On acyclic conjunctive queries and constant delay enumeration

Guillaume Bagan * Arnaud Durand † Etienne Grandjean ‡

Abstract

We study the enumeration complexity of the natural extension of acyclic conjunctive queries with disequalities. In this language, a number of **NP**-complete problems can be expressed. We first improve a previous result of Papadimitriou and Yannakakis by proving that such queries can be computed in time $c.|\mathcal{M}|.|\varphi(\mathcal{M})|$ where $\mathcal{M}$ is the structure, $\varphi(\mathcal{M})$ is the result set of the query and c is a simple exponential in the size of the formula φ. A consequence of our method is that, in the general case, tuples of such queries can be enumerated with a linear delay between two tuples.

We then introduce a large subclass of acyclic formulas called $\mathbf{CCQ}^{\neq}$ and prove that the tuples of a $\mathbf{CCQ}^{\neq}$ query can be enumerated with a linear time precomputation and a constant delay between consecutive solutions. Moreover, under the hypothesis that the multiplication of two $n \times n$ boolean matrices cannot be done in time $O(n^2)$, this leads to the following dichotomy for acyclic queries: either such a query is in $\mathbf{CCQ}^{\neq}$ or it cannot be enumerated with linear precomputation and constant delay. Furthermore we prove that testing whether an acyclic formula is in $\mathbf{CCQ}^{\neq}$ can be performed in polynomial time.

Finally, the notion of free-connex treewidth of a structure is defined. We show that for each query of free-connex treewidth bounded by some constant k, enumeration of results can be done with $O(|\mathcal{M}|^{k+1})$ precomputation steps and constant delay.

*GREYC - CNRS UMR 6072, Université de Caen - Campus 2, F-14032 Caen cedex - France. Email: gbagan@info.unicaen.fr

†Équipe de Logique Mathématique - CNRS UMR 7056, UFR de mathématiques, 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 - Campus 2, F-14032 Caen cedex - France. Email: etienne.grandjean@info.unicaen.fr

Contents

1 Preliminaries **4**
- 1.1 Problems . . . 4
- 1.2 Complexity framework . . . 4
- 1.3 Logical framework . . . 6
- 1.4 Hypergraphs . . . 6
- 1.5 Acyclic conjunctive queries . . . 6
- 1.6 Acyclic conjunctive functional queries. . . . 7

2 Covers and representative sets **10**

3 Quantifier elimination **14**
- 3.1 Improving the constant size . . . 15

4 Linear delay enumeration **19**

5 Constant delay enumeration **20**
- 5.1 Combinatorial tools . . . 20
- 5.2 Enumeration of quantifier-free and free-connex acyclic queries . . . 22

6 A dichotomy result **23**
- 6.1 S-connectivity and free-connex queries . . . 23
- 6.2 Polynomial time characterization of S-connex acyclic hypergraphs . . . 25
- 6.3 A dichotomy theorem . . . 30

7 Free-connex treewidth **32**

8 Fixed-parameter linearity of some natural problems **34**
- 8.1 Acyclic Subgraph problems . . . 34
- 8.2 Covering and matching problems . . . 36

Introduction

From a complexity theoretical point of view, very few is known about enumeration problems and their complexity classes. In the context of logical or database query languages, enumeration amounts to produce one by one all the tuples of the result $\varphi(\mathcal{M})$ of a query $\varphi(\bar{x})$ over a structure $\mathcal{M}$. Producing solutions of a query with a regular delay or on user demand makes sense in that area. Although a lot of results are known about the complexity of query evaluation in the classical sense (from hard cases to islands of tractability) [6, 9, 13, 15, 17], only some recent works have started to investigate complexity issues of enumeration for query languages [8, 10, 1, 11, 2].

A large and well-studied class of queries is the class **ACQ** of acyclic conjunctive queries. A classical result of database theory, proved in [18], is that any acyclic conjunctive query φ can be evaluated in time $O(|\varphi|.|\mathcal{M}|.|\varphi(\mathcal{M})|)$. In [17], an extension $\mathbf{ACQ}^{\neq}$ is introduced in which disequalities between variables are allowed. In full generality, deciding a boolean $\mathbf{ACQ}^{\neq}$ query is **NP**-complete for combined complexity. However, it is shown in [17] that such a query can be evaluated in time $f(|\varphi|).|\mathcal{M}|.|\varphi(\mathcal{M})|.\log^2|\mathcal{M}|$ where f is some exponential function. Many **NP**-complete problems can be expressed by $\mathbf{ACQ}^{\neq}$ formulas: existence of some path of a given length k in a graph, multidimensional matching, color matching, etc.

The main goal of this paper is to revisit the complexity of the evaluation of conjunctive queries with or without disequalities from a dynamical point of view. As in [8, 10, 1, 11, 2], queries are seen as enumeration problems and the complexity is measured in terms of delay between two consecutive solutions. We consider the class of **F-ACQ** (resp $\mathbf{F\text{-}ACQ}^{\neq}$) formulas which are the "equivalent" of acyclic conjunctive formulas in functional structures. First, we prove that an elimination of quantifiers can be done on such formulas. More precisely, using a combinatorial argument, we show that given an $\mathbf{F\text{-}ACQ}^{\neq}$ formula φ and a structure $\mathcal{M}$ one can construct a formula φ' which is a disjunction of quantifier-free formulas of $\mathbf{F\text{-}ACQ}^{\neq}$ and a model $\mathcal{M}'$ such that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$. As a consequence, we prove that any query of $\mathbf{F\text{-}ACQ}^{\neq}$ can be evaluated with a linear time delay. This immediately implies that a boolean $\mathbf{ACQ}^{\neq}$ query can be evaluated in time $f(|\varphi|).|\mathcal{M}|$ and a general $\mathbf{ACQ}^{\neq}$ query can be evaluated in time $f(|\varphi|).|\mathcal{M}|.|\varphi(\mathcal{M})|$ where f is a simple exponential function which depends on the number of variables and the number of disequalities of the formula.

Constant-Delay$_{lin}$ introduced in [10] is the class of enumeration problems which are solved by algorithms that run in two phases : the first phase performs a precomputation in linear time and the second phase, using the precomputation, enumerates all the solutions with a constant delay between two consecutive ones. This class is regarded as the minimal (nontrivial) robust class for the complexity of enumeration. A natural question addressed is this paper is the following: is there a large subclass of $\mathbf{ACQ}^{\neq}$ queries whose enumeration belongs to Constant-Delay$_{lin}$? We introduce the class $\mathbf{CCQ}^{\neq}$ of formulas called *free-connex acyclic* formulas and defined by a connectivity condition between the free variables (in particular, it contains the quantifier-free acyclic formulas with disequalities). We show that such queries can be evaluated with linear precomputation and constant delay. Furthermore, it is proved that testing whether a formula is free-connex acyclic can be performed in polynomial time. This result is optimal in some sense because we prove the following dichotomy theorem: under the assumption that any two $n \times n$ boolean matrices *cannot* be multiplied in time $O(n^2)$ (a very reasonable conjecture),

each acyclic conjunctive query (with or without disequalities) which is *simple* [1] either is free-connex acyclic or cannot be enumerated with linear precomputation and constant delay. Finally, we introduce the notions of *free-connex tree-decomposition* and *free-connex treewidth*. We show that for any fixed query of free-connex treewidth bounded by some constant k, enumeration of solutions can be done with $O(|D|^{k+1} + |\mathcal{M}|)$ precomputation steps and constant delay (D is the domain of $\mathcal{M}$).
Notice that essentially all the results of this paper are new, to our knowledge, even if we consider only conjunctive queries (without disequalities). The paper is organized as follows. In Section 1, the main definitions about complexity, queries, hypergraphs and enumeration problems are given. Our results about quantifier elimination and enumeration are proved respectively in Sections 3 and 5. The notion of free-connex acyclic formula and the dichotomy theorem are exposed in Section 6. Finally, in Section 7, results about free-connex treewidth queries are presented.

1 Preliminaries

1.1 Problems

In order to handle problems in a general and uniform framework, let us consider the notion of *general problem* which is merely a binary relation $A \subseteq I \times O$ over two sets I and O respectively called *space of instances* (or *input space*) and *space of solutions* (or *output space*). For each input $x \in I$, the set $A(x) = \{y \in O : (x, y) \in A\}$ is assumed to be finite and is called the *set of solutions* of A on x. Let $<$ be a strict linear order on the space of solutions of A. For any input x and $A(x) = \{y_0, \ldots, y_{n-1}\}$ with $y_0 < y_1 < \ldots < y_{n-1}$, the increasing list of elements of $A(x)$ is denoted by $\text{enum}(A, x, <) = y_0 y_1, \ldots y_{n-1}$. We are interested in the following problems.

EVAL(A)
Input: an instance $x \in I$
Output: $A(x)$

ENUM($A, <$)
Input: an instance $x \in I$
Output: $\text{enum}(A, x, <)$

In the case the order does not matter for enumeration, it is omitted and we write ENUM(A).

1.2 Complexity framework

The computation model used in this paper is the random access machine (RAM) with uniform cost measure and addition and substraction as the basic operations (see [14]). It has read-only input registers (containing the input x), read-write work memory registers and write-only output registers. Let $|x|$ be the size of the input x, that is, the number of

[1] This means that we assume that each relation symbol occurs once only in the conjunction (a slight restriction)

registers it occupies. Register values and memory addresses are assumed to be bounded by $c|x|$.
An algorithm $\mathcal{A}$ is said to compute the problem $\textsc{Enum}(A, <)$ if for any input x, $\mathcal{A}$ computes one by one the elements of the sequence $enum(A, x, <)$ and stops immediately after writing the last one. We define $\text{delay}_j(x) = \text{time}_j(x) - \text{time}_{j-1}(x)$ where $\text{time}_j(x)$ denotes the moment when $\mathcal{A}$ has completed the writing of the j^{th} solution (by convention $\text{time}_0(x) = 0$). We say that $\mathcal{A}$ is a *constant delay* algorithm if for a constant c and any input x, we have $\text{delay}_j(x) \leq c$ and if furthermore, $\mathcal{A}$ uses space c (i.e. at most c memory registers) during its computation.

Definition 1 *A problem* $\textsc{Enum}(A, <)$ *is computable* with constant delay and linear precomputation*, which is written* $\textsc{Enum}(A, <) \in \textsc{Constant-Delay}_{lin}$ *if there is a function* $r : x \mapsto r(x)$ *(precomputation phase) computable in linear time and a constant delay algorithm* $\mathcal{A}$ *(enumeration phase) which, when applied to* $r(x)$ *for any input* x*, computes* $enum(A, x, <)$.

Note that the additional requirement that the enumeration phase uses constant space (uniform cost measure) appears to be a very strong condition. Nevertheless we have checked that it holds for all the constant delay algorithms we know (see [1, 10, 11, 2]). However, this condition is not essential in this paper. We need a very precise notion of reduction for enumeration problems.

Definition 2 *Let* $A \subseteq I_A \times O_A$ *and* $B \subseteq I_B \times O_B$ *be two general problems. An* exact reduction *from* $\textsc{Enum}(A)$ *to* $\textsc{Enum}(B)$ *(resp. from* $\textsc{Enum}(A, <_A)$ *to* $\textsc{Enum}(B, <_B)$*) is a pair* (s, t) *of mappings* $s : I_A \to I_B$ *and* $t : I_B \times O_B \to O_A$ *which satisfy the two conditions below.*

1. *For every instance* x *of* A*, the correspondence* $y \mapsto t(s(x), y)$ *is a one-one (resp. strictly increasing) mapping from the set (resp* $<_B$*-ordered set)* $B(s(x))$ *onto the set (resp.* $<_A$*-ordered set)* $A(x)$.

2. *The instance* $s(x)$ *of* B *is computable from* x *in time and space* $O(|x|)$ *and the solution* $t(s(x), y)$ *is computable from* $(s(x), y)$ *in constant time and constant space.*

The following results are easy consequences of our definitions.

Lemma 1 *Assume there exists an exact reduction from* $\textsc{Enum}(A)$ *to* $\textsc{Enum}(B)$*. Then* $\textsc{Enum}(B) \in \textsc{Constant-Delay}_{lin}$ *implies* $\textsc{Enum}(A) \in \textsc{Constant-Delay}_{lin}$*. The similar result holds for* $\textsc{Enum}(A, <_A)$ *and* $\textsc{Enum}(B, <_B)$.

Proof - Suppose $\textsc{Enum}(B) \in \textsc{Constant-Delay}_{lin}$, one briefly describes a constant delay algorithm for $\textsc{Enum}(A)$ (under the notations of Definitions 1 and 2). Let $x \in I_A$. As a precomputation phase, one computes $s(x)$ and $r(s(x))$ in linear time and space $O(|x|)$. Then, one starts the enumeration phase $\mathcal{B}$ for $\textsc{Enum}(B)$ on input $r(s(x))$. For each solution y of $s(x)$ produced by $\mathcal{B}$, one outputs $t(s(x), y)$. Since $\mathcal{B}$ runs in constant delay and retrieving $t(s(x), y)$ from $(s(x), y)$ can be done in constant time and space, the conclusion follows. □

Lemma 2 *Let $A, B \subseteq I \times O$ be two general problems over the same input and output spaces. Let $<$ be a linear order on O. If both problems* $\textsc{Enum}(A, <)$ *and* $\textsc{Enum}(B, <)$ *belong to* $\textsc{Constant-Delay}_{lin}$*, then so does the union problem* $\textsc{Enum}(A \cup B, <)$.

Proof - It suffices to concurently run the enumeration phases of $\textsc{Enum}(A, <)$ and $\textsc{Enum}(B, <)$. At each step, the next solutions that both algorithms may output are compared in constant time and only the smallest one of the two is outputed. □

1.3 Logical framework

The reader is supposed to be familiar with the basics of first-order logic on finite structures (see [16]). For a signature σ, a (finite) σ-structure consists of a domain D together with an interpretation of each symbol of σ over D (we do not distinguish between a symbol and its interpretation). For each σ-formula (or σ-query) $\varphi(\bar{x})$ with m variables $\bar{x} = (x_1, \ldots, x_m)$ and for each σ-structure $\mathcal{M}$ of domain D, we denote by $\varphi(\mathcal{M})$ the set $\{\bar{a} \in D^m : \mathcal{M} \models \varphi(\bar{a})\}$. Let $<$ be a linear order on D^m. We are interested in the evaluation and enumeration problems associated to $\varphi(\bar{x})$. Note that in most of our problems formula φ is considered as *fixed*.

$\textsc{Eval}(\varphi)$
Input: a σ-structure $\mathcal{M}$
Output: $\varphi(\mathcal{M})$

$\textsc{Enum}(\varphi, <)$
Input: a σ-structure $\mathcal{M}$
Output: an $<$-ordered enumeration of $\varphi(\mathcal{M})$

1.4 Hypergraphs

Let us introduce some definitions on hypergraphs. A hypergraph is a pair (V, E) where V is a set of elements, called *vertices*, and E is a set of non-empty subsets of V called *hyperedges*. The *induced subhypergraph* of $H = (V, E)$ on a set of vertices $S \subseteq V$ is the hypergraph [2] $H[S] = (S, \{e \cap S \neq \emptyset : e \in E\})$. A hypergraph is *acyclic* if there is a tree T, called a *tree-structure* of H, whose vertices are the hyperedges of H and having the following *connectivity property*: for each vertex v of H, the set of hyperedges that contain v consists of a subtree (connected subgraph) of T. For two hypergraphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$, let us denote by $H_1 \cup H_2$ the hypergraph $(V_1 \cup V_2, E_1 \cup E_2)$. Let $x, y \in V$, a *path* between x and y is a subset of edges $e_1, \ldots, e_k$ such that $x \in e_1$, $y \in e_k$, and for all $i \leq k-1$, $e_i \cap e_{i+1} \neq \emptyset$. Finally, given a graph $G = (V, E)$, the neighborhood of x denoted by $N(x)$ is $\{y \in V : \{x, y\} \in E\}$. For more details on hypergraphs see [3].

1.5 Acyclic conjunctive queries

As usual, let **CQ** (resp. $\mathbf{CQ}^{\neq}$) denote the set of *conjunctive queries* (resp *conjunctive queries with disequalities*), i.e. relational σ-formulas of the form $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where

[2] Note that taking $e \cap S$ may be a multiset. In that case, only one copy of each hyperedge is kept to define $H[S]$ as an hypergraph

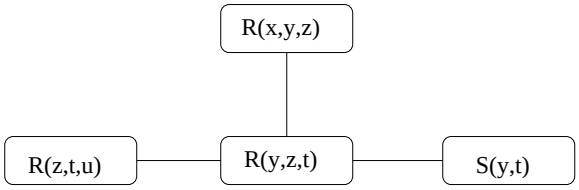

Figure 1: The join-tree of φ

ψ is a conjunction of σ-atoms $R(\bar{v})$ for $\bar{v} \subseteq \{\bar{x}, \bar{y}\}$ (resp σ-atoms and disequalities $u \neq v$ for $u, v \in \{\bar{x}, \bar{y}\}$).
To each conjunctive query φ one can associate a hypergraph $H_\varphi = (V_\varphi, E_\varphi)$ defined as follows:

- $V_\varphi = var(\varphi)$ and,
- $E_\varphi = \{\{v_1, \ldots, v_{p_i}\} : R_i(v_1, \ldots, v_{p_i})$ is a σ-atom of $\varphi\}$.

Note that disequalities, if there are, are not taken into account in the definition of acyclicity. A conjunctive query φ (resp. with disequalities) is *acyclic* or **ACQ** (resp. $\mathbf{ACQ}^{\neq}$) if its hypergraph H_φ is acyclic A tree-structure of H_φ is said to be a *join-tree* of φ.

EXAMPLE. $\varphi(x, y) \equiv \exists z \exists t \exists u (Rxyz \wedge Ryzt \wedge Rztu \wedge Syt \wedge x \neq u)$ is an $\mathbf{ACQ}^{\neq}$ formula whose join-tree T_φ is given in Figure 1. $\dashv$

1.6 Acyclic conjunctive functional queries.

An acyclic conjunctive σ-query can be naturally translated in the functional framework, i.e., into a σ'-formula for a signature σ' that consists of unary fuction symbols and unary predicates only. Intuitively, instead of considering relational structures on some domain one considers sets of tuples with unary functions describing their projections on the initial domain.

Formula translation: Introduce for each σ-atom $A_i, 1 \leq i \leq q$, of the formula a new variable α_i.

*Translation (1) of a quantifier-free **ACQ** σ-formula*: $\psi(\bar{x}) \equiv \bigwedge_{1 \leq i \leq q} A_i$. Associate to ψ the following quantifier-free functional σ'-formula:

$$\psi'(\alpha_1, \ldots, \alpha_q) \equiv \bigwedge_{1 \leq i \leq q} D_{R_i}(\alpha_i) \wedge \bigwedge_{(A_i, A_j) \in T_\psi} \bigwedge_{\mathrm{var}_h(A_i) = \mathrm{var}_k(A_j)} f_h(\alpha_i) = f_k(\alpha_j) \quad (1)$$

where $\sigma' = \{D_R : R \in \sigma\} \cup \{f_1, \ldots, f_p\} \cup \{Id\}$ [3], $p = \mathrm{arity}(\sigma) = \max_{R \in \sigma}(\mathrm{arity}(R))$, T_ψ is a join-tree of ψ, $A_i \equiv R_i(v_i^1, \ldots, v_i^h, \ldots, v_i^{p_i})$, p_i is the arity of R_i and $\mathrm{var}_h(A_i)$ denotes the variable v_i^h that occurs at rank h in atom A_i.

[3] Id will be interpreted by the identity function

Translation (2) of any ***ACQ*** *formula* $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where ψ is quantifier-free and $\bar{x} = (x_1, \ldots, x_m)$. Associate to φ the following functional σ'-formula:

$$\varphi''(\bar{x}) \equiv \exists \alpha_1 \ldots \exists \alpha_q (\psi'(\bar{\alpha}) \wedge \bigwedge_{1 \leq i \leq m} f_{j_i}(\alpha_{k_i}) = x_i) \tag{2}$$

where ψ' is the above translation (1) of the quantifier-free part ψ of φ and A_{k_i} (represented by variable α_{k_i}) is a chosen atom of φ in which x_i occurs (at rank j_i).

Translation of the structure: Each relational σ-model $\mathcal{M} = (D; R_1, \ldots, R_s)$ is translated into the functional σ'-model $\mathcal{M}' = (D'; D, D_{R_1}, \ldots, D_{R_s}, f_1, \ldots, f_p)$:

- $D, D_{R_1}, \ldots, D_{R_s}$ are disjoint unary relations so that $D' = D \cup D_{R_1} \cup \ldots \cup D_{R_s} \cup \{\perp\}$ and, for each $i = 1, \ldots, s$, we have $D_{R_i} = R_i$, i.e. D_{R_i} is the set of tuples of R_i, and $\perp$ is an extra element.
- Each f_j $(1 \leq j \leq p)$ is a unary function: $D' \to D \cup \{\perp\}$ such that, for every $t = (a_1, \ldots, a_{p_i}) \in D_{R_i}$, we have $f_j(t) = a_j$ for $1 \leq j \leq p_i$ and $f_j(t) = \perp$ otherwise.

EXAMPLE. Translation 1 transforms the quantifier-free **ACQ** formula $\psi(x, y, z) \equiv R(x, y) \wedge S(y, z)$ into the following (quantifier-free) formula: $\psi'(\alpha_1, \alpha_2) \equiv D_R(\alpha_1) \wedge D_S(\alpha_2) \wedge f_2(\alpha_1) = f_1(\alpha_2)$. ⊣

EXAMPLE. Translation 2 transforms the **ACQ** formula $\varphi(x, z) \equiv \exists y (R(x, y) \wedge S(y, z))$ into the following formula (with the same free variables and the subformula ψ' as above): $\varphi''(x, z) \equiv \exists \alpha_1 \exists \alpha_2 (\psi'(\alpha_1, \alpha_2) \wedge f_1(\alpha_1) = x \wedge f_2(\alpha_2) = z)$. ⊣

REMARK. Our translations (1) and (2) are easily extended to $\mathbf{ACQ}^{\neq}$ formulas. ⊣

On the model of "classical", say relational, conjunctive queries, one now introduce the notions of conjunctive and acyclic conjunctive functional queries. Basically, functional conjunctive queries are first-order queries on a unary functional signature built over existential quantification and conjunction only. Here again, one may allow disequalities.

Definition 3 *The set of conjunctive functional queries (resp. with disequalities), denoted* ***F-CQ*** *(resp.* ***F-CQ***$^{\neq}$*) is the set of* σ*-formulas (where* σ *is unary functional signature), of the form* $\varphi = \exists \bar{x} \psi$ *such that* ψ *is a conjunction of unary atoms* $U(v)$ *and equalities* $\tau = \tau'$ *(resp. equalities* $\tau = \tau'$ *and disequalities* $\tau \neq \tau'$*), where* τ, τ' *are functional terms of the form* $f(v)$ *or* v*, with* $v \in var(\varphi)$*.*

A natural graph based notion of acyclicity can be associated with such a query.

Definition 4 *To each* ***F-CQ*** *or* ***F-CQ***$^{\neq}$ *formula* φ*, we naturally associate the undirected graph* $G_\varphi = (V_\varphi, E_\varphi)$ *defined as follows:*

- $V_\varphi = var(\varphi)$ *and*
- *for any pair of distinct variables* v, v'*, set* $\{v, v'\} \in E_\varphi$ *iff an equality involving both* v *and* v' *occurs as a conjunct* [4] *in* φ*.*

[4] Here again, the disequalities of φ are not involved in the definition of G_φ

An ***F-CQ*** *(resp.* ***F-CQ***$^{\neq}$*) formula* φ *is* acyclic*, i.e., belongs to* ***F-ACQ*** *(resp.* ***F-ACQ***$^{\neq}$*) if its graph* G_φ *is acyclic.*

Without loss of generality we assume that G_φ is also connex and hence is a tree T called a *join-tree* of φ . The following results are easy to prove.

Lemma 3 *1. Preservation of acyclicity: Let* φ *be a* ***CQ***$^{\neq}$ *formula. If* φ *is acyclic then its functional translations 1 (in the case* φ *is quantifier-free) and 2 are acyclic too.*

2. Linearity of the transformations:

- *For formulas: The sizes of the transformed formulas 1 and 2 are linear in the size of* φ*.*
- *For structures: Our transformation of a relational* σ*-structure* $\mathcal{M}$ *into a functional* σ'*-structure* $\mathcal{M}'$ *is computable in time* $O(|\mathcal{M}|)$*.*

3. Correctness of the reductions:

- *Translation 1: Let* ψ' *be the quantifier-free translation (***F-ACQ***$^{\neq}$*) of a (quantifier-free) formula* ψ *(***ACQ***$^{\neq}$*). There exists an exact reduction from problem* Enum(ψ) *to* Enum(ψ')*.*
- *Translation 2: We have* $\varphi(\mathcal{M}) = \varphi''(\mathcal{M}')$*.*

Equal and disequal-trees. In this paragraph, we refine the notion of join-tree to take into account through labelling information about equality and disequality in formulas. This notion will be usefull from Section 3.

Definition 5 *The* equal-tree *of a* ***F-ACQ*** *or* ***F-ACQ***$^{\neq}$ *formula* φ *is its join-tree whose each edge* (x, y) *is labelled by the set of equality of conjuncts of* φ *of the form* $\tau(x) = \tau'(y)$*. The* disequal-tree *of a* ***F-ACQ***$^{\neq}$ *formula* φ *is its equal-tree that is completed by disequality links. More precisely, if* φ *contains* k *disequality conjuncts of the form* $\tau(x) \neq \tau'(y)$ *for two distinct variables* x *and* y *then* k *unlabelled disequality links* (x, y) *are added to* T*.*

Definition 6 *Let* T *be a disequal-tree,* y *be a leaf of* T *and* x *be the parent of* y *in* T*. The* y-elagated tree *of* T*, denoted* $T - \{y\}$*, is the disequal-tree* T *modified as follows:*

- *delete the node* y*, the labelled edge* (x, y) *and the disequality links* (x, y) *if there exists some;*
- *replace each disequality link* (y, z) *of* T*, for* $z \neq x$*, by one disequality link* (x, z)*.*

More generally, an elegated tree of T *is a disequal-tree* T' *constructed from* T *by iterated leaf elagations.*

Example. Let φ be the following **F-ACQ**$^{\neq}$ formula:

$$\varphi(x, y, z) \equiv f(x) = g(y) \wedge f'(x) = h(z) \wedge h(x) \neq f'(y) \wedge f(y) \neq g(z) \wedge g(y) \neq h(z).$$

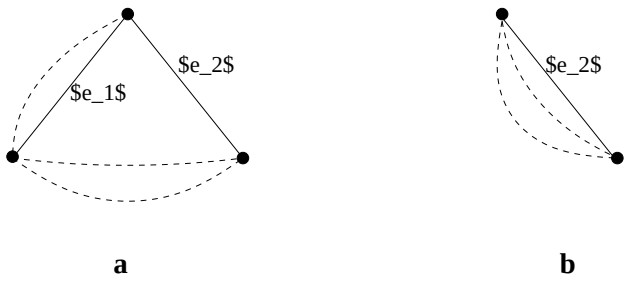

Figure 2: The three disequality links are represented by dashed lines

The disequal-tree of φ is represented by Figure 2.a. The labels are respectively the set of equalities $e_1 = \{f(x) = g(y)\}$ and $e_2 = \{f'(x) = h(z)\}$. The y-elagated tree $T - \{y\}$ is represented in Figure 2.b. The two disequality links (y, z) are replaced by two disequality links (x, z) in $T - \{y\}$.

⊣

Definition 7 *In a disequal-tree T, the* disequal degree *of a node x, denoted* $\text{disdeg}_T(x)$*, is the number of disequality links adjacent to x. The* disequal degree *of a subtree S of T, denoted* $\text{disdeg}_T(S)$ *is the sumof the disequal degrees of its nodes:*

$$\text{disdeg}_T(S) = \sum_{x \in S} \text{disdeg}_T(x).$$

Lemma 4 *Let k be the number of disequality links of the disequal-tree T.*

(i) For each subtree S of T, it holds $\text{disdeg}_T(S) \leq 2k$

(ii) For each elagated treeT' of T and each node x in T', it holds $\text{disdeg}_{T'}(x) \leq k$.

Proof -

(i) Trivial since each disequality link is adjacent to exactly 2 nodes.

(ii) Follows form the fact that each elagated tree $T - \{y\}$ has at most the same number of disequality links as T.

□

2 Covers and representative sets

In this section, we introduce and study some combinatorial notions which are our main tools for quantifier elimination in acyclic formulas. However, they may be interesting for their own. First, let us introduce some notations.
- In all the definitions and results of this section, E and F are two finite sets and $\bar{f} = (f_1, \ldots, f_k)$ is a tuple of k unary functions from E to F. In the following, $(E, \bar{f})$ is called a *table*.

- For any $i \leq k$ and any $a \in E$, let E_a^i denote the set $\{x \in E : f_i(x) \neq f_i(a)\}$.
- Let λ denote the empty k-tuple and let $\sqcup$ be a special symbol that do not belong to F.
- For any k-tuple $\bar{x} = (x_1, \ldots, x_k)$ and any $i \leq k$, let x_{-i} denote the $(k-1)$-tuple $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k)$.

We first define a notion of cover and of minimal cover.

Definition 8 *A* cover $\bar{c}$ *of a table* $(E, \bar{f})$ *is a tuple* $(c_1, \ldots, c_k) \in (F \cup \{\sqcup\})^k$ *such that, for all* $x \in E$*, there exists some* $i \leq k$*, such that* $c_i = f_i(x)$*. We denote by* $\textsc{covers}(E, \bar{f})$ *the set of covers of* $(E, \bar{f})$*.*

Covers can be partially ordered according to the following definition.

Definition 9 *A cover* $\bar{c}'$ *is* more general *than a cover* $\bar{c}$*, denoted* $\bar{c}' \leq \bar{c}$ *if, for all* $i \leq k$*, either* $c_i = c'_i$ *or* $c_i = \sqcup$*. A cover* $\bar{c}$ *of a table* $(E, \bar{f})$ *is* minimal *if this table has no more general cover.*

Definition 10 *A* complete cover set *of* $(E, \bar{f})$ *is a set* $\mathcal{C}$ *of covers of* $(E, \bar{f})$ *such that, for each cover* $\bar{c}$ *of* $(E, \bar{f})$*, there is some* $\bar{c}' \in \mathcal{C}$ *which is more general than* $\bar{c}$*. Moreover, if each element of* $\mathcal{C}$ *is a minimal cover of* $(E, \bar{f})$ *then* $\mathcal{C}$ *is called a* minimal complete cover set *of* $(E, \bar{f})$*.*

Lemma 5 *There exists one and only one minimal complete cover set of* $(E, \bar{f})$*. It is the set* $\textsc{min-covers}(E, \bar{f})$ *of minimal covers of* $(E, \bar{f})$*, also called the minimal cover set of* $(E, \bar{f})$*.*

Proof - For the existence it is sufficient to note that for each $\bar{c} \in \textsc{covers}(E, \bar{f})$, there is some $\bar{c}' \in \textsc{min-covers}(E, \bar{f})$ which is more general. For the unicity, let us consider $\mathcal{C}$ and $\mathcal{C}'$ two minimal complete cover sets of $(E, \bar{f})$. By completeness of $\mathcal{C}'$, there is, for each $\bar{c} \in \mathcal{C}$, some $\bar{c}' \in \mathcal{C}'$ such that $\bar{c}' \leq \bar{c}$. By minimality of $\mathcal{C}$, this imply $\bar{c} = \bar{c}'$. So we have proved that $\mathcal{C} \subseteq \mathcal{C}'$. Similarly, $\mathcal{C}' \subseteq \mathcal{C}$, hence $\mathcal{C} = \mathcal{C}'$ □

Example. Let $E = \{a, b, c, d, e\}$, $F = \{1, 2, 3, 4, 5\}$ and $\bar{f} = (f_1, f_2, f_3)$ be the following tuple of unary functions over E:

	f_1	f_2	f_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 $(\sqcup, 5, 4)$ are covers of $\bar{f}$ over $T(E, \bar{f})$ and form a complete cover set of $(E, \bar{f})$. The minimal cover set of $(E, \bar{f})$ is the set $\{(1, 2, 3), (3, 2, 1), (\sqcup, 5, 4)\}$. ⊣

The main notion used in the procedure of quantifier elimination is that of *representative set*. It is derived immediately form the notion of cover.

Definition 11 *A* representative set *of* $(E, \bar{f})$ *is a subset* $E' \subseteq E$ *such that* $\textsc{covers}(E, \bar{f}) = \textsc{covers}(E', \bar{f})$*.*

By abuse of notation, the restriction of $\bar{f}$ to E' in the above definition is also denoted $\bar{f}$. In Example 2, $E' = \{a, b, c, d\}$ is a representative set of $(E, \bar{f})$.
Covers and representative sets can be computed recursively by the following lemmas.

Lemma 6 (covers) *The following properties hold:*

(i) *For all* $\bar{c}$, $\bar{c} \in \text{COVERS}(\emptyset, \bar{f})$

(ii) *If* $E \neq \emptyset$ *then* $\text{COVERS}(E, \lambda) = \emptyset$

(iii) *Assume* $E \neq \emptyset$, $a \in E$ *and* $\bar{f} \neq \lambda$. *Then, the following equivalence holds for all* $\bar{c}$: $\bar{c} \in \text{COVERS}(E, \bar{f})$ *iff there is some* $i \leq k$ *such that* $c_i = f_i(a)$ *and* $\bar{c}_{-i} \in \text{COVERS}(E_i^a, \bar{f}_i)$.

Proof - Straightforward and left to the reader. □

Lemma 7 (representative sets) *The following properties hold:*

(i) *The empty set* $\emptyset$ *is a representative set of* $(\emptyset, \bar{f})$.

(ii) *If* $E \neq \emptyset$ *then* $a \in E$ *the n* $\{a\}$ *is a representative set of* (E, λ).

(iii) *Assume* $E \neq \emptyset$, $a \in E$ *and* $\bar{f} \neq \lambda$ *and, for all* $i \leq k$, R_i *is a representative set of* $E_i^a, \bar{f}_{-i}$. *Then,* $\{a\} \cup \bigcup_{i \leq k} R_i$ *is a representative set of* $(E, \bar{f})$.

Proof - Items (i) and (ii) are obvious. Let us prove (iii). By hypothesis, $a \in E$ and, for all $i \leq k$, $R_i \subseteq E_i^a$ and $\text{COVERS}(R_i, \bar{f}_{-i}) \subseteq \text{COVERS}(E_i^a, \bar{f}_{-i})$. We only have to prove that

$$\text{COVERS}(\{a\} \cup \bigcup_{i \leq k} R_i, \bar{f}) \subseteq \text{COVERS}(E, \bar{f}),$$

since the other implication is obvious. Let $\bar{c} \in \text{COVERS}(\{a\} \cup \bigcup_{i \leq k} R_i, \bar{f})$. In particular, there exsits some $j \leq k$ such that $f_j(a) = c_j$. Let b be any element of E. There are two possible cases:

- either $f_j(b) = f_j(a) = c_j$,
- or $f_j(b) \neq f_j(a) = c_j$. In that case $b \in E_j^a$. By hypothesis $\bar{c} \in \text{COVERS}(R_j, \bar{f})$ and $R_j \subseteq E_j^a$. Hence, for all $x \in R_j$, $f_j(x) \neq f_j(a) = c_j$. This yields $\bar{c} \in \text{COVERS}(R_j, \bar{f}_{-j})$ Hence $\bar{c} \in \text{COVERS}(E_j^a, \bar{f}_{-j})$ by hypothesis. Since $b \in E_j^a$ there is some $i \neq j$ such that $f_i(b) = c_i$.

Finally, we have proved that for each $b \in E$ there is some $i \leq k$ such that $f_i(b) = c_i$. That means $\bar{c} \in \text{COVERS}(E, \bar{f})$ are required. □

The following proposition expresses that the cardinality ofthe minimal cover set of any table $(E, \bar{f})$ is bounded by a number independant of $|E|$ and that $(E, \bar{f})$ has some representative set of cardinality similarly bounded. Moreoever the minimal cover set of $(E, \bar{f})$ and a "small" representative set of $(E, \bar{f})$ are efficiently computable.

Proposition 8 *The following properties hold:*

(i) $|\textsc{min-covers}(E, \bar{f})| \leq k!$.

(ii) There exists a function $h(k) = O(k!)$ *such that each table* $(E, \bar{f})$ *has some representative set* E' *of cardinality bounded by* $h(k)$ *called a* small representative set *of* $(E, \bar{f})$.

(iii) There is an algorithm which, for each table $(E, \bar{f})$ *as input, computes a complete cover set of* $(E, \bar{f})$ *of cardinality bounded by* $k!$ *and a small representative set of* $(E, \bar{f})$. *This algorithm works in time* $O(k! \cdot k \cdot |E|)$.

Proof - These properties are consequences of the preceding Lemmas 6 and 7 as we will show. Lemma 6 justifies the following algorithm which computes a complete cover set of any table $(E, \bar{f})$.

Algorithm 1 Complete-cover-set

```
1: Input: table (E, f̄) and cover c̄
2: if E = ∅ then
3:    output c̄
4: else if f̄ ≠ λ then
5:    let a ∈ E
6:    for i = 1, ..., k do
7:       c_i ← f_i(a)
8:       COVERS(E_i^a, f̄_{-i})
9:    end for
10: end if
```

The algorithm is first called with the input table $(E, \bar{f})$ and $\bar{c} = (\sqcup, \ldots, \sqcup)$. It is easy to see recursively that the complete cover set so computed contains at most $k!$ covers. This implies $|\textsc{min-covers}(E, \bar{f})| \leq k!$. As for representative sets, Lemma 7 justifies the following procedure which for any table $(E, \bar{f})$ returns some representative set of the table.

Algorithm 2 Rep-Set

```
1: Input: table (E, f̄)
2: if E = ∅ then
3:    return ∅
4: else
5:    Let a ∈ E
6:    return {a} ∪ ⋃_{i≤k} REP-SET(E_a^i, f̄_{-i})
7: end if
```

The cardinality of the representative set computed by the algorithm is bounded by the function $h(k)$ defined recursively as $h(0) = 1$ and, for $k \geq 1$, $h(k) = 1 + k \cdot h(k-1)$. It is easily seen that $h(k) = O(k!)$.
By using adapted linked data structures with pointers, it is not difficult to construct an implementation of the algorithms Complete-Cover-Set and Rep-Set that computes

each subset E_i^a in time $O(k \cdot |\{x \in E : f_i(x) = f_i(a)\}|) = O(k \cdot |E - E_i^a|)$ and hence globally runs in time $O(k \cdot k! \cdot |E|)$. This completes the proof of the proposition. □

3 Quantifier elimination

Lemma 9 *Let $\varphi(\bar{x})$ be an **F-ACQ**$^{\neq}$ σ-formula with a disequal-tree T and of the form $\varphi(\bar{x}) = \exists z \psi(\bar{x}, z)$ where ψ is quantifier-free and z is a leaf of T. Let k (resp. m) be the number of disequality (resp. equality) conjuncts in $\varphi(\bar{x})$ that involve the variable z. Let $\mathcal{M}$ be a σ-structure of domain D. Then we can compute in time $O(m \cdot |D| + k! \cdot k|D|)$ a quantifier-free σ'-formula $\varphi'(\bar{x})$ with $\sigma \subseteq \sigma'$ and a σ'-structure $\mathcal{M}'$ that expands $\mathcal{M}$ such that*

- *$\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$;*
- *$\varphi'(\bar{x})$ is a disjunction of $h = h(k) = O(k!)$, **F-ACQ**$^{\neq}$ formulas ψ_i of disequal-tree $T - \{z\}$ and hence $|\psi_i| \leq |\varphi|$.*

Proof - Formula $\varphi(\bar{x})$ can be put under the form:

$$\varphi(\bar{x}) \equiv \psi_0(\bar{x}) \wedge \exists z (P(z) \wedge \bigwedge_{1 \leq i \leq m} g_i(z) = g'_i(y) \wedge \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\bar{x}))$$

where $\psi_0(\bar{x})$ is a quantifier-free **F-ACQ**$^{\neq}$ formula of disequal-tree $T - \{z\}$, $y \in \bar{x}$ is the parent of the leaf $z \notin \bar{x}$ in the tree T, $f'_i(\bar{x})$ denotes a term $f'_i(v_i)$ for $v_i \in \bar{x}$ and $P(z)$ is a quantifier-free formula on z. Let $\bar{g}(x) = (g_1(x), \ldots, g_m(x))$, $\bar{g}'(x) = (g'_1(x), \ldots, g'_m(x))$ and similarly for $\bar{f}$ and $\bar{f}'$. One executes bucket-sort to partition the set $P = P(\mathcal{M})$ according to the values of $\bar{g}$ within time $O(m(|P| + |D|)) = O(m|D|)$; more precisely, for each $\bar{\alpha} \in \bar{g}(D)$, one defines the set $P_{\bar{\alpha}} = P \cap \bar{g}^{-1}(\bar{\alpha})$ and computes a small representative set of $(P_{\bar{\alpha}}, \bar{f})$ denoted $P'_{\bar{\alpha}}$ (of cardinality at most $h = h(k) = O(k!)$). By Lemma 7 this can be done globally in time $\Sigma_{\bar{\alpha} \in \bar{g}(D)} O(k! \cdot k \cdot |P_{\bar{\alpha}}|) = O(k! \cdot k \cdot |P|) = O(k! \cdot k \cdot |D|)$. Clearly, $\varphi(\bar{x})$ can be rephrased as

$$\psi_0(\bar{x}) \wedge \exists z \in P_{\bar{g}'(y)} \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\bar{x})$$

or, by definition of the covers, as $\psi_0(\bar{x}) \wedge \bar{f}'(\bar{x}) \notin \text{COVER}(P_{\bar{g}'(y)}, \bar{f})$. By definition of the representative sets, it is also equivalent to the following formula on $\mathcal{M}$:

$$\psi_0(\bar{x}) \wedge \exists z \in P'_{\bar{g}'(y)} \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\bar{x}).$$

Now, let us use the cardinality bound of the small representative set $P'_{\bar{g}'(y)} \leq h(k)$. Let $\mathcal{M}'$ be the σ'-expansion of $\mathcal{M}$ obtained by introducing the unary predicates E_j and the unary functions v_j and $f_{i,j}$, for $1 \leq i \leq k$ and $1 \leq j \leq h = h(k)$, defined as follows:

- $y \in E_j \Leftrightarrow |P'_{\bar{g}'(y)}| \geq j$;
- $v_j(y)$ is the j^{th} element of $P'_{\bar{g}'(y)}$ if $|P'_{\bar{g}'(y)}| \geq j$; otherwise it is an arbitrary value ;
- set $f_{i,j} = f_i \circ v_j$.

This yields $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$ for the σ'-formula $\varphi'(\bar{x}) \equiv \bigvee_{1 \leq j \leq h} \psi_j(\bar{x})$ where

$$\psi_j(\bar{x}) \equiv \psi_0(\bar{x}) \wedge E_j(y) \wedge \bigwedge_{1 \leq i \leq k} f_{i,j}(y) \neq f'_i(\bar{x}).$$

Notice that $|\varphi'| \leq h(k) \cdot |\varphi| = O(k! \cdot |\varphi|)$. Finally, summing up the arguments, the whole time complexity of the transformation $(\varphi, \mathcal{M}) \mapsto (\varphi', \mathcal{M}')$ is $O(m \cdot |D| + k! \cdot k \cdot |D| + k! \cdot |\varphi|)$ as required.

□

Proposition 10 *Let $\varphi(x)$ be an **F-ACQ**$^{\neq}$ σ-formula with only one free variable x. Let $\mathcal{M}$ be a σ-structure. One can compute $\varphi(\mathcal{M})$ in time $O(f(\varphi) \cdot |\mathcal{M}|)$ for some computable function f.*

3.1 Improving the constant size

In this section, we will make precise the value of the constant size in the elimination process and in consequence improve the value of the function f of Proposition 10. In order to reach this goal, we will need to use the notion of equal and disequal tree more deeply and to adopt different strategies depending on properties of the variables to eliminate in formulas.
The following lemma 11 is a variant of lemma 9. It is convenient to assume that each variable x of the studied **F-ACQ**$^{\neq}$ σ-formula φ has its own domain D_x. This convention will allow to duplicate the domain D_x of x without duplicating the domains of the other variables. Our definition of a σ-model $\mathcal{M}$ is modified accordingly. More precisely, henceforth we assume, w.l.g., that for each **F-ACQ**$^{\neq}$ σ-formula φ and each σ-structure $\mathcal{M}$, the following conditions hold:

1. each (unary) function symbol f (resp. predicate symbol U) of σ occurs only once in φ;
2. if $f(x)$ (resp. U(x)) is the unique term (resp. atom) that involves f (resp. U) in φ, then f (resp. U) is called a x-function symbol (resp. x-predicate symbol) or, for short, x-symbol, and the interpretation of the function symbol f (resp. predicate symbol U) in the structure $\mathcal{M}$ is a function $f : D_x \to D$ (subset $U \subseteq D_x$) where D is the domain of $\mathcal{M}$ that is the union $\bigcup_{x \in var(\varphi)} D_x$ of the domains of the variables of φ.

We denote by σ_x the set of x-function symbols and x-predicate symbols of σ. Clearly, σ is the disjoint union of the subsets σ_x, for all $x \in var(\varphi)$.

Lemma 11 *Let $\varphi(\bar{x})$ be an **F-ACQ**$^{\neq}$ σ-formula with a disequal-tree T and of the form $\varphi(\bar{x}) = \exists y \exists z \psi(\bar{x}, y, z)$ where ψ is quantifier-free, z is a leaf of T and y is its parent, $y, z \notin \bar{x}$. Let k (resp. m) be the number of disequality (resp. equality) conjuncts in ψ that involve variable z. For each σ-structure $\mathcal{M}$, one can compute a quantifier-free σ'-formula $\psi'(\bar{x}, y)$ with $\sigma \subseteq \sigma'$ and a σ'-structure $\mathcal{M}'$ so that the following conditions hold:*

- $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$ *where* $\varphi'(\bar{x}) = \exists y \psi'(\bar{x}, y)$;

- *$\psi'(\bar{x}, y)$ is an **F-ACQ**$^{\neq}$ formula of disequal-tree $T - \{z\}$ and hence $|\psi'| \leq |\psi|$;*
- *$card(\sigma') = card(\sigma) + k$; more precisely, $card(\sigma'_y) = card(\sigma_y) + k$;*
- *in the transformation $\mathcal{M} \mapsto \mathcal{M}'$, the domains of the variables are not modified except the domain of y whose cardinality is multiplied by $h(k) = O(k!)$;*
- *moreover, the transformation $(\sigma, \varphi, \mathcal{M}) \mapsto (\sigma', \varphi', \mathcal{M}')$ is computed in time $O(m.card(D_z) + k!.k(card(D_y) + card(D_z)) + k!card(\sigma_y)card(D_y))$ where D_y and D_z are the respective domains of y and z in $\mathcal{M}$ and σ_y is the set of y-symbols of σ.*

Proof - We describe the differences with the proof of Lemma 9. The first part of the proof is similar with the formula $\psi(\bar{x}, y, z)$ of the form

$$\psi(\bar{x}, y, z) \equiv \psi_0(\bar{x}, y) \wedge P(z) \wedge \bigwedge_{1 \leq i \leq m} g_i(z) = g'_i(y) \wedge \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\bar{x}, y)$$

where $\psi_0(\bar{x}, y)$ and $P(z)$ are quantifier-free formulas and $f'_i(\bar{x}, y)$ means $f'_i(v_i)$ for some $v_i \in \{\bar{x}, y\}$. Similarly, $\exists z \psi(\bar{x}, y, z)$ is equivalent to the following formula on $\mathcal{M}$:

$$\psi_0(\bar{x}, y) \wedge \exists z \in P'_{\bar{g}'(y)} \bigwedge_{1 \leq i \leq k} f_i(z) \neq f'_i(\bar{x}, y).$$

Recall that $P'_{\bar{g}'(y)}$ is a small representative set of $(P_{\bar{g}'(y)}, \bar{f})$ where $P_{\bar{g}'(y)} = P \cap \bar{g}^{-1}(\bar{g}'(y))$. Let $\mathcal{M}'$ denote the σ'-structure, $\sigma' \supseteq \sigma$, constructed from $\mathcal{M}$ as follows:

- first, replace the domain D_y of variable y by the following domain

  $$D'_y = \{(b, j) \in D_y \times \{1, \ldots, h(k)\} : card(P'_{\bar{g}'(b)}) \geq j\};$$

  note that $|D'_y| \leq h(k) \cdot |D'_y| = O(k! \cdot |D_y|))$;
- second, define $\sigma' = \sigma \cup \{f_i" : 1 \leq i \leq k\}$ where the f_i"'s are k new unary y-function symbols and define their interpretations as follows on domain D'_y: for all $(b, j) \in D'_y$, $f_i"((b, j)) = f_i(u)$ where u is the j^{th} element of $P'_{\bar{g}'(b)}$;
- third, for each y-function symbol f (resp. y-predicate symbol U) of σ and each $(b, j) \in D'_y$, set $f^{\mathcal{M}'}((b, j)) = f^{\mathcal{M}}(b)$ and $U^{\mathcal{M}'}((b, j)) \Leftrightarrow U^{\mathcal{M}}(b)$.

Let ψ' denote the following formula

$$\psi'(\bar{x}, y) \equiv \psi_0(\bar{x}, y) \wedge \bigwedge_{1 \leq i \leq k} f''_i(y) \neq f'_i(\bar{x}, y).$$

Notice that the disequal-tree of ψ' is the elagated tree $T - z$ of the disequal-tree T of ψ. One similarly concludes $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$ for $\varphi(\bar{x}) \equiv \exists z \exists y \psi(\bar{x}, y, z)$ and $\varphi'(\bar{x}) \equiv \exists y \psi'(\bar{x}, y)$. The only new point is the equivalence of the "prefixes" $\exists b \in D_y \bigvee_{1 \leq j \leq h(k)}$ and $\exists (b, j) \in D'_y$.
Finally, the whole time required by the transformation $(\sigma, \varphi, \mathcal{M}) \mapsto (\sigma', \varphi', \mathcal{M}')$ is clearly $O(m \cdot |D_z| + k! \cdot k \cdot (|D_y| + |D_z|) + k! \cdot |\sigma_y| \cdot |D_y|)$. This achieves the proof.

□

Using the result above, one can eliminate one by one all the variables but one in a bottom-up approach and derive the two following results. The iterative treatment of variables is similar in spirit to Yannakakis' algorithm [18].

Proposition 12 *Let $\varphi(x)$ be an* ***F-ACQ***$^{\neq}$ *σ-formula with only one free variable x. Let $\mathcal{M}$ be a σ-structure. Let k be the number of disequalities of $\varphi(x)$. One can compute $\varphi(\mathcal{M})$ in time $O((2k)!^2 \cdot |\varphi| \cdot |\mathcal{M}|)$, hence in time $O(2^{k \log k} \cdot |\varphi| \cdot |\mathcal{M}|)$.*

Proof - Let T be the disequal-tree of φ. We can assume, w.l.o.g., that x is the root of T and φ is of the form

$$\varphi(x) \equiv \exists \bar{y} \exists \bar{z} \Psi(x, \bar{y}, \bar{z})$$

$\bar{y} = (y_1, \ldots, y_p)$ is the set of children of x in T. The algorithm that computes $\varphi(\mathcal{M})$ runs in three steps.

- Step 1 eliminates successively the variables $\bar{z} = z_1, \ldots, z_q$ in some T-order by using iteratively Lemma 11
- Step 2 eliminates successively the children of x, i.e. variables $y_1, \ldots, y_p$ which are the leaves of $T - \bar{z}$, by using Lemma 9.
- Finally, Step 3 computes $\varphi(\mathcal{M})$.

We describe and analyse successively these 3 steps.

Step 1. The leaves $z_q, \ldots, z_2, z_1$ are eliminated one after the other by iterating the algorithm of Lemma 11. The initial σ-structure $\mathcal{M}$ is transformed into successive σ'-structures $\mathcal{M}'$ by extending the signature and duplicating the domains of the variables, except the domain of the root x which is never modified. The following Claim describes how, for any variable v (distinct of x), the number of v-symbols in the signature and the cardinality of the domain of v can increase in the transformation $(\sigma, \varphi, \mathcal{M}) \mapsto (\sigma', \varphi', \mathcal{M}')$

Claim 1 *Let $v \in \bar{y} \cup \bar{z}$ of subtree T_v in T. We have, at any moment of Step 1, the following upper bounds:*

$$|\sigma'_v| \leq |\sigma_v| + \mathrm{disdeg}_T(T_v), \text{ and, } |D'_v| = O(|D_v| \cdot (\mathrm{disdeg}_T(T_v))!)$$

where D_v (resp. D'_v) is the domain of v in $\mathcal{M}$ (resp. $\mathcal{M}'$) and σ_v (resp. σ'_v) is the set of v-symbols in σ (resp. σ')

Proof of the claim - Let $u_1, u_2, \ldots, u_r \in \bar{z}$ denotes the list of the children of v in T. By Lemma 11, when variable u_i is eliminated, the cardinality of the signature associated to its parent v is increased by k_i where k_i is the number of disequality links adjacent to u_i; similarly, the cardinality of the domain of v is multiplied by $h(k_i) = O(k_i!)$. Obviously, we have $k_i \leq d_i$ for $d_i = \mathrm{disdeg}_T(T_{u_i})$ where T_{u_i} is the subtree of root u_i in T. This implies:

$$|\sigma'_v| \leq |\sigma_v| + \sum_{1 \leq i \leq r} d_i, \text{, and, } |D'_v| = O(|D_v|. \prod_{1 \leq i \leq r} d_i!)$$

Notice the unequalities:

$$\prod_{1 \leq i \leq r} d_i! \leq (\sum_{1 \leq i \leq r} d_i)!, \text{ and, } \sum_{1 \leq i \leq r} d_i = \sum_{1 \leq i \leq r} \mathrm{disdeg}_T(T_{u_i}) \leq \mathrm{disdeg}_T(T_v)$$

This proves the claim. □

Let us now analyze the time complexity, denoted $\mathsf{Time}(u)$, of the elimination of some variable u of parent v in T during Step 1. By Lemma 11, it is

$$\mathsf{Time}(u) = O(m_u \cdot |D'_u| + k_u! \cdot k_u \cdot (|D'_u| + |D'_v|) + k_u! \cdot |\sigma'_v| \cdot |D'_v|),$$

where m_u is the number of equality conjuncts involving u (and v) in φ (and in φ') and k_u is the number of disequality conjuncts involving u in φ'. Notice $k_u \leq k$.
By the claim, we have $|D'_u| = O(|D_u| \cdot d_u!)$ and $|D'_v| = O(|D_v| \cdot d_v!)$ where $d_u = \mathrm{disdeg}_T(T_u)$ and $d_v = \mathrm{disdeg}_T(T_v)$; by this claim, we also have $|\sigma'_v| \leq |\sigma_v| + d_v$.
By Lemma 4, we have $d_u \leq 2k$ and $d_v \leq 2k$. Hence $|\sigma'_v| \leq |\sigma_v| + 2k$, $|D'_v| = O(|D| \cdot (2k)!)$ and $|D'_u| = O(|D| \cdot (2k)!)$. So, $\mathsf{Time}(u) = O((m_u + k! \cdot k + k! \cdot |\sigma|) \cdot (2k)! \cdot |D|)$. Notice the inequalities $\sum_{u \in \bar{z}} m_u \leq m$ and $|\sigma| \cdot |D| \leq |\mathcal{M}|$ and $|\bar{z}| \leq |\varphi|$.
Finally, the total time of Step 1 is:

$$\begin{array}{rcl} \sum_{u \in \bar{z}} \mathsf{Time}(u) = & O((2k)!(m \cdot |D| + |\varphi|(k! \cdot k \cdot |D| + k! \cdot |\mathcal{M}|))) \\ = & O(k! \cdot k \cdot (2k)! \cdot |\varphi| \cdot |\mathcal{M}|) \end{array}$$

Step 2. Recall that at the end of Step 1 the σ'-formula φ' obtained has a disequal-tree $T - \bar{z}$ of depth 1 that consists of the root x and its children $y_1, \ldots, y_p$; the corresponding σ'-structure $\mathcal{M}'$ has domain $D'_x = D_x = D$, and, for $i = 1, \ldots, p$, $|D'_{y_i}| = |D_{y_i} \cdot d_i!|$ where $d_i = \mathrm{disdeg}_T(T_{y_i})$.
One successively eliminates each variable y_i, $i = 1, \ldots, p$, by performing iteratively the algorithm of Lemma 9. Let $k_i = \mathrm{disdeg}_{T-\bar{z}}(y_i)$. We observe $\sum_{1 \leq i \leq p} k_i \leq 2k$. By Lemma 11, after the elimination of $y_1, \ldots, y_{i-1}$ $(1 \leq i \leq p)$, the formula φ has been transformed into a disjunction of $O(k_1! \cdot k_2! \cdots k_{i-1}!)$ **F-ACQ**$^{\neq}$ formulas, each of disequal-tree $T - \{\bar{z}, y_1, y_2, \ldots, y_{i-1}\}$. So, by Lemma 9, the time required to eliminate y_i in all those **F-ACQ**$^{\neq}$ disjuncts is:

$$\mathsf{Time}(y_i) = O(k_1! \cdot k_2! \cdots k_{i-1}! \cdot (m_i \cdot |D'_{y_i}| + k_i! \cdot k_i(|D'_{y_i}| + |D_x| + k_i! \cdot |\varphi|))).$$

Recall the inequalities:

$$\prod_{1 \leq i \leq r} k_i! \leq (\sum_{1 \leq i \leq p} k_i)! \leq (2k)!, \text{ and, } |D'_{y_i}| \leq (2k)! \cdot |D|.$$

Hence,

$$\mathsf{Time}(y_i) = O(((2k)! \cdot m_i + (2k)! \cdot k_i + |\varphi|) \cdot (2k)! \cdot |\mathcal{M}|).$$

By noticing that $\sum_{1 \leq i \leq p} (m_i + l_i) = O(|\varphi|)$. So, the total time of the elimination of $y_1, \ldots, y_p$ is

$$\sum_{1 \leq i \leq p} \mathsf{Time}(y_i) = O((2k)!^2 \cdot |\varphi| \cdot |\mathcal{M}|).$$

Step 3. At the end of Step 2, one obtains a disjunction $\varphi''(x) \equiv \bigvee_{1 \leq i \leq K} \Psi_i(x)$ of $K = O(\prod_{1 \leq i \leq p} k_i!)$ quantifier-free **F-ACQ**$^{\neq}$ σ''-formulas $\Psi_i(x)$ with only one variable x and a σ''-structure $\mathcal{M}''$ of domain $D_x = D$ so that $\varphi(\mathcal{M}) = \varphi''(\mathcal{M}'') = \bigcup_{1 \leq i \leq K} \Psi_i(\mathcal{M}'')$.

Clearly, $K = O((\sum_{1\leq i\leq p} k_i)!) = O((2k)!)$ and each $\Psi_i(\mathcal{M}'')$ can be computed in time $O(|\Psi_i| \cdot |D|) = O(|D|)$. So $\varphi = \bigcup_{1\leq i\leq k} \Psi_i(\mathcal{M}'')$ is computed in time $O((2k)! \cdot |D|) = O((2k)! \cdot |M|)$.
Finally, the whole time of computing $\varphi(\mathcal{M})$ by Steps 1, 2 and 3 is $O((2k!)^2 \cdot |\varphi| \cdot |\mathcal{M}|)$.
This completes the proof of the proposition. □

4 Linear delay enumeration

The following theorem is an easy consequence of Lemma 12

Theorem 13 *Let φ be a fixed **F-ACQ**$^{\neq}$ (resp. **ACQ**$^{\neq}$) query over a structure $\mathcal{M}$. Then* ENUM(φ) *can be evaluated with delay $O(|\mathcal{M}|)$. More precisely, the delay is $O(m \cdot |D| + k! \cdot k \cdot |D| + k! \cdot |\varphi|)$ where $|D|$ is the size of structure $\mathcal{M}$ and m (resp. k) is the number of equality (resp. disequality) conjuncts of φ. In particular, if φ is a **F-ACQ** (resp. **ACQ**) query then, the delay is $O(|\varphi| \cdot |\mathcal{M}|)$.*

Proof - We give an algorithm for **F-ACQ**$^{\neq}$ formulas. The same result for **ACQ**$^{\neq}$ formulas is deduced by using exact reductions. Let $\mathcal{M}$ be a functional σ-structure and $\varphi(x_1, \ldots, x_p)$ be a σ-query in **F-ACQ**$^{\neq}$.
The simple (recursive) algorithm returns all the satisfying tuples of $\varphi(x_1, \ldots, x_k)$.

Algorithm 3 ENUM($\varphi(x_1, \ldots, x_p), \mathcal{M}$)

```
1: if k = 1 then
2:    for a ∈ φ(M) do
3:       output a
4:    end for
5: else
6:    let φ'_1(x_1) ≡ ∃x_2 ... ∃x_p φ(x_1, ... x_p)
7:    for a ∈ φ'_1(M') do
8:       let φ_a ≡ φ(a, x_2, ..., x_p)
9:       for b̄ ∈ ENUM(φ_a(x_2, ..., x_p), M) do
10:         output (a, b̄)
11:      end for
12:   end for
13: end if
```

Clearly, the delay for any output $(a_1, \ldots, a_p) \in \varphi(\mathcal{M})$ is $O(\sum_{1\leq i\leq p} \mathsf{Time}(a_1, \ldots, a_{i-1}))$ where $\mathsf{Time}(a_1, \ldots, a_{i-1})$ denotes the time of the transformation:

$$(\varphi_{a_1,\ldots,a_{i-1}}, \mathcal{M}) \mapsto (\varphi_{a_1,\ldots,a'_{i-1}}, \mathcal{M}_{a_1,\ldots,a'_{i-1}})$$

with
The case $p = 1$ can easily be done with delay $O(|\varphi| \cdot |D|) = O((m + k) \cdot |D|)$. □

5 Constant delay enumeration

5.1 Combinatorial tools

In this section, we will need a convenient notion of *two-phase algorithm* for a problem with two inputs : a *static input* x and a *dynamic input* y. It is defined by the following general scheme.

- *Static phase (or precomputation phase)*: it computes some data called $r(x)$ (only depending on x);
- *Dynamic phase*: from the input $(r(x), y)$, it computes the output of the problem.

Notice the following two points:

- The precomputation phase is independent of the second (dynamic) input y;
- As a consequence, the dynamic phase can be iterated for different dynamic inputs y without having to iterate the static phase if the static input x does not change.

LeastNonCover

Static input :	Two finite ordered sets E and F, a k-tuple of unary functions $\bar{f} = (f_1, \dots f_k)$ from E to F
Dynamic input :	a vector $\bar{u} \in F^k$, an element $x \in E$
Parameter:	k
Output:	the least element $y \in E$ such that $y \geq x \wedge \bigwedge_{1 \leq i \leq k} f_i(y) \neq u_i$ if such an element exists, and $\perp$ otherwise.

Lemma 14 LeastNonCover *is computable with a static phase of linear time and a dynamic phase of constant time.*

Proof - In this proof, E, F and $\bar{f}$ are considered as implicit and we assume that $E = \{1, \dots, |E|\}$. Let x be an element of E, $\bar{u}$ be a k-tuple of elements of F and S be a subset of $[1, k]$. Then define $\text{Find}(x, \bar{u}, S)$ as the least element $y \in E$ if it exists ($\perp$ otherwise) such that $y \geq x \wedge \bigwedge_{i \in S} f_i(y) \neq u_i$. For each list $L = (a_1, \dots, a_l)$ of distinct elements of $[1, k]$, $0 \leq l \leq k$, we define the "pointer function" $\text{Ptr}_L : E \to E \cup \{\perp\}$ useful in the computation of Find as follows

- $\text{Ptr}_\epsilon(x) = x$;
- if $l \geq 1$ then $\text{Ptr}_{a_1,\dots,a_l}(x)$ is the least element $y \in E$ such that $y \geq x \wedge \forall j, 1 \leq j \leq l : f_{a_j}(y) \neq f_{a_j}(\text{Ptr}_{a_1,\dots a_{j-1}}(x))$ if it exists and $\text{Ptr}_{a_1,\dots a_{l-1}}(x) \neq \perp$, and is $\perp$ otherwise.

We assume that during the precomputation phase, we have computed the table PTR where $\text{PTR}_L[x] = \text{Ptr}_L(x)$ for each element $x \in E$ and each list L. We give an algorithm $\text{FIND}(x, \bar{u}, S)$ which computes $\text{Find}(x, \bar{u}, S)$. Clearly, the dynamic phase of LeastNonCover consists in calling FIND with $S = [1, k]$. $\text{FIND}(x, \bar{u}, S)$ consists of the call $\text{FIND-aux}(x, \bar{u}, S, \epsilon)$.
Finally, Algorithm 5 is given which computes the PTR table for the precomputation phase.

Algorithm 4 FIND-aux$(x, \bar{u}, S, L)$

$y \leftarrow \mathrm{PTR}_L[x]$
if $y = \perp \vee \forall i \in S - L : f_i(y) \neq u_i$ **then**
  return y
else
  choose $i \in S - L$ such that $f_i(y) = u_i$
  return FIND-aux$(x, \bar{u}, S, L.i)$
end if

Algorithm 5 Precomputation phase

for x from $|E|$ to 1 **do**
  $\mathrm{PTR}_\epsilon[x] \leftarrow x$
  for l from 1 to k **do**
    for each list $L = (a_1, \ldots, a_l)$ of l distinct elements of $[1, k]$ **do**
      if $x = |E|$ or $\mathrm{PTR}_{a_1,\ldots,a_{l-1}}[x] = \perp$ **then**
        $\mathrm{PTR}_L[x] \leftarrow \perp$
      else
        let $\bar{u} = (u_1, \ldots, u_k)$ such that $\forall j \leq l : u_{a_j} = f_{a_j}(\mathrm{PTR}_{a_1,\ldots,a_{j-1}}[x])$
        $\mathrm{PTR}_L[x] \leftarrow \mathrm{FIND}(x + 1, \bar{u}, \{a_1, \ldots, a_l\})$
      end if
    end for
  end for
end for

We easily see that the algorithm runs in k steps and thus in constant time as k is fixed. The correction of the algorithm can be established by proving by induction on the structure of L that for any call of the procedure FIND-aux, with $L = (a_1, \ldots, a_l)$, inside the execution of FIND, the following properties hold:

- for any $i \in [1, l]$, $f_{a_i}(\mathrm{PTR}_{a_1,\ldots,a_{i-1}}(x)) = u_{a_i}$
- $\mathrm{PTR}_L[x]$ is the least element $y \in E$ such that $y \geq x$ and for any $i \in L, f_i(y) \neq u_i$
- FIND-aux$(x, \bar{u}, S, L) = \mathrm{Find}(x, \bar{u}, S)$

Notice that the precomputation phase uses the function FIND whereas FIND uses the PTR table. In fact all the elements of PTR needed by a call of FIND have been previously computed. Thus the algorithm runs in time $O(|E|)$. It is easily seen by definition of Find and Ptr that the algorithm is correct. □

The more elaborated problem below need to be defined.

EnumNonCover

Static input:	Two finite ordered set E and F
	a k-tuple of unary functions $\bar{f} = (f_1, \ldots f_k)$ from E to F
Dynamic input:	a vector $\bar{u} \in F^k$
Parameter:	k
Output:	the set $\{x \in E : \bigwedge_{1 \leq i \leq k} f_i(x) \neq u_i\}$ in increasing order

The following lemma is a consequence of Lemma 14

Lemma 15 EnumNonCover *is computable with a static phase in linear time and a dynamic phase with a constant delay.*

5.2 Enumeration of quantifier-free and free-connex acyclic queries

Definition 12 *Let φ be a quantifier-free* ***F-ACQ***$^{\neq}$ *formula and T a join-tree of φ. An elimination order (or T-order) of the variables of φ is an ordered list (may be empty) $x_1, \ldots, x_p$ of its p variables such that (if $p > 0$) x_p is a leaf of T and $x_1, \ldots, x_{p-1}$ is an elimination order of the tree $T - \{x_p\}$.*

In the following, we implicity assume that the variables of any quantifier-free **F-ACQ**$^{\neq}$ formula are numbered in some fixed T-order.

Theorem 16 *Let $\varphi(x_1, \ldots, x_p)$ be a quantifier-free σ-formula in* ***F-ACQ***$^{\neq}$*. Then we have* $\textsc{Enum}(\varphi, <_{lex}) \in \textsc{Constant-Delay}_{lin}$ *for the lexicographical order $<_{lex}$.*

Proof - The theorem is proved by induction on the number of variables of φ. Without loss of generality, assume that φ is of the form $\varphi(\bar{x}, y) \equiv \varphi_0(\bar{x}) \wedge \Theta(\bar{x}, y)$ where

$$\Theta(\bar{x}, y) \equiv P(y) \wedge \bigwedge_{1 \leq i \leq m} f_i(y) = g_i(z) \wedge \bigwedge_{1 \leq j \leq k} h_j(y) \neq h'_j(\bar{x}).$$

In this formula, the variables y and $z \in \bar{x} = (x_1, \ldots, x_{p-1})$ are respectively the leaf x_p and its parent in the join-tree T of φ and $h'_j(\bar{x})$ denotes a term $h'_j(v)$ for some variable $v \in \bar{x}$. Trivially, φ is logically equivalent to the following formula $\varphi_1(\bar{x}, y) \equiv \psi(\bar{x}) \wedge \Theta(\bar{x}, y)$ where $\psi(\bar{x}) \equiv \exists y \varphi(\bar{x}, y)$. By Lemma 9, one can compute in linear time, for each σ-structure $\mathcal{M}$, a new σ'-structure $\mathcal{M}'$ (σ'-expansion of $\mathcal{M}$, $\sigma \subseteq \sigma'$) and a quantifier-free disjunction $\psi'(\bar{x}) \equiv \bigvee_{1 \leq i \leq N} \psi_i(\bar{x})$ of **ACQ**$^{\neq}$ σ'-formulas ψ_i (each of join-tree $T - \{y\}$) so that we have $\psi(\mathcal{M}) = \psi'(\mathcal{M}')$. This yields $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$ for the following quantifier-free formula: $\varphi'(\bar{x}, y) \equiv \psi'(\bar{x}) \wedge \Theta(\bar{x}, y)$. φ' is equivalent to the disjunction $\bigvee_{1 \leq i \leq N} \varphi_i(\bar{x}, y)$ where φ_i is the formula

$$\varphi_i(\bar{x}, y) \equiv \psi_i(\bar{x}) \wedge P(y) \wedge \bar{f}(y) = \bar{g}(z) \wedge \bigwedge_{1 \leq j \leq k} h_j(y) \neq h'_j(\bar{x})$$

By the induction hypothesis, we have for each $i \leq N$, $\textsc{Enum}(\psi_i, <_{lex}) \in \textsc{Constant-Delay}_{lin}$. By Lemma 2, it is sufficient to prove the same complexity result for $\textsc{Enum}(\varphi_i, <_{lex}), 1 \leq i \leq N$ (Notice the essential fact that the linear order $<_{lex}$ is the same for each i). Algorithms 6 and 7 below describe the precomputation and the enumeration phases for problem $\textsc{Enum}(\varphi_i, <_{lex})$.

Correctness of the algorithm: Clear, by the structure of the formula φ_i and by the specification of algorithm EnumNonCover.

Complexity of the algorithm: We easily check that each one of the five points of the precomputation phase is performed in time $O(|\mathcal{M}'|)$. The reader can also easily check that enumeration phase is constant delay. The following *extension property* is essential for that: by construction of ψ, ψ' and ψ_i, each solution tuple $\bar{a} \in \psi_i(\mathcal{M}')$ can be extended into (at least) one solution $(\bar{a}, b) \in \varphi_i(\mathcal{M}')$. So, $\textsc{Enum}(\varphi_i, <_{lex}) \in \textsc{Constant-Delay}_{lin}$ as required. This completes the inductive proof of the Theorem. □

Algorithm 6 Precomputation phase for $\textsc{Enum}(\varphi_i, <_{lex})$

Input: A σ'-structure $\mathcal{M}'$
Perform the precomputation phase of $\textsc{Enum}(\psi_i, <_{lex})$
$P \leftarrow \{y \in D : \mathcal{M}' \models P(y)\}$
Sort the set P according to the values $\bar{f}(y)$ for $y \in P$
Partition P into sets $P_{\bar{\alpha}} = \{y \in P : \bar{f}(y) = \bar{\alpha}\}$
$A \leftarrow \{\bar{\alpha} \in D^m : P_{\bar{\alpha}} \neq \emptyset\}$
for $\bar{\alpha} \in A$ **do**
    Perform the precomputation phase for EnumNonCover on input $E = P_{\bar{\alpha}}$ and $(h_j)_{j \leq k}$
end for

Algorithm 7 Enumeration phase for $\textsc{Enum}(\varphi_i, <_{lex})$

for $\bar{a} \in \textsc{Enum}(\psi_i, <_{lex})$ **do**
    $\bar{\alpha} \leftarrow \bar{g}(c)$ where c is the value of variable z in the tuple $\bar{a}$ (if z is the variable x_l, then $c = a_l$)
    $\bar{u} \leftarrow (h'_j(\bar{a}))_{j \leq k}$
    for $b \in \textsc{EnumNonCover}(P_{\bar{\alpha}}, (h_j)_{j \leq k}, \bar{u})$ **do**
        Output $(\bar{a}, b)$
    end for
end for

Definition 13 *A query of* ***F-ACQ*** *(resp* ***F-ACQ***$^{\neq}$*) is* free-connex acyclic *if φ is acyclic and the set of free variables of φ is a connex subset of the join-tree of φ. We denote by* ***F-CCQ*** *(resp* ***F-CCQ***$^{\neq}$*) the class of free-connex acyclic queries of* ***F-ACQ*** *(resp* ***F-ACQ***$^{\neq}$*).*

Since free variables form a connex part of the tree decomposition, a repeated use of Lemma 9 permits to eliminate one by one all the quantified variables until only the free variables remain. Then, one applies Theorem 16 to obtain the following result.

Theorem 17 *For any φ of* ***F-CCQ***$^{\neq}$*,* $\textsc{Enum}(\varphi) \in \textsc{Constant-Delay}_{lin}$.

6 A dichotomy result

In this section, we will give a precise characterization of acyclic conjunctive queries that can be enumerated with constant delay and of those that can not be. To obtain this characterization, we need to define a subclass of acyclic hypergraph (hence of conjunctive queries) based on properties satisfied by some specified sets of vertices.

6.1 S-connectivity and free-connex queries

Definition 14 *Let H be a hypergraph, we say that H' is a* inclusive extension *of H if*

- $V(H) = V(H')$,

Figure 3:

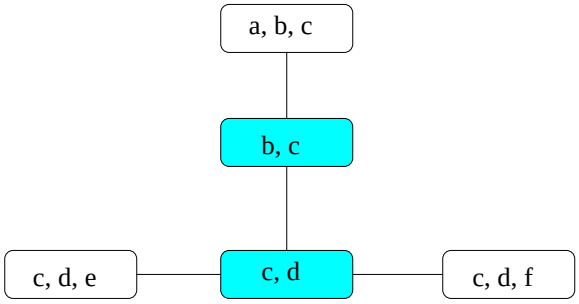

Figure 4: A $\{b, c, d\}$-connex-acyclic hypergraph

- $E(H) \subseteq E(H')$,
- $\forall e' \in E(H')\ \exists e \in E(H)\ e' \subseteq e$.

Definition 15 *Let H be a hypergraph, $S \subseteq V(H)$ and $A \subseteq E(H)$.*

- *(T, A) is an* S-connex tree-structure *of H if T is a tree-structure of H, and A is a connex subset of the set of vertices of T such that $\bigcup_{e \in A} e = S$.*
- *H is* S-connex acyclic *if there is an S-connex tree-structure (T, A) of H.*
- *H is* S-connex by inclusive extension*,* ext-S-connex acyclic *for short, if there is a inclusive extension H' of H such that H' is S-connex acyclic.*
- *A conjunctive formula φ is* free-connex acyclic *if the hypergraph associated to φ is ext-S-connex acyclic where S is the set free(φ) of free variables of φ.*

  *We denote by **CCQ** (resp **CCQ**$^{\neq}$) the class of free-connex acyclic queries φ of **CQ** (resp **CQ**$^{\neq}$).*

Example. Let $H = (\{a, b, c, d, e, f\}, \{\{a, b, c\}, \{b, c\}, \{c, d\}, \{c, d, e\}, \{c, d, f\}\})$. H is $\{b, c, d\}$-connex acyclic. It admits a (T, A) tree structre with T depicted in Figure 4 and $A = \{\{b, c\}, \{c, d\}\}$. The hypergraph H can be seen as an inclusive extension of the hypergraph $H_0 = (\{a, b, c, d, e, f\}, \{\{a, b, c\}, \{c, d, e\}, \{c, d, f\}\})$.

⊣

Lemma 18 *Let φ be a formula of **CCQ**$^{\neq}$. Then there exists a formula φ' of **F-CCQ**$^{\neq}$ and an exact reduction from problem* Enum(φ) *to* Enum(φ')

Proof - Let φ be a σ-formula of $\mathbf{CCQ}^{\neq}$ of the form $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where ψ is quantifier-free, H be the hypergraph of φ and H' be an free(φ)-connex acyclic inclusive extension of H. Let $\mathcal{M}$ be a σ-structure.

Consider

$$\varphi'(\bar{x}) \equiv \exists \bar{y}(\psi(\bar{x},\bar{y}) \wedge \bigwedge_{e \in E(H')-E(H)} R_e(\bar{z}_e))$$

where R_e is a new relation symbol of arity $|e|$ and $\bar{z}_e$ is the tuple of variables of e. We construct $\mathcal{M}'$ as an extension of $\mathcal{M}$ where for each edge $e' \in E(H') - E(H)$ such that $e' \subseteq e \in E(H)$, $R_{e'}$ is projection of R_e on the variables of e'. It is easily seen that $\varphi'(\mathcal{M}') = \varphi(\mathcal{M})$ and $\mathcal{M}'$ can be built in time $O(|\mathcal{M}|)$.
Let (T, A) be a free(φ)-connex tree-structure of H'. Denote by ψ' the matrix of φ'. Introduce for each σ-atom A_i of edge $e \in A$ a new variable α_i and for each each σ-atom B_i of edge $e \notin A$ a new variable β_i. Consider the formula $\varphi''(\alpha_1, \ldots, \alpha_l) \equiv \exists \beta_1 \ldots \exists \beta_m \psi'(\alpha_1, \ldots, \alpha_l, \beta_1, \ldots, \beta_m)$. where ψ'' is the formula obtained from ψ' by translation (1) of preliminaries.
It is easily seen that φ'' is an **F-CCQ**-formula. and that this is an exact reduction from problem Enum(φ) to Enum(φ'').

□

This immediately yields the following Theorem

Theorem 19 *Let φ be a query of $\boldsymbol{CCQ}^{\neq}$. Then* Enum(φ) $\in$ Constant-Delay$_{lin}$.

6.2 Polynomial time characterization of S-connex acyclic hypergraphs

This part is devoted to the notion of (ext)-S-connex acyclic hypergraphs for which several characterizations are given. As a consequence we will obtain that checking if a hypergraph is S-connex acyclic by inclusive extension can be done in polynomial time.

Definition 16 *Let H be an hypergraph. Two vertices x, y of $V(H)$ are* neighbors *if they both belong to some hyperedge $e \in E(H)$.*

Lemma 20 (Helly property of acyclic hypergraphs)

(i) Let $T_1, \ldots, T_n$ be subtrees of some tree T such that $T_i \cap T_j \neq \emptyset$, $1 \leq i < j \leq n$. Then, $\bigcap_{i \leq n} T_i \neq \emptyset$ i.e. the trees have a part in common.

(ii) Let $H = (V, E)$ be an acyclic hypergraph and $A \subseteq V$. If there is no hyperedge $e \in E$ such that $A \subseteq e$ then there exist two vertices $x, y \in A$ which are not neighbors in H.

Proof - For the proof of Item (i), let us fix some arbitrary vertex r as a root of T. Let r_i be the root of the subtree T_i in the rooted tree T. Since, for all i, j, $1 \leq i < j \leq n$, $T_i \cap T_j \neq \emptyset$ then, either r_i belongs to the subtree T_j or r_j belongs to the subtree T_i. Consequently, all the roots $r_1, \ldots, r_n$ consecutively appear on the same branch of T. W.l.o.g., one can suppose that r_1 is the farest root from r among them. Together with the fact that $T_1 \cap T_i \neq \emptyset$, for all $1 < i \leq n$, this implies that $r_i \notin T_1$, hence $r_1 \in T_i$ and $r_1 \in \bigcap_{i \leq n} T_i$.
For (ii), we prove the reverse implication. Let T be the tree-structure associated to H and T_x be the (connex) subtree of T of hyperedges that contains x. Suppose that for each pair of vertices x, y of H, there exists $e \in E$ such that $\{x, y\} \subseteq e$. This implies that $T_x \cap T_y \neq \emptyset$, for all $x, y \in V$ and that $\bigcap_{x \in V} T_x \neq \emptyset$ by Item (i). In other words, there exists some $e \in E$ that contain all $x \in A$ i.e. $A \subseteq e$. □

Recall that the *induced subhypergraph* of $H = (V, E)$ on a set of vertices $S \subseteq V$ is the hypergraph $H[S] = (S, \{e \cap S \neq \emptyset : e \in E\})$.

Definition 17 *Let T be the tree-structure of an hypergraph H and $A \subseteq V(H)$. Then, $T[A]$ denotes the tree T where each node e of T is replaced by $e \cap A$.*

One have the following set of results on preservation of acyclicity through tree or hypergraph restrictions.

Lemma 21 *Let H be an hypergraph and $A \subseteq V(H)$. The following properties are true.*

(i) If H is acyclic with tree-structure T then $H[A]$ is acyclic with tree-structure $T[A]$.

(ii) For each $S \subseteq V(H)$, if H is S-connex acyclic then $H[A]$ is $(S \cap A)$-connex acyclic.

(iii) For each $S \subseteq V(H)$, if H is ext-S-connex acyclic then $H[A]$ is ext-$(S \cap A)$-connex acyclic.

Proof -

(i) If the connectivity property is true for T, it is also true for $T[A]$

(ii) Suppose (T, B) is a S-connex tree-structure of H, then $(T[A], B)$ is a $S \cap A$-connex tree-structure of $H[A]$ because $S = \bigcup_{e \in B} e$ implies $S \cap A = \bigcup_{e \in B} (e \cap A)$.

(iii) Suppose H' is an inclusive extension of H and that it is S-connex acyclic. Then, $H'[A]$ is $S \cap A$-connex acyclic and is an inclusive extension of $H[A]$. It suffices to remark that the hyperedges of $H'[A]$ are of two kinds: those of the form $e \cap A$ for $e \in E(H)$ which are the hyperedges of $H[A]$; those of the form $e' \cap A$ for $e' \subseteq e \in E$ which verify $e' \cap A \subseteq e \cap A \in E(H[A])$.

□

Lemma 22 *Let $C \subseteq V$ be a connex subset of an acyclic hypergraph $H = (V, E)$ then the sub-hypergraph $H_{\cap C}$ whose set of hyperedges is $E_{\cap C} = \{e \in E : e \cap C \neq \emptyset\}$ is acyclic*

REMARK. Note that the Lemma above is not true anymore if C is not connex. Consider the set $C = \{d, e, f\}$ with the following hypergraph H:

$$H = (\{a, b, c, d, e, f\}, \{\{a, b, d\}, \{b, c, e\}, \{a, c, f\}, \{a, b, c\}\})$$

H is acyclic, C is not connex in H and $E_{\cap C} = (\{a, b, c, d, e, f\}, \{\{a, b, d\}, \{b, c, e\}, \{a, c, f\}\})$ is not acyclic. ⊣

Proof of Lemma 22 - It is enough to show that the set $E_{\cap C}$ is a connex subtree of a tree-structure T of H. We prove it by induction on $|C|$. If $C = \{x\}$ then, $E_{\cap C}$ is the set E_x of nodes of T that contain x which is connex by definition of a tree-structure. Let $|C| \geq 2$, $x, y \in C$ and $e_{x,y} \in E$ to which x and y belong (recall that C is connex). Let also $C' = C - \{x\}$. By induction hypothesis the set $E_{\cap C'}$ is a connex part of T. In addition:
- for each $e \in E$, $e \cap C \neq \emptyset$ if and only if $e \cap C' \neq \emptyset$ or $x \in e$, then, $E_{\cap C} = E_{\cap C'} \cup E_x$;
- $E_{\cap C'} \cap E_x \neq \emptyset$ since $e_{x,y} \in E_{\cap C'} \cap E_x$.
Consequently, since $E_{\cap C}$ is the union of two non disjoint connex parts of T, it is a connex part of T. □

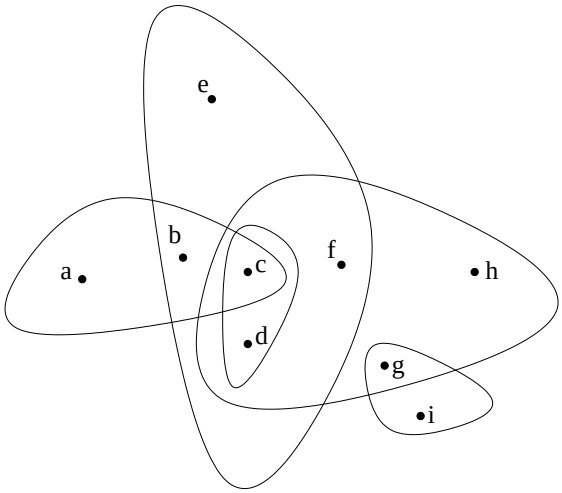

Figure 5: Hypergraph H with $S = \{c, d, f, g\}$

Definition 18 (S-component) *Let $H = (V, E)$ be a hypergraph and $S \subseteq V$. One denotes by $E_{\not\subseteq S}$ the set of hyperedges $\{e \in E : e \not\subseteq S\}$. The* S-component *of a hyperedge $e \in E_{\not\subseteq S}$ is the set of hyperedges $e' \in E_{\not\subseteq S}$ such that in the induced hypergraph $H[V - S]$, there exists a path between the hyperedges $e - S$ and $e' - S$. More precisely, there exist $e_0, e_1, \ldots, e_k$ such that $e_0 = e - S$, $e_k = e' - S$, $e_i \in E(H[V - S])$ and $e_i \cap e_{i+1} \neq \emptyset$ for all $i \leq k$.*

It is clear that the set of hyperedges $E_{\not\subseteq S}$ can be partitioned in pairwise disjoint S-component.

EXAMPLE. Let us consider the hypergraph H of Figure 5 with $S = \{c, d, f, g\}$. There are two S-components $E_1 = \{\{a, b, c\}, \{b, c, d, e, f\}\}$ and $E_2 = \{\{g, i\}, \{c, d, f, g, h\}\}$. The hypergraph $H[S] = (\{g\}, \{c, d\}, \{\{c, d, f, g\}\})$

⊣

In order to state the various characterizations of ext-S-connex acyclic hypergraph, we need to introduce the notion of S-path.

Definition 19 *S-path Let $H = (V, E)$ be a hypergraph and let $S \subseteq V$. An S-path is a path $(x, a_1, \ldots, a_k, y)$ such that*

- *x and y belong to S,*
- *vertices $a_i, 1 \leq i \leq k$, belong to $V - S$, and*
- *no hyperedge includes both x and y.*

Lemma 23 *Let $H = (V, E)$ be an acyclic hypergraph and $S \subseteq \bigcup_{e \in E} e$. Let $E_1, \ldots, E_l$ be the list of all S-components of $E_{\not\subseteq S}$. For all $i \leq l$, let $H_i = (V_i, E_i)$ with $V_i = \bigcup_{e \in E_i} e$. Let $e_i = V_i \cap S$. Finally, let H' be the hypergraph with $V(H') = V$ and $E(H')$ be the disjoint union $E(H') = E(H[S]) \dot{\cup} \dot{\bigcup}_{i \leq l} E_i$.*
Then, the four conditions below are equivalent.

1. *H is ext-S-connex acyclic.*
2. *H is acyclic and does not contain a S-path.*
3. *For each $i \leq l$, there exists a hyperedge $e'_i \in E_i$ such that $e_i \subseteq e'_i$.*
4.
   (i) *for each $i \leq l$, $e_i \in E(H[S])$ and there exists a hyperedge $e'_i \in E_i$ such that $e_i \subseteq e'_i$.*
   (ii) *H' is S-connex acyclic with (T', A) as tree-structure where $A = E(H[S])$ and the tree T' is made of the pairwise disjoint tree-structures T_S of $H[S]$, T_i of H_i, $1 \leq i \leq l$ together with the edges (e_i, e'_i) between T_S and T_i.*

Proof - We first show that, for each $i \leq l$, the hypergraph $H_i = (V_i, E_i)$ is acyclic. By definition of E_i and V_i, the set of vertices $C_i = V_i - S$ is connex in H and $E_{\cap C_i} = \{e \in E : e \cap C_i \neq \emptyset\} = E_i$. By Lemma 22, the hypergraph $H_i = H_{\cap C_i}$ is acyclic. It is also immediate to see that $H[S]$ is acyclic by Lemma 21.
The proof of $4 \Rightarrow 1$ is evident, since, by definition H' is an inclusive extension of H.

Proof of 1⇒2. By absurd. Suppose that H is ext-S-connex acyclic and that it contains a S-path $P = (x, a_1, \ldots, a_k, y)$, $k \geq 1$. W.l.o.g. P is supposed to be chordless. By Lemma 21 and since $S \cap P = \{x, y\}$ the induced hypergraph $H[P]$ is ext-$\{x, y\}$-connex acyclic. But, $H[P]$ is a binary hypergraph (essentially a path graph), it is made of binary hyperedges $\{x, a_1\}$, $\{a_1, a_2\}$, $\ldots$, $\{a_k, y\}$ and may contain some additional unary hyperedges. No inclusive extension of $H[P]$ can contain a (binary) hyperedge e such that $x \in e$ and $y \in e$. Contradiction.

Proof of 2⇒3 (by proving ¬3 ⇒ ¬2). Let us suppose that there is a $i \leq l$ for which there is no hyperedge $e \in E_i$ such that $e_i \subseteq e$. By Lemma 20 for hypergraph $H_i = (V_i, E_i)$ and part $e_i \subseteq V_i$, there exist two vertices $x, y \in e_i$ which are not neighbors in H_i. By definition of e_i, there must exist two distinct hyperedges e_x and e_y of E_i which contain respectively x and y. By definition of the S-component E_i (in the hypergraph $H[V - S]$), the hyperedges $e_x - S$ and $e_y - S$ belong to a path $(e_0, e_1, \ldots, e_k)$ with $e_0 = e_x - S$, $e_k = e_y - S$ and $e_j \cap e_{j+1} \neq \emptyset$ for all $j < k$. In other words, there exists a path $(a_1, a_2, \ldots, a_k)$ where each $a_j \in e_{j-1} \cap e_j$ hence $a_j \in V - S$. One of the two following cases may occur:

1. Either x and y are not neighbors in H. In this case $(x, a_1, \ldots, a_k, y)$ is a S-path.
2. Either there exists a hyperedge $e \in E$ such that $x, y \in e$. In this case, since x and y are not neighbors in H_i, then $e \notin E_i$ and $x, y \in e \in E - E_i$. W.l.o.g., one can suppose that the path $(x, a_1, \ldots, a_k, y)$ is chordless. By Lemma 22, the hypergraph $H[P]$ is acyclic. Its set of hyperedges exactly contains $\{x, a_1\}$, $\{a_1, a_2\}$, $\ldots$, $\{a_k, y\}$ and $e \cap P$ (and maybe additional unary hyperedges). By definition of the S-component and the fact that $e \notin E_i$, no a_j, $1 \leq j \leq k$ belongs to e. Then $e \cap P = \{x, y\}$ which contradicts the acyclicity of $H[P]$. This later case can then never occur.

Proof of 3⇒4. Suppose that, for each $i \leq l$, there exists some hyperedge $e'_i \in E_i$ such that $e_i \subseteq e'_i$. We will prove Items 4i and 4ii.
Since $V_i = \bigcup_{e \in E_i} e$, it holds that $e'_i \subseteq V_i$. Then:

$$e_i \subseteq e'_i \cap S \subseteq V_i \cap S = e_i,$$

which implies $e_i = e'_i \cap S$. Since $e'_i \in E$, it holds that $e_i \in E(H[S])$. Hence 4i is true.
By construction, T' is a tree whose set of nodes is $E(H')$ and because each vertex of S belongs to some hyperedge of H, it holds $\bigcup_{e \in A} e = S$. It remains to verify the connectivity condition for T'. We have proved that all sub-trees T_S and T_i, $1 \leq$, are acyclic. It suffices to show that, for each vertex x, if x belongs to two of these subtrees, it also belongs to the nodes e_i and e'_i that relates these subtrees. Let $x \in e \cap e'$ with $e \in E(H[S])$ and $e' \in E_i$. It holds that $x \in e \cap e' \subseteq S \cap V_i = e_i \subseteq e'_i$. Then, $x \in e_i$ and $x \in e'_i$ as needed. Let now $x \in e \cap e'$ with $e \in E_i$ and $e' \in E_j$ for $i \neq j$. By definition of S-components, $e \cap e' \subseteq S$ which implies $x \in S$. Since $x \in e \cap S \in E(H[S])$, $x \in e' \cap S \in E(H[S])$, one obtains $x \in e_i$, $x \in e'_i$, $x \in e_j$, $x \in e'_j$ as in the preceding case. Then, T' verifies the connectivity property and (T', A) is a S-connex tree structure of H'. □

EXAMPLE. The hypergraph H of Figure 5 detailled in the preceding example is ext-S-connex acyclic. Condition 3 of Lemma 23, is the easiest to check. ⊣

Lemma 23 implies the following corollary.

Corollary 24 *Let $H = (V, E)$ be an acyclic hypergraph and $S \subseteq \bigcup_{e \in E} e$. Let $E_1, \ldots, E_l$ be the list of all S-components of $E_{\not\subseteq S}$. For all $i \leq l$, let $V_i = \bigcup_{e \in E_i} e$, $e_i = V_i \cap S$ and $H'_i = (V_i, E'_i)$ with $E'_i = E_i \cup \{e_i\}$. The four conditions below are equivalent.*

1. *H is ext-S-connex acyclic.*
2. *H does not contain a S-path.*
3. *The hypergraphs $H[S]$ and H'_i, $1 \leq i \leq l$, are acyclic.*
4. *H has for S-connex acyclic inclusive extension the hypergraph $H' = H[S] \cup \bigcup_{i \leq l} H'_i$ whose tree-structure is $T' = T_S \cup \bigcup_{i \leq l} T'_i$ where T_S and T'_i are the tree-structure of $H[S]$ and H'_i, $1 \leq i \leq l$.*

CONNEX-ACYCLIC

Input: A hypergraph $H = (V, E)$ and a set $S \subseteq \bigcup_{e \in E} e$
Output: is H a ext-S-connex acyclic hypergraph ? if yes, output such a S-connex acyclic extension H' together with its tree-structure (T', A).

Theorem 25 *The problem* CONNEX-ACYCLIC *is polynomial time decidable.*

Proof - Due to the preceding Corollary, it suffices to run the following algorithm.

- Build $H[S]$. If $H[S]$ is not acyclic then return false, else build the tree-structure T_S of $H[S]$
- Build the S-components $E_1, \ldots, E_l$ of $E_{\not\subseteq S}$ and the sets $V_i = \bigcup_{e \in E_i} e$, $1 \leq i \leq l$

- For each $i \leq l$, compute $e_i = V_i \cap S$ and the hypergraph $H'_i = (V_i, E'_i)$ where $E'_i = E_i \cup \{e_i\}$. If H'_i is not acyclic return false else build the tree-structure T'_i of H'_i.
- Build the hypergraph $H' = (V, E')$ with $E' = E(H[S]) \cup \bigcup_{i \leq l} E_i$.
- Return H' together with the S-connex tree-structure (T', A) with $T' = T_S \cup \bigcup_{i \leq l} T'_i$ and $A = E(H[S])$.

□

6.3 A dichotomy theorem

We aim to show that the evaluation and enumeration problems of any acyclic query φ which is not **CCQ** are "hard" in some precise sense. For that purpose, we want to exhibit an exact reduction from the multiplication problem of $n \times n$ boolean matrices (a problem that is strongly conjectured to be not computable in $O(n^2)$ time) to Enum(φ). We need to encode our matrix problem in the first-order framework. A Two-Matrix structure is a relational σ_{AB}-structure $\mathcal{M} = (D, A, B)$ where $D = [1, n]$, $\sigma_{AB} = \{A, B\}$, and A, B are binary relations. Clearly, the multiplication problem of two $n \times n$ boolean matrices is expressed by the following acyclic σ_{AB}-query: $\Pi(x, y) \equiv \exists z(A(x, z) \wedge B(z, y))$. More precisely, Enum($\Pi$) is the problem of enumerating, for each Two-Matrix structure $\mathcal{M} = ([1, n], A, B)$ given as input, the ordered pairs of indices (i, j) where 1 occurs in the matrix product $C = A \times B$, i.e., $C_{i,j} = 1, 1 \leq i, j \leq n$.

Definition 20 *A conjunctive formula is* simple *if each relation symbol appears in at most one atom.*

Remark. Given a formula φ of **ACQ** (resp **CCQ**, **ACQ**$^{\neq}$, **CCQ**$^{\neq}$) and a structure $\mathcal{M}$, one can compute in linear time a simple formula φ' of **ACQ** (resp **CCQ**, **ACQ**$^{\neq}$, **CCQ**$^{\neq}$) and a structure $\mathcal{M}'$ such that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$. ⊣

Lemma 26 *Let φ be a simple formula which is **ACQ** (resp **ACQ**$^{\neq}$) but not **CCQ** (resp not **CCQ**$^{\neq}$). Then there exists an exact reduction from the problem* Enum(Π) *to* Enum(φ).

Proof - Suppose $\varphi \in \mathbf{ACQ} - \mathbf{CCQ}$, the hypergraph of φ is acyclic but is not e-S-connex for $S = \text{free}(\varphi)$. By Lemma 24, H_ϕ admits a chordless S-path : that is a sequence of distinct vertices $P = (x, z_1, \ldots, z_k, y)$ with $k \geq 1$, so that

- P is a path: there are $k + 1$ hyperedges $e_0, e_1, \ldots, e_{k-1}, e_k \in E$ that contain the respective pairs $e'_0 = \{x, z_1\}, e'_1 = \{z_1, z_2\}, \ldots, e'_{k-1} = \{z_{k-1}, z_k\}, e'_k = \{z_k, y\}$
- P is an S-path: $x, y \in S$ and $z_1, \ldots, z_{k-1} \notin S$
- P is chordless: for each $e \in E, |e \cap P| \leq 1$ or $|e \cap P| = e'_i$ for some $i (0 \leq i \leq k)$

Without loss of generality, assume that the (simple) formula φ is of the form

$$\varphi(x, y, \bar{t}) \equiv \exists z_1 \ldots \exists z_k \exists \bar{u} \psi(x, y, \bar{z}, \bar{t}, \bar{u})$$

where $t = (t_1, \ldots, t_m), m \geq 0, k \geq 1, q \geq 0$, ψ is of the form $\psi \equiv \bigwedge_{e \in E} A_e$ where $A_e = R(L_e)$ and L_e is a sequence of distinct variables that consists of all the elements (vertices) of e. We can assume (without loss of generality) that each atom A_e of φ is of one of the following forms $(1-5)$ where $\bar{v} \subseteq \{\bar{t}, \bar{u}\}$:

1. $R_e(x, z_1, \bar{v})$
2. $R_e(z_k, y, \bar{v})$
3. $R_e(z_i, z_{i+1}, \bar{v})$ where $1 \leq i < k$
4. $R_e(w, \bar{v})$ where $w \in \{x, y, z_1, \ldots, z_k\}$
5. $R_e(v)$

Transformation r of the structure: To each σ_{AB}-structure $\mathcal{M} = (D, A, B)$, one associates the structure $r(\mathcal{M}) = \mathcal{M}'$ of signature $\sigma_E = \{R_e, e \in E\}$ defined as follows.

- $\mathcal{M}' = (D', (R_e)_{e \in E})$
- $D' = D \cup \{\perp\}$: here $\perp$ is a new special "padding" symbol
- For each $e \in E$ define the relation of arity p R_e according to the form (1-5: see above) of its (unique) occurrence in φ
  1. $R_e = A \times \{\perp\}^{p-2}$
  2. $R_e = B \times \{\perp\}^{p-2}$
  3. $R_e = I \times \{\perp\}^{p-2}$ where I is the identity (equality) relation of D ($I = \{(a, a) : a \in D\}$)
  4. $R_e = D \times \{\perp\}^{p-1}$
  5. $R_e = \{\perp\}^p$

The following facts are easy to prove:
Fact 1: The map $r : \mathcal{M} \to \mathcal{M}'$ is computable in time $O(|\mathcal{M}|)$
Fact 2: $\varphi'(\mathcal{M}') = \Pi(\mathcal{M}) \times \{\perp\}^m$
Hint: The variables x, y play the same role in $\varphi(x, y, \bar{t})$ as in $\Pi(x, y)$. As a consequence of Fact 1, we obtain the following fact
Fact 3: The projection $(a, b, \bar{c}) \to (a, b)$ is a one-one mapping from $\varphi(\mathcal{M}')$ onto $\Pi(\mathcal{M})$
By those facts, we have exhibited and justified an exact reduction from $\textsc{Enum}(\Pi)$ to $\textsc{Enum}(\varphi)$. This proves the Lemma for $\varphi \in \mathbf{ACQ} - \mathbf{CCQ}$. In case φ belongs to $\mathbf{ACQ}^{\neq} - \mathbf{CCQ}^{\neq}$ we can easily adapt our reduction as follows. We oblige any two distinct variables to have disjoint domains: duplicate the domain D' as $D' \times [h]$ where $h = |\text{var}(\varphi)|$, give the i^{th} variable, $1 \leq i \leq h$ the specific domain $D_i = D' \times \{i\}$, modify the transformed structure $\mathcal{M}'$ accordingly. □

Finally, one obtains the following result

Theorem 27 *(Dichotomy Theorem) Let φ be a simple* $\boldsymbol{ACQ}$ *(resp* $\boldsymbol{ACQ}^{\neq}$*) query. We have either (1) or (2):*

1. $\varphi \in \boldsymbol{CCQ}$ *(resp* $\boldsymbol{CCQ}^{\neq}$*) and* $\textsc{Enum}(\varphi) \in \textsc{Constant-Delay}_{lin}$ *(and* $\textsc{Eval}(\varphi)$ *is computable in time* $O(|\mathcal{M}| + |\varphi(\mathcal{M})|)$*)*
2. *There is an exact reduction from* $\textsc{Enum}(\Pi)$ *to* $\textsc{Enum}(\varphi)$ *and then* $\textsc{Enum}(\varphi) \notin \textsc{Constant-Delay}_{lin}$ *(and* $\textsc{Eval}(\varphi)$ *is not computable in time* $O(|\mathcal{M}| + |\varphi(\mathcal{M})|)$*) under the hypothesis that the product of two $n \times n$ matrices cannot be computed in time $O(n^2)$.*

7 Free-connex treewidth

In this section, we introduce the notion of free-connex tree-decomposition of conjunctive queries: this is a variant of the tree-decomposition of graphs

Definition 21 *(see [4] for details) A* tree-decomposition *of a graph $G = (V, E)$ is a tree T whose vertices are subsets of V and are called the* bags *of T, so that*

- *T has the connectivity property (see above),*
- *each edge of G is included in some bag of T.*

The width *of a tree-decomposition T is $\max\{|B| : B \text{ is a bag of } T\} - 1$. The* treewidth *of G denoted by $tw(G)$ is the smallest width of any tree-decomposition of G.*

Definition 22 *Let $G = (V, E)$ be a graph and S be a subset of V. A S-*connex tree-decomposition *of G is a tuple (T, A) where T is a tree-decomposition of G and A is a connected subset of $V(T)$ such that $\bigcup_{B \in A} B = S$.*
*Given a graph $G = (V, E)$ and a set $S \subseteq V$, the S-*connex treewidth *of G denoted by $ctw(G, S)$ is the smallest width of any S-connex tree-decomposition of G.*
Let φ be a formula of $\boldsymbol{CQ}$ *(resp* $\boldsymbol{CQ}^{\neq}$*). The* free-connex treewidth *of φ is the S-connex treewidth of the Gaifman graph* [5] *of φ for $S = free(\varphi)$.*

Lemma 28 *Let φ be a* $\boldsymbol{CQ}$ *(resp* $\boldsymbol{CQ}^{\neq}$*) formula of free-connex treewidth k over a structure $\mathcal{M}$ of domain D. Then we can compute in time $O(|D|^{k+1} + |\mathcal{M}|)$ a formula of* $\boldsymbol{CCQ}$ *(resp* $\boldsymbol{CCQ}^{\neq}$*) φ' (of size only depending on $|\varphi|$) and a model $\mathcal{M}'$ for φ' such that $\varphi(\mathcal{M}) = \varphi'(\mathcal{M}')$.*

Proof - Let φ be a τ-formula of the form $\varphi(\bar{x}) \equiv \exists \bar{y} \psi(\bar{x}, \bar{y})$ where ψ is quantifier-free. Let (T, A) be a free-connex tree-decomposition of φ. and $\mathcal{M}$ be a τ-structure of domain D. For each $B_i \in V(T)$ of size l, we introduce a new relation symbol R_{B_i} of arity l.
We define the formula

$$\varphi'(\bar{x}) = \exists \bar{y} \psi(\bar{x}, \bar{y}) \wedge \bigwedge_{i \in V(T)} R_{B_i}(z_{\bar{B}_i})$$

[5] The Gaifman graph of φ is the graph $G_\varphi = (V, E)$ where $V = \text{var}(\varphi)$ and E is the set of pairs of variables $\{x, y\}$ such that there exists an atom of φ that contains both x and y.

where $z_{\bar{B}_i}$ is the tuple of variables of B_i. The structure $\mathcal{M}'$ is an expansion of $\mathcal{M}$ where for every $B \in T$ or size l, $R_B = D^l$. Clearly $|\mathcal{M}'| = O(|D|^{k+1} + |\mathcal{M}|)$ and $\varphi'(\mathcal{M}') = \varphi(\mathcal{M})$ and we easily see, by definition of a free-connex tree-decomposition, that φ' is a $\mathbf{CCQ}^{\neq}$ formula. □

By the previous Lemma and Theorem 19, we obtain the following theorem

Theorem 29 *Given a fixed formula φ of $\boldsymbol{CQ}^{\neq}$ of free-connex treewidth k, the problem* Enum(φ) *can be computed with $O(|D|^{k+1} + |\mathcal{M}|)$ precomputation and constant delay.*

Proof - 1) $\Rightarrow$: Let $G = (V, E)$ be a graph and $G' = (V, E')$ be the completed graph of G. Consider $G'[S]$ the induced subgraph of G by S and $C_1, \dots C_k$ the connex componants of $G' - S$
Denote $A_i = N(C_i) \cap S$. A_i is a clique: consider x and y two distinct elements of $N(C_i) \cap S$. There are two (non necessary distinct) elements x' and y' of C_i such that $\{x, x'\}$ and $\{y, y'\}$ are edges of G. As $G[C_i]$ is connex, there is a path $(x', a_1, \dots, a_l, y')$ between x' and y' where all vertices except x' and y' are elements of C_i. Therefore $(x, x', a_1, \dots, a_l, y', y)$ is an S-path and by construction of G', there is a edge between x and y.
Consider now $G_i = G[C_i \cup A_i]$ and T_i a tree-decomposition of G_i. As A_i is a clique, there is a bag of T_i which contains A_i. We can assume without loss of generality that A_i is a bag of T_i by adding the bag A_i as leaf of the bag containing A_i. Consider T_S be a tree decomposition of $G[S]$. Similarly to the previous case, we can assume that each A_i is a bag of T_S. Let $T = T_S \cup \bigcup_i T_i$. By Lemma **??** T is a tree-decomposition of G and as $\bigcup_{B \in T_S} B = S$, $(T, V(T_S)$ is an S-connex tree-decomposition of G.
$\Leftarrow$: Let (T, A) be an S-connex tree-decomposition of G. We will prove that any edge added in the completed graph G' belongs to some bag of T. Consider $C_1, \dots C_l$ the connex componants of $G - S$. For any C_i, consider A_i and A'_i two neighbor bags in T such that $A_i \in A$ and $A'_i \cap C_i \neq \emptyset$. Denote by T'_i the subtree T containing the bag A'_i but not containing the bag A_i.
For any element $x \in C_i$ and $B \in V(T) - V(T'_i)$, it holds $x \notin B$: by contradiction, assume that there is a bag $B \in V(T) - V(T'_i)$ and a vertex $x \in B \cap C_i$. Let y be a vertex of $A'_i - A_i$. As $G[C_i]$ is connex, there is a path P between x and y where all elements except x and y are elements of C_i. As the path in T between A'_i and B contains the bag A_i, there is a vertex of P which belongs to A_i. This is a contradiction.
$N(C_i) \cap S \subseteq A_i$: any vertex x of C_i belongs to a bag $B \in V(T'_i)$ therefore any neighbor y of x belong to a bag B' neighbor of B in T. So $B' \in V(T_i)$ or $B' = A_i$. $y \in A_i \cup C_i$.
Let $\{x, y\}$ be an edge added in the completed graph. By definition, there is a path x, a_1, a_p, y with $a_1, \dots, a_l \notin S$. $a_1, \dots a_p$ belong to the same connex componant C_i. As $N(C_i) \cap S \subseteq A_i$, the edge $\{x, y\}$ belongs to the bag A_i which is the desired conclusion.
2) This can be done by the following algorithm

- Compute the completed graph G' of G
- Return False if $tw(G') > k$
- Compute the set of connex componants C_i of $G'[V - S]$
- For any C_i, compute $A_i = N(C_i) \cap S$

- Compute a tree-decomposition T_S of $G'[S]$ of size k where each A_i is a bag of T_S
- For each i, compute a tree-decomposition T'_i of $G'[A_i \cup C_i]$ of size k such that A_i is a bag of T_i
- Return the S-connex tree-decomposition $T = T_S \cup \bigcup T'_i$ of H'

Bodlaender [5] proves that for a given fixed k, there is a linear time algorithm which returns a tree-decomposition of width k of a graph G if $tw(G) \leq k$ and returns False otherwise. All other steps can be easily be done in polynomial time. □

We now give a polynomial algorithm to compute the free-connex treewidth of a formula. We need to introduce the notion of completed graph.

Definition 23 *Let G be a graph. The* completed graph G' *of (G, S) is built as follows*

- $V(G') = V(G)$
- $E(G') = E(G) \cup \{\{x, y\} : x, y \in V$ *and there is an S-path between x and $y\}$*

Theorem 30 *Let G be a graph, let $S \subseteq V(G)$ and let G' be the completed graph of (G, S). It holds*

1. *$ctw(G, S) = tw(G')$;*
2. *For any fixed k, there is a polynomial time algorithm which returns an S-connex tree-decomposition of G of width at most k if $ctw(G, S) \leq k$ and returns False otherwise.*

Notice that the algorithm of point (2) uses the k-tree decomposition procedure of [5] applied to the completed graph G'.

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: $|H|$.
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: $|H|, d$.
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 31 *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:

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

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 **??**.

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 **??**. The bound on the linear delay comes from Corollary **??**. □

8.2 Covering and matching problems

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

Corollary 32 *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 33 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 [12] 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, $1 \leq i \leq k$, $|S \cap X_i| = 1$?

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 33 *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:

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

The formulas are similar for the two other problems. □

References

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

[1] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proc. of the Annual Conference of the European Association for Computer Science Logic (CSL)*, pages 167–181, 2006.

[2] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth element of a first-order query, 2007. submitted.

[3] C. Berge. *Graphs and hypergraphs*. Amsterdam, 2nd edition, 1973.

[4] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[5] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[6] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[7] Henry Cohn, Robert D. Kleinberg, Balázs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *FOCS*, pages 379–388, 2005.

[8] Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Maths*, 2006. To appear.

[9] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.

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

[10] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *Transactions on Computational Logic*, To appear.

[11] Arnaud Durand and Frédéric Olive. First-order queries over one unary function. In *Proc. of the Annual Conference of the European Association for Computer Science Logic (CSL)*, pages 334–348, 2006.

[12] 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.

[13] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[14] Etienne Grandjean and Thomas Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM J. Comput.*, 32(1):196–230, 2002.

[15] Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, pages 657–666, 2001.

[16] Leonid Libkin. *Elements of finite model theory*. Springer, 2004.

[17] Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

[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.

[18] M. Yannakakis. Algorithms for acyclic database schemes. pages 82–94, 1981.