \bigwedge_{i,j \leq k} x_i \neq x_j \wedge \bigwedge_{V_H^\alpha(h_i)} V_H^\alpha(x_i) \wedge \bigwedge_{E_H(h_i,h_j)} E(x_i, x_j).$$

Formula Q simply check that H is a subgraph of G and that each distinguished vertex x_i has the same degree than its associated vertex h_i of H. The size of Q is linear in the size of H and d. Again, Q defines a boolean $\mathbf{ACQ}^{\neq}$ query and the result follows again from Corollary 11. The bound on the linear delay comes from Corollary 14. □

8.2 Covering and matching problems

MULTIDIMENSIONAL MATCHING

Input:	a set $M \subseteq X_1 \times \ldots \times X_r$ where the X_i are pairwise disjoints
Parameter:	r, k.
Question:	is there a subset $M' \subseteq M$ with $\lvert M'\rvert = k$, such that no two elements of M' agree in any coordinate ?

Corollary 16 *Problem* MULTIDIMENSIONAL MATCHING *can be solved in time* $O_{r,k}(|M|)$.

Proof. Let $\mathcal{F}_M = \langle M; f_1, \ldots, f_r\rangle$ where for all $x = (x_1, \ldots, x_r) \in M$, it is set $f_i(x) = x_i$. Then, there exists a multidimensional matching M' of M if and only if:

$$\mathcal{F}_M \models \exists x_1 \ldots \exists x_k : \bigwedge_{i \leq r} \bigwedge_{1 \leq j < h \leq k} f_i(x_j) \neq f_i(x_h)$$

□

Corollary 17 below improves the bound of $O_{r,k}(|M|(\log |M|)^6)$ (reported in [DF99]) obtained by perfect hashing methods. A recent result however of [FKN$^+$04] based on the color coding method of [AYZ95] gives a bound of $O(|M| + 2^{O(k)})$ for the r-MULTIDIMENSIONAL MATCHING problem.

The following problems are also known to be fixed-parameter tractable [DF99].

UNIQUE HITTING SET

Input:	a set X and k subsets $X_1, \ldots, X_k$ of X.
Parameter:	k.
Question:	is there a set $S \subseteq X$ such that for all i, $1 \leq i \leq k$, $\lvert S \cap X_i\rvert = 1$?

ANTICHAIN OF r-SUBSETS

Input:	a collection $\mathcal{F}$ of r subsets of a set X, a positive integer k.
Parameter:	r, k.
Question:	are there k subsets $S_1, \ldots, S_k \in \mathcal{F}$ such that $\forall i, j \in \{1, \ldots, k\}$ with $i \neq j$, both $S_i - S_j$ and $S_j - S_i$ are nonempty ?

DISJOINT r-SUBSETS

Input:	a collection $\mathcal{F}$ of r subsets of a set X, a positive integer k.
Parameter:	r, k.
Question:	are there k disjoint subsets of $\mathcal{F}$?

Corollary 17 *Problems* UNIQUE HITTING SET, ANTICHAIN OF r-SUBSETS *and* DISJOINT r-SUBSETS *can be solved in time $O_{r,k}(|M|)$. In all cases, the respective sets of solutions can be generated with a linear delay.*

Proof. The following acyclic formula holds for the UNIQUE HITTING SET problem:

$$\varphi \equiv \exists x_1 \dots \exists x_k : \bigwedge_{i \leq k} X_i(x_i) \bigwedge_{1 \leq i < j \leq k} (x_i \neq x_j \Rightarrow \neg X_j(x_i)).$$

The formulas are similar for the two other problems. □

9 Conclusion: summary of results and open problems

The following array summarizes the main results of this paper. For all our classes of ("classical", i.e., relational, or functional) queries we make use here of the notation φ for the query formula, $\mathcal{S}$ for the database, i.e., the input structure **db** or $\mathcal{F}$, and $\varphi(\mathcal{S})$ for the result of the query, i.e., the output.

Query Problems	Complexity
$\mathbf{ACQ}_1, \mathbf{ACQ}_1^+, \mathbf{F\text{-}ACQ}_1, \mathbf{F\text{-}ACQ}_1^+$	$\lvert\varphi\rvert.\lvert\mathcal{S}\rvert$
$\mathbf{ACQ}_1^{\neq}, \mathbf{F\text{-}ACQ}_1^{\neq}, \mathbf{F\text{-}FO}_1^{var2}$	$f(\lvert\varphi\rvert).\lvert\mathcal{S}\rvert$
$\mathbf{ACQ}, \mathbf{ACQ}^+, \mathbf{F\text{-}ACQ}, \mathbf{F\text{-}ACQ}^+$	$\lvert\varphi\rvert.\lvert\mathcal{S}\rvert.\lvert\varphi(\mathcal{S})\rvert$
$\mathbf{ACQ}^{\neq}, \mathbf{F\text{-}ACQ}^{\neq}$	$f(\lvert\varphi\rvert).\lvert\mathcal{S}\rvert.\lvert\varphi(\mathcal{S})\rvert$

Note that among those complexity results the only ones to be known before this paper (to our knowledge) where those concerning **ACQ** and $\mathbf{ACQ}_1$.

We are convinced that (variants of) our technics of construction of *minimal samples* can be efficiently implemented to compute such queries. The reason is that we think that the total number of minimal samples should be very low in most databases.

Finally, four lines of research are worthwhile to develop:

- Generalize our complexity results to tractable or f.p. tractable tree-like queries e.g., queries of bounded tree-width (see [CR00, FFG02]) or of bounded hypertree-width ([GLS02]). Our reduction technic from relational to functional queries, which preserves acyclicity, may permit also to control the value of the tree-width when passing from one context to the other.

- Apply our results to constraint satisfaction problems by using the now well-known correspondence between conjunctive query problems and constraint problems (see among others [KV00]).

- Enlarge the classes of tractable or f.p. tractable problems as much as possible, i.e., determine the frontier of tractability/intractability, and obtain for the (f.p.) tractable problems the best sequential or parallel algorithms; e.g., it is reasonable to conjecture that the $\mathbf{ACQ}^+$ evaluation problem is highly parallelizable as it is known for **ACQ** (see for example [GLS01] which proves that this problem is LOGCFL-complete).

- Apply our methods to queries over tree-structured data (recall that a rooted tree can be seen as a graph of a unary function).

References

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundation of databases*. Addison-Wesley, 1995.

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color coding. *Journal of the ACM*, 42(4):844–856, 1995.

[CM77] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In ACM New York, editor, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[CR00] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

[DF99] R. G. Downey and M. R. Fellows. *Parametrized complexity*. Springer-Verlag, 1999.

[DGO04] A. Durand, E. Grandjean, and F. Olive. New results on arity vs. number of variables. Research report 20-2004, LIF, Marseille, France, April 2004.

[EF99] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 2nd edition, 1999.

[Fag83] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.

[FFG02] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree decompositions. *Journal of the ACM*, 49(6):716–752, 2002.

[FKN$^+$04] M. Fellows, C. Knauer, N. Nishimura, P. Ragde, F. Rosamond, U. Stege, D. Thilikos, and S. Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. In *European Symposium on Algorithms 2004*, pages 311–322, 2004.

[GLS01] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.

[GLS02] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[GO04] E. Grandjean and F. Olive. Graphs properties checkable in linear time in the number of vertices. *Journal of Computer and System Sciences*, 68(3):546–597, 2004.

[Gra79] R. Graham. On the universal relation. Technical report, Univ. Toronto, 1979.

[GS02] E. Grandjean and T. Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM Journal on Computing*, 32(1):196–230, 2002.

[JYP88] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

[KV00] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Science*, 61(2):302–332, 2000.

[Lib04] L. Libkin. *Elements of finite model theory*. EATCS Series. Springer, 2004.

[PV90] J. Plehn and B. Voigt. Finding minimally weighted subgraphs. In Springer, editor, *16th workshop on graph theoretic concepts in computer science*, volume 484 of *Lecture Notes in Computer Science*, pages 18–29, 1990.

[PY99] C. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

[Sar91] Y. Saraiya. *Subtree elimination algorithms in deductive databases*. PhD thesis, Stanford University, 1991.

[Ull89] J.D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press, 1989.

[Yan81] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Databases*, pages 82–94, 1981.

[YO79] C. T. Yu and M. Z Özsoyoglu. An algorithm for tree-query membership of a distributed query. In IEEE Computer Society Press, editor, *IEEE COMPSAC*, pages 306–312, 1979.