RAIRO-Theor. Inf. Appl. **42** (2008) 147–164
DOI: 10.1051/ita:2007046

Available online at:
www.rairo-ita.org

COMPUTING THE JTH SOLUTION OF A FIRST-ORDER QUERY *

Guillaume Bagan[1], Arnaud Durand[2], Etienne Grandjean[1] and Frédéric Olive[3]

Abstract. We design algorithms of "optimal" *data complexity* for several natural problems about first-order queries on structures of bounded degree. For that purpose, we first introduce a framework to deal with logical or combinatorial problems $R \subset I \times O$ whose instances $x \in I$ may admit of several solutions $R(x) = \{y \in O : (x,y) \in R\}$. One associates to such a problem several specific tasks: compute a *random* (for the uniform probability distribution) solution $y \in R(x)$; *enumerate* without repetition each solution y_j in some specific linear order $y_0 < y_1 < \ldots < y_{n-1}$ where $R(x) = \{y_0, \ldots, y_{n-1}\}$; compute the solution y_j of *rank* j in the linear order $<$. Algorithms of "minimal" data complexity are presented for the following problems: given any first-order formula $\varphi(\overline{v})$ and any structure S of bounded degree: (1) compute a random element of $\varphi(S) = \{\overline{a} : (S, \overline{a}) \models \varphi(\overline{v})\}$; (2) compute the jth element of $\varphi(S)$ for some linear order of $\varphi(S)$; (3) enumerate the elements of $\varphi(S)$ in lexicographical order. More precisely, we prove that, for any fixed formula φ, the above problem (1) (resp. (2), (3)) can be computed within average constant time (resp. within constant time, with constant delay) after a linear ($O(|S|)$) precomputation. Our essential tool for deriving those complexity results is a *normalization* procedure of first-order formulas on *bijective structures*.

Mathematics Subject Classification. 68Q15, 68Q19.

Keywords and phrases. Complexity of enumeration, first-order queries, structures of bounded degree, linear time, constant time, constant delay.

* *In Honour of Serge Grigorieff at the occasion of his 60th Birthday. In Memoriam Clemens Lautemann.*

[1] GREYC, Université de Caen, ENSICAEN, CNRS, Campus 2, 14032 Caen Cedex, France; gbagan@info.unicaen.fr; grandjean@info.unicaen.fr

[2] Équipe de Logique Mathématique, Université Denis Diderot, CNRS UMR 7056, 2 place Jussieu, 75251 Paris Cedex 05, France; durand@logique.jussieu.fr

[3] LIF, Université Aix-Marseille 1, CNRS, 39 rue Joliot Curie, 13453 Marseille Cedex 13, France; frederic.olive@lif.univ-mrs.fr

1. Introduction

After the study of the complexity of the logical theories (see [9] for a nice survey), the complexity of query evaluation is a natural and well-studied problem of computer science. Several algorithmic questions may be considered. Classically, given a structure S in some class $\mathcal{C}$ and a sentence φ in some logic $\mathcal{L}$ one wants to know how hard it is to evaluate whether φ holds in S. The complexity can be measured either in terms of the structure size (*data complexity*) or in terms of the structure size *and* the formula size (*combined complexity*). When the formula has free variables then the size of the output also serves in the complexity evaluation. It is well known that the combined complexity of first-order query evaluation is intractable in general [15]. Hence, finding restrictions on the structure or on the class of formulas that admit efficient query evaluation is an important problem and has deserved a lot of attention (see, for example, [6–8, 12, 14, 16, 17]).

In this article, we do not consider the global cost of computing the whole result of a query. Instead, query evaluation is studied here from a *dynamical* point of view. Three kinds of algorithmic tasks are considered: (i) the *Random solution* problem which consists in computing a random tuple of the result (for the uniform probability distribution); (ii) the *Enumeration* problem that aims at generating successively each tuple of the result in some specific linear order $<$; (iii) the jth *solution* problem which computes the tuple of rank j for the linear order $<$. In other words, the objective is to compute a single or a new solution (tuple) of the query result. A central notion in algorithms that solve these problems is that of *precomputation.* Before starting each specific task, a (short) precomputation is allowed, based on the structure and the formula only. Afterwards, the algorithm must be able to compute efficiently a particular solution (for example, the jth one, whatever j is, for some order $<$) or to enumerate all the solutions with a small delay between two consecutive ones. Complexity bounds for tasks (ii) or (iii) obviously provide complexity bounds for the classical query problem: *e.g.*, the total number of steps to compute the entire result of the query is *time(precomputation)* + *(delay* $*$ *number of solutions).* Of course, analyzing tasks (i–iii) gives *additional* information on *uniformity* or *regularity* of the query evaluation process.

The complexity of the enumeration problem of (several classes of) logical queries has begun to be studied in some recent articles: see [1,3,5] for first-order or monadic second-order logic over structures of bounded tree-width and [2] for acyclic conjunctive queries with disequalities over any structures. However, the present article is the first one, to our knowledge, that studies the complexity of the random solution and the jth solution problems of queries.

In this work that improves and completes the results of [4, 16], we consider first-order queries on *bounded degree structures* and we give complexity results for tasks (i–iii) above. Given a first-order formula $\varphi(x_1, \ldots, x_k)$ and a structure S of bounded degree, $\varphi(S)$ denotes the set of tuples $\{\overline{a} : (S, \overline{a}) \models \varphi(\overline{x})\}$ that constitute the result of query φ over S. We prove that, for a fixed formula φ, computing a random tuple of $\varphi(S)$ (resp. computing the jth element of $\varphi(S)$ for some linear order of $\varphi(S)$) can be done in average constant time (resp. in constant time) after a linear ($O(|S|)$) precomputation that depends on formula φ and structure S only. In other words, the value of j is given only *after* the precomputation step is completed. For the enumeration task, we prove that one can enumerate all the elements of $\varphi(S)$ in *lexicographical* order with a constant delay between two consecutive solutions, again after a linear precomputation. On the one hand, it seems necessary to read the whole input structure: this corresponds to the linear time precomputation. On the other hand, the time required to compute one tuple is at least its (constant) size. That means that our results are of *optimal data complexity*.

In this article, we consider first-order logic over the so-called *bijective structures* that consist of bijective unary functions and monadic predicates. First-order logic over such structures is the analog of first-order logic over *structures of bounded degree* when dealing with *functions* instead of *relations*. Our algorithmic results will derive from a *quantifier elimination* and *normalization* procedure of first-order formulas over bijective structures. More precisely, the notion of *inductive description* is introduced as a natural way to describe relationships between variables and a new normal form for first-order logic on bijective structures is proved: every first-order formula can be put in a quantifier-free form that is equivalent to an *exclusive* disjunction of inductive descriptions.

The article is organized as follows. In Section 2, the main definitions of our problems and complexity classes are given. Section 3 presents the logical notions and our main normalization result. They are used in Section 4 to prove our complexity bounds for first-order queries over bijective structures. Finally, Section 5 derives similar results for queries over structures of bounded degree.

2. Preliminaries

2.1. Problems

We will deal with problems whose instances may admit of several solutions. Given an instance, we can be interested in different actions: counting its solutions, enumerating them all, choosing one of them randomly, and so on. In order to handle this diversity of problems in a uniform way, we introduce the notion of *problem relation* which is merely a binary relations $R \subseteq I \times O$ 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 $R(x) = \{y : (x, y) \in R\}$ is called the *set of solutions* of R on x. We will demand this set to be finite for each x. Now, the standard *decision* and *evaluation* problems associated with R can be defined as follows:

DECIDE(R)
Input: an instance x of R
Output: accept iff $R(x) \neq \emptyset$

EVAL(R)
Input: an instance x of R
Output: $R(x)$

We will deal with the *counting* and *random* problem associated with R:

COUNT(R)
Input: an instance x of R
Output: $card(R(x))$

RANDOM(R)
Input: an instance x of R
Output: a *random* element of $R(x)$

But we will mainly focus on two other problems whose definition needs some more notation: Let $<$ be a strict linear order defined on the space of solutions of R. For any input x, we denote by $\mathsf{enum}(R, x, <)$ the strictly increasing enumeration of $R(x)$ according to $<$ and by $\mathsf{jth}(R, x, <, j)$ the element of rank j in this enumeration. That is: if $R(x) = \{y_0, \ldots, y_{n-1}\}$ with $y_0 < \cdots < y_{n-1}$, then $\mathsf{enum}(R, x, <) = y_0 y_1 \ldots y_{n-1}$ and $\mathsf{jth}(R, x, <, j) = y_j$. The *enumeration* and jth *solution* problems related to R and $<$ are respectively defined by:

ENUM$(R, <)$
Input: an instance x of R
Output: $\mathsf{enum}(R, x, <)$

JTH$(R, <)$
Input: an instance x of R and $j \in \mathbb{N}$
Output: $\mathsf{jth}(R, x, <, j)$

Notice that beside the ordering in which the solutions of R on x appear in the final output, there is an important difference between the problems EVAL(R) and ENUM$(R, <)$ which is not quite explicit in our definitions: in the enumeration problem, it is essential that the solutions are returned one after another, while it is not required in the evaluation version of the problem. This will be made precise in the forthcoming definition of enumeration algorithms.

2.2. Complexity classes

The computation model used in this article is a *random access machine* model (RAM) with *uniform cost measure*, with addition and substraction as its basic arithmetic operations (see [10]). It has read-only *input registers* (containing the input x of the problem), read-write *work registers* and write-only *output registers*. The *size* of an input (resp. output) x, denoted by $|x|$, is the number of registers it occupies (in a "reasonable" representation). We suppose that in each computation of the RAM, register values or memory addresses are bounded by $c.|x|$, for some constant integer c. (Note that this bound applies to each register including input and output registers.) Although this latter assumption may appear to restrict the computational power of RAMs, it is in fact realistic and close to the intuition of what linear time (or related small complexity measure) is. Moreover, one defines a *random RAM* to be a RAM with an additional instruction $RANDOM(reg)$ whose semantics is: store in register reg a *random* integer in the interval $[0, c.|x| - 1]$.

We say that a (RAM) algorithm $\mathcal{A}$ computes the enumeration problem ENUM$(R, <)$ if, for any input x, $\mathcal{A}$ computes one after one the elements of the sequence $\mathsf{enum}(R, x, <)$ and stops immediately after writing the last one. For such a procedure, we denote by $\text{time}_j(x)$ the moment when $\mathcal{A}$ has completed the writing of the jth solution (by convention, $\text{time}_0(x) = 0$). We define $\text{delay}_j(x) = \text{time}_j(x) - \text{time}_{j-1}(x)$.

An enumeration algorithm $\mathcal{A}$ that computes a problem $\text{ENUM}(R,<)$ is *constant delay* if there is a constant c such that for, any input x, we have $delay_j(x) \leq c$ for any $j < card(R(x))$, and if, furthermore, $\mathcal{A}$ uses space less than c (*i.e.* at most c memory registers) during its computation.

The problem $\text{Jth}(R,<)$ is *computable in constant time* if there is a constant time and constant space RAM which, for each input x and each integer j, computes $card(R(x))$ and $\mathsf{jth}(R,x,<,j)$ if $j \leq card(R(x))$. Similarly, $\text{RANDOM}(R)$ is *computable in average constant time* if there is a random RAM which, for each input x, computes a random element of $R(x)$ in constant space and average constant time.

These complexity classes of problems are not computationally robust. However, their respective closures by linear reductions are.

Definition 2.1. We write $\text{ENUM}(R,<) \in \text{Constant-Delay}_{\text{lin}}$ if there is a function $r : x \mapsto r(x)$ computable in linear time and a constant delay algorithm $\mathcal{A}$ which, when applied to $r(x)$ for any input x computes $\mathsf{enum}(R,x,<)$.

Similarly, we write $\text{Jth}(R,<) \in \text{Constant-Time}_{\text{lin}}$ (resp. $\text{RANDOM}(R) \in \text{AVG}-\text{CONST}-\text{TIME}_{\text{lin}}$) if there is a function $r : x \mapsto r(x)$ computable in linear time and a RAM (resp. random RAM) $\mathcal{A}$ which, when applied to $(r(x), j)$ (resp. $r(x)$), for any input x and any integer j (resp. for any input x), runs in constant time (average constant time) and constant space and computes $card(R(x))$ and $\mathsf{jth}(R,x,<,j)$ (resp. computes a random element of $R(x)$)[1].

2.3. Algorithms with precomputation

It is convenient to reformulate the above defined classes by using *RAM algorithms with precomputation*: $\text{Jth}(R,<)$ belongs to $\text{Constant-Time}_{\text{lin}}$ if there is a *RAM* algorithm $\mathcal{A}$ which, for any input x and any integer j, decomposes into the following two successive phases:

(*a*) $\mathcal{A}.\mathbf{precomp}(x)$ which performs linear time precomputation that computes $r(x)$ and $card(R(x))$ from x (and independently of j).
(*b*) $\mathcal{A}.\mathbf{output}(r(x), j)$ which computes $\mathsf{jth}(R,x,<,j)$ (if $j < card(R(x))$) in constant time and constant space.

By abuse of notation, phase (*b*) will be called $\mathcal{A}.\mathbf{output}(x, j)$ in the next sections.

Remark 2.2.
- Note the crucial assumption that the precomputation phase $\mathcal{A}.\mathbf{precomp}$ does not use the part j of the input.

[1]Note that any of those three conditions implies that $card(R(x))$ is "polynomial", that means, is $O(|x|^k)$, for some fixed k. For $\text{RANDOM}(R) \in \text{AVG}-\text{CONST}-\text{TIME}_{\text{lin}}$ (for example), this can be justified as follows: if we have $card(R(x)) > (c.|x|)^k$, for some integer k, then most solutions $y \in R(x)$ have size $|y| > k$ (that means, y should be stored in more than k registers) and, hence, the average time required to compute and write a random element of $R(x)$ is greater than $k/2$.

- The phase $\mathcal{A}$.**output** uses constant space, that is, a fixed number of registers. Those "work" registers are disjoint from the "input" registers that contain $(r(x), j)$ and are read-only.

An $\mathrm{AVG-CONST-TIME}_{\mathrm{lin}}$ algorithm $\mathcal{A}$ for $\textsc{random}(R)$ is similarly defined by replacing the phase $\mathcal{A}.\mathbf{output}(r(x), j)$ by the new phase $\mathcal{A}.\mathbf{random}(r(x))$ which computes a random element of $R(x)$ in constant space and average constant time by using the instruction $RANDOM(reg)$.

Finally, a $\textsc{Constant-Delay}_{\mathrm{lin}}$ algorithm $\mathcal{A}$ for $\textsc{enum}(R, <)$ is defined in the same way by replacing the phase $\mathcal{A}$.**output** by a procedure $\mathcal{A}.\mathbf{enum}(r(x))$ which computes $\mathsf{enum}(R, x, <)$ with constant delay and constant space[2].

Remark 2.3. In some of the forthcoming algorithms (see $\mathsf{COUNT}(\varphi, S)$ and $\mathsf{JTH}(\varphi, S)$ in the proof of Prop. 4.1) we fully use the robustness of the linear time complexity and of the $\textsc{Constant-Time}_{\mathrm{lin}}$ and $\textsc{Constant-Delay}_{\mathrm{lin}}$ classes. These classes are not affected by the enrichment by arithmetical operations (multiplication, exponentiation, ...) of the set of basic operations that equip our RAM model. In particular, an algorithm $\mathcal{A}$ designed to prove the belonging to one of these classes can freely refer to $u \operatorname{div} v$ or $u \bmod v$, for some fixed v smaller than the size of the input and for any u which is polynomial in this size ($u = O(|x|^k)$, for some fixed k). This is because we can build and store the tables of those operations in the precomputation phase of $\mathcal{A}$, so that their results on any operand u can be evaluated in constant time and constant space during the output phase.

The class $\mathrm{AVG-CONST-TIME}_{\mathrm{lin}}$ is also robust for similar reasons: observe that any instruction of the form $j \leftarrow RANDOM(0 \dots v-1)$ (take an integer randomly in the interval $[0 \dots v-1]$) where v is any integer such that $v = O(|x|^k)$, for some fixed k, can be simulated in constant space and average constant time by standard probabilistic techniques using the instruction $RANDOM(reg)$ (see, for example, Exercise 1.2 in [13]).

The following lemmas are consequences of our definitions.

Lemma 2.4. *If* $\textsc{Jth}(R, <)$ *belongs to* $\textsc{Constant-Time}_{\mathrm{lin}}$ *then* $\textsc{enum}(R, <) \in \textsc{Constant-Delay}_{\mathrm{lin}}$ *and* $\textsc{random}(R) \in \mathrm{AVG-CONST-TIME}_{\mathrm{lin}}$.

Proof. The algorithms of both problems have the same precomputation phase as $\textsc{Jth}(R, <)$, for each input x. Then, the enumeration phase consists in iterating, for each j varying from 0 to $card(R(x)) - 1$, the output phase on input $(r(x), j)$. Similarly, the random phase is the sequence of instructions $v \leftarrow card(R(x))$; $j \leftarrow RANDOM(0 \dots v-1)$; *return* $\mathsf{jth}(R, x, <, j)$. $\square$

[2]Note that imposing constant space for enumeration algorithms is a very strong assumption. From an algorithmic point of view, this corresponds to "oblivious" enumeration algorithms that output new solutions without memorization of previously enumerated solutions (and without repetition). A more liberal use of space (from linear to exponential restrictions) would define seemingly larger complexity classes which are also worth to be studied.

Lemma 2.5. *Let $R, R' \subseteq I \times O$ be two problem relations over the same input and output spaces. Let furthermore $<$ be a linear order on O. If both problems* $\textsc{enum}(R, <)$ *and* $\textsc{enum}(R', <)$ *belong to* $\textsc{Constant-Delay}_{\text{lin}}$*, then so does the union problem* $\textsc{enum}(R \cup R', <)$.

Proof. Obvious and left to the reader. □

3. Logical framework

Basics of first-order logic for finite structures can be found, for example, in [11]. A *signature* σ is a finite set of relation and function symbols of given arities (0-ary function symbols are constant symbols). 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). We denote by $\textsc{fo}$ the class of first-order formulas, and by $\textsc{fo}(\text{qf})$ its quantifier-free fragment.

3.1. Query problems

We are interested in the problems described in the previous section in the context of first-order queries. For each σ-formula $\varphi(\overline{x})$ with k free variables $\overline{x} = (x_1, \ldots, x_k)$ and for each σ-structure S of domain D, we denote by $\varphi(S)$ the set $\{\overline{a} \in D^k : S \models \varphi(\overline{a})\}$. Moreover, we denote by $\mathsf{enum}(\varphi, S, <)$ the increasing enumeration of $\varphi(S)$ according to a given linear order $<$ defined on D^k. The jth element of such an enumeration is denoted by $\mathsf{jth}(\varphi, S, <, j)$. Given a set $\mathcal{C}$ of structures and a linear order $<$ over k-tuples, we define the following problems.

$\textsc{count}(\varphi, \mathcal{C})$
Input: $S \in \mathcal{C}$
Output: $card(\varphi(S))$

$\textsc{random}(\varphi, \mathcal{C})$
Input: $S \in \mathcal{C}$
Output: a random element of $\varphi(S)$

$\textsc{enum}(\varphi, \mathcal{C}, <)$
Input: $S \in \mathcal{C}$
Output: $\mathsf{enum}(\varphi, S, <)$

$\textsc{jth}(\varphi, \mathcal{C}, <)$
Input: $S \in \mathcal{C}$ and $j \in \mathbb{N}$
Output: $\mathsf{jth}(\varphi, S, <, j)$

The most natural linear order for the enumeration and jth solution problems is the lexicographical order $<_{\text{lex}}$ on D^k associated with the natural linear order of the domain $D = \{0, \ldots, n-1\}$ of the input structure S. It will also happen that the linear order involved in these problems depends on the formula φ. We will make this dependence explicit in our notation. Typically, we will later enumerate $\varphi(S)$ according to some special lexicographical order denoted by $<^{\varphi}_{\text{lex}}$ that depends on $\varphi(S)$ and we will study the corresponding problem $\textsc{jth}(\varphi, \mathcal{C}, <^{\varphi}_{\text{lex}})$ (see Def. 4.4).

3.2. Logical definitions

A central notion studied in this article as in [4] is that of *bijective* structure. It is defined as follows.

Definition 3.1. The *arity of a signature* is the maximal arity of its symbols. A *unary signature* has arity 1. Thus, a unary signature is made of constant, monadic predicate and unary function symbols. By a *unary structure* we mean a structure over a unary signature. A *bijective structure* is a unary structure in which all the unary function symbols are interpreted as permutations of the domain. We will assume that any bijective structure contains *at least one* unary function. The class of all bijective structures is denoted by BIJ.

As in [4], we will focus on first-order logic over bijective structures. For convenience, we will handle functions and their reciprocals directly in the syntax. This suggests the following definition.

Definition 3.2. A *bijective term* $\tau(x)$ is of the form $f_1^{\epsilon_1} \ldots f_l^{\epsilon_l}(x)$ where $l \geq 0$, $\epsilon_i = \pm 1$, x is a variable and each $f_i^{\epsilon_i}$ is either the function symbol f_i or its inverse f_i^{-1}.

The notions of *bijective atomic formulas*, *bijective literals* and *bijective first-order formulas* are defined as usual from bijective terms.

We will often denote by τ or $\tau(x)$ a bijective term, according to our intention to ignore or to specify its constitutive variable. We will also make an extensive use of the intermediate notation $\tau(\overline{x})$, which indicates that the constitutive variable of τ *belongs* to the tuple $\overline{x}$. Alternatively, we will point up this fact through the expression "τ is a bijective term *on* $\overline{x}$".

3.3. First-order formulas vs. inductive descriptions

Definition 3.3. Let $\psi_1(\overline{x}), \ldots, \psi_s(\overline{x})$ be first-order formulas with the same variables. The disjunction $\bigvee_{i=1}^{s} \psi_i(\overline{x})$ is said *exclusive* if its disjuncts are mutually exclusive, that is, if $\psi_i(\overline{x})$ implies $\neg\psi_j(\overline{x})$, for all $1 \leq i < j \leq s$.

We will denote by $\curlyvee_{i=1}^{s} \psi_i$ a disjunction to emphasize that it is exclusive. We won't argue for the use of such a writing when its justification is obvious from the context. Clearly, the exclusive disjunction inherits the associativity and commutativity properties from the usual disjunction. In particular, a formula of the form $\curlyvee_{i\in I} \curlyvee_{j\in J} \psi_i^j$ can be written $\curlyvee_{i\in I, j\in J} \psi_i^j$.

We will need a normalization of quantifier-free formulas (Prop. 3.5) in order to prove that first-order formulas fulfil a kind of quantifier elimination result over bijective structures (Cor. 3.6). This normalization involves the notion of *inductive description.* Roughly speaking, an inductive description is a conjunction of bijective literals in which a free variable y can be singled out in such a way that the formula can be written $\psi(\overline{x}) \wedge \theta(\overline{x}, y)$ where ψ is an inductive description and where θ describes in a very simple way the connection between y and the other variables. Let us detail this notion:

Definition 3.4. The set of *inductive descriptions*, denoted by ID, is the smallest set of quantifier-free conjunctive bijective formulas that contains:

(i) quantifier-free conjunctive bijective formulas $\delta(x)$ of arity 1;

(ii) formulas of arity $d+1$ (with $d>0$) of the form

$$\varphi(\overline{x},y) \equiv \psi(\overline{x}) \wedge y = \tau(\overline{x}),$$

where $\psi(\overline{x})$ is an inductive description of arity d and $\tau(\overline{x})$ is a bijective term built on one variable of $\overline{x}$;

(iii) formulas of arity $d+1$ (with $d>0$) of the form

$$\varphi(\overline{x},y) \equiv \psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i\le m} y \neq \tau_i(\overline{x}),$$

where δ and ψ are inductive descriptions of respective arity 1 and d, the $\tau_i(\overline{x})$ are bijective terms, and $\psi(\overline{x})$ logically implies both $\mathrm{PWD}_{i\in[m]}(\tau_i(\overline{x}))$ and $\bigwedge_{i\in[m]} \delta(\tau_i(\overline{x}))$.

(We abbreviate by $\mathrm{PWD}_{i\in[k]}(t_i)$ the formula $\bigwedge_{i\neq j\in[k]} t_i \neq t_j$ stating that the t_i are pairwise distinct.)

Given two sets $\mathcal{L}$, $\mathcal{L}'$ of formulas and a set $\mathcal{C}$ of structures, we write "$\mathcal{L} \subseteq \mathcal{L}'$ on $\mathcal{C}$" if for all $\varphi \in \mathcal{L}$ there exists $\varphi' \in \mathcal{L}'$ such that $\varphi(S) = \varphi'(S)$ for all $S \in \mathcal{C}$. We write "$\mathcal{L} = \mathcal{L}'$ on $\mathcal{C}$" when both $\mathcal{L} \subseteq \mathcal{L}'$ and $\mathcal{L}' \subseteq \mathcal{L}$ hold on $\mathcal{C}$. Recall that FO(qf) denotes the set of quantifier free first-order formulas. Besides, we denote by $\curlyvee$ ID the set of formulas that are exclusive disjunctions of inductive descriptions.

Proposition 3.5. *On bijective structures,* FO(*qf*) $= \curlyvee$ ID.

Proof. It is easily seen, by standard methods in logic, that each quantifier-free formula is equivalent to an exclusive disjunction $\curlyvee_{i=1}^{s} \varphi_i$ where each φ_i is a conjunctive formula (that is, a conjunction of literals). Furthermore, since the logic $\curlyvee$ ID is closed under exclusive disjunction, we can consider without loss of generality that the quantifier-free formula to be written in $\curlyvee$ ID is conjunctive. So let φ be a quantifier-free conjunctive formula. We prove that φ is equivalent to an exclusive disjunction of inductive descriptions by induction on its arity. If arity$(\varphi)=1$, the result is clear. Otherwise, distinguish one (arbitrary) free variable y in φ and write $\varphi \equiv \varphi(\overline{x},y)$ with $y \notin \overline{x}$. Now φ has the shape:

$$\varphi \equiv \bigwedge \pm\alpha_i(\overline{x},y) = \beta_i(\overline{x},y) \wedge \bigwedge \pm U_i(\gamma_i(\overline{x},y)) \tag{1}$$

where all the α_i, β_i, γ_i's are terms built on one variable of $\overline{x},y$ and where the U_i's are unary relation symbols. By authorizing *bijective* terms, we can rewrite φ in such a way that each equality (resp. unequality) involving y and $\overline{x}$ has the shape

$y = \tau(\overline{x})$ (resp. $y \neq \sigma(\overline{x})$), where τ and σ are now bijective terms on $\overline{x}$. Moreover, by sorting the atomic formulas of φ according to the variables involved in them, we can finally write φ under the form:

$$\varphi \equiv \theta(\overline{x}) \wedge \delta(y) \wedge \bigwedge y = \tau_i(\overline{x}) \wedge \bigwedge y \neq \sigma_i(\overline{x}) \qquad (2)$$

where both θ and δ are quantifier-free conjunctive bijective formulas. (The conjuncts $U_i(\gamma_i)$ occurring in (1) have been incorporated in θ or δ, according to the fact that the term γ_i is built on $\overline{x}$ or on y.)

Two cases occur, according to the emptiness of the conjunction of equalities. **If it is not empty, it contains a conjunct** $y = \tau_j(\overline{x})$. Thus, by substituting the term $\tau_j(\overline{x})$ for each occurrence of y in φ, but that of the conjunct $y = \tau_j(\overline{x})$, we get the equivalent form:

$$\varphi \equiv \psi(\overline{x}) \wedge y = \tau_j(\overline{x})$$

where ψ is a quantifier-free conjunctive bijective formula and $\tau_j(\overline{x})$ is a bijective term. By induction hypothesis, ψ is equivalent to an exclusive disjunction of inductive descriptions, say $\psi \equiv \bigvee\!\!\!\!\!\curlyvee_i \psi_i(\overline{x})$. Thus

$$\varphi \equiv \curlyvee_i (\psi_i(\overline{x}) \wedge y = \tau_j(\overline{x}))$$

and we are done, since each disjunct above is an inductive description, by item (ii) of Definition 3.4.

If φ **does not contain any conjunct of the form** $y = \tau(\overline{x})$, equation (2) has the form:

$$\varphi(\overline{x}, y) \equiv \theta(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i=1}^{m} y \neq \sigma_i(\overline{x}). \qquad (3)$$

Here, θ and δ are quantifier-free conjunctive bijective formulas and the $\sigma_i(\overline{x})$ are bijective terms. Let's keep in mind this last formula, that has to be written in $\curlyvee$ ID. To carry on with the normalization of φ, we are going to concentrate for a while on its subformula $\delta(y) \wedge \bigwedge_{i=1}^{m} y \neq \sigma_i(\overline{x})$. More precisely, we are going to evaluate this subformula with respect to the number of distinct σ_i that satisfy δ. Actually, in order to isolate the heart of our reasoning from minor features, we will focus on formulas of the form $\delta(y) \wedge \bigwedge_{i=1}^{m} y \neq v_i$, where the v_i are new first-order variables. Let us first introduce some notations.

For $m > 0$, we denote by $2^{[m]}$ the set of subsets of $[m] = [1, m]$, by $\min P$ the minimal element of a non empty $P \in 2^{[m]}$, and by $\bigcup \mathcal{P}$ the union set $\bigcup_{P \in \mathcal{P}} P$ of any $\mathcal{P} \subseteq 2^{[m]}$. If the elements of some $\mathcal{P} \subseteq 2^{[m]}$ are pairwise disjoint, we say that $\mathcal{P}$ is a *set of disjoint m-subsets* and we note $\mathcal{P} \in \text{SDS}[m]$. Equivalently, $\mathcal{P} \in \text{SDS}[m]$ if $\{(P)_{P \in \mathcal{P}}, [m] \setminus \bigcup \mathcal{P}\}$ is a partition of $[m]$.

Let δ be a first-order formula of arity 1 and $\overline{v} = v_1, \ldots, v_m$ be a tuple of first-order variables. A *δ-type for $\overline{v}$* is a formula $\delta^m_{\mathcal{P}}(\overline{v})$, associated with a given $\mathcal{P} \subseteq 2^{[m]}$, that comprehensively describes a particular behaviour of the variables with respect to δ. Such a formula asserts that, for any $i, j \in [m]$:

(i) if i, j belong to a same $P \in \mathcal{P}$, then v_i, v_j must be interpreted by a same element;
(ii) if i, j belong to different components of $\mathcal{P}$, then v_i, v_j must be interpreted by distinct elements;
(iii) if $i \in \bigcup \mathcal{P}$, then v_i must be interpreted by some elements that fulfil δ;
(iv) if $i \in [m] \setminus \bigcup \mathcal{P}$, then v_i must be interpreted by some element that does not fulfil δ.

For convenience, we formalize these assertions as follows:

$$(i) \quad \bigwedge_{P \in \mathcal{P}} \bigwedge_{i \in P} v_i = v_{\min P} \qquad\qquad (ii) \quad \bigwedge_{P \neq Q \in \mathcal{P}} v_{\min P} \neq v_{\min Q}$$

$$(iii) \quad \bigwedge_{P \in \mathcal{P}} \delta(v_{\min P}) \qquad\qquad (iv) \quad \bigwedge_{i \in [m] \setminus \bigcup \mathcal{P}} \neg\delta(v_i)$$

and we set: $\delta^m_{\mathcal{P}}(v_1, \ldots, v_m) \equiv (i) \wedge (ii) \wedge (iii) \wedge (iv)$. Now, the following remarks follow from the definition of $\delta^m_{\mathcal{P}}$: First, the formula $\bigvee_{\mathcal{P} \in \text{SDS}[m]} \delta^m_{\mathcal{P}}(\overline{v})$ is a tautology and its disjuncts are mutually exclusive. Therefore, $\delta(y) \wedge \bigwedge_{i \in [m]} y \neq v_i$ is equivalent to

$$\left(\bigvee_{\mathcal{P} \in \text{SDS}[m]} \delta^m_{\mathcal{P}}(\overline{v}) \right) \wedge \delta(y) \wedge \bigwedge_{i \in [m]} y \neq v_i,$$

and hence to

$$\bigvee_{\mathcal{P} \in \text{SDS}[m]} \left(\delta^m_{\mathcal{P}}(\overline{v}) \wedge \delta(y) \wedge \bigwedge_{i \in [m]} y \neq v_i \right).$$

Now, each disjunct of the above formula can be written

$$\delta^m_{\mathcal{P}}(\overline{v}) \wedge \delta(y) \wedge \bigwedge_{P \in \mathcal{P}} y \neq v_{\min P}.$$

This is because, under the hypothesis $\delta^m_{\mathcal{P}}(\overline{v}) \wedge \delta(y)$, each v_i such that $i \notin \bigcup \mathcal{P}$ clearly differs from y, since it doesn't satisfy δ while y does, and because for each $P \in \mathcal{P}$, the sets $\{v_i, i \in P\}$ and $\{v_{\min P}\}$ coincide. Therefore we get:

$$\delta(y) \wedge \bigwedge_{i \in [m]} y \neq v_i \;\sim \bigvee_{\mathcal{P} \in \text{SDS}[m]} \left(\delta^m_{\mathcal{P}}(\overline{v}) \wedge \delta(y) \wedge \bigwedge_{P \in \mathcal{P}} y \neq v_{\min P} \right). \tag{4}$$

Notice that $\delta^m_{\mathcal{P}}(\overline{v})$ implies $\textsc{pwd}_{P\in\mathcal{P}}(v_{\min P}) \wedge \bigwedge_{P\in\mathcal{P}} \delta(v_{\min P})$ (immediately follows from the definition of $\delta^m_{\mathcal{P}}$).

Let us now come back to the initial formula $\varphi(\overline{x}, y)$ described in (3), that we aim to normalize. By (4), it can be written:

$$\theta(\overline{x}) \wedge \bigcurlyvee_{\mathcal{P}\in\textsc{sds}[m]} \left(\delta^m_{\mathcal{P}}(\sigma_1(\overline{x}), \dots, \sigma_m(\overline{x})) \wedge \delta(y) \wedge \bigwedge_{P\in\mathcal{P}} y \neq \sigma_{\min P}(\overline{x}) \right),$$

or equivalently:

$$\bigcurlyvee_{\mathcal{P}\in\textsc{sds}[m]} \left(\theta(\overline{x}) \wedge \delta^m_{\mathcal{P}}(\sigma_1(\overline{x}), \dots, \sigma_m(\overline{x})) \wedge \delta(y) \wedge \bigwedge_{P\in\mathcal{P}} y \neq \sigma_{\min P}(\overline{x}) \right).$$

Let us denote by $\psi_{\mathcal{P}}(\overline{x})$ the formula $\theta(\overline{x}) \wedge \delta^m_{\mathcal{P}}(\sigma_1(\overline{x}), \dots, \sigma_m(\overline{x}))$. By induction hypothesis, $\psi_{\mathcal{P}}$ can be written as an exclusive disjunction of inductive descriptions, say $\bigcurlyvee_{i\leq k} \psi^i_{\mathcal{P}}$. Of course, each $\psi^i_{\mathcal{P}}$ implies $\psi_{\mathcal{P}}$, and hence $\delta^m_{\mathcal{P}}(\sigma_1(\overline{x}), \dots, \sigma_m(\overline{x}))$. Therefore, according to the remark that follows equation (4), each $\psi^i_{\mathcal{P}}$ implies $\textsc{pwd}_{P\in\mathcal{P}}(\sigma_{\min P}(\overline{x})) \wedge \bigwedge_{P\in\mathcal{P}} \delta(\sigma_{\min P}(\overline{x}))$. Finally, the initial formula (3) can be written

$$\bigcurlyvee_{\mathcal{P}\in\textsc{sds}[m]} \bigcurlyvee_{i\leq k} \left(\psi^i_{\mathcal{P}}(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{P\in\mathcal{P}} y \neq \sigma_{\min P}(\overline{x}) \right)$$

where each $\psi^i_{\mathcal{P}}$ is an inductive description that implies both $\textsc{pwd}_{P\in\mathcal{P}}(\sigma_{\min P}(\overline{x}))$ and $\bigwedge_{P\in\mathcal{P}} \delta(\sigma_{\min P}(\overline{x}))$. The conclusion follows from item (iii) of Definition 3.4. □

Corollary 3.6. *For each* $(\varphi, S) \in \textsc{fo} \times \textsc{bij}$, *there exists* $\varphi' \in \bigcurlyvee\textsc{id}$ *such that* $\varphi'(S) = \varphi(S)$. *Furthermore, there is some function* f *such that* $|\varphi'| \leq f(|\varphi|)$ *(upper bound independent of* S*) and* φ' *is computable from* (φ, S) *in time* $f(|\varphi|)|S|$.

Proof. We can assume, without loss of generality, that the formula φ of the statement has the form $\exists z\gamma(\overline{x}, z)$, where γ is a quantifier-free formula. By Proposition 3.5, such a γ can be written as a disjunction of inductive descriptions. Now, recall the three forms that a formula of ID can fit, according to Definition 3.4:

(i) $\delta(y)$ with arity$(\delta) = 1$;
(ii) $\psi(\overline{x}) \wedge y = \tau(\overline{x})$, where $\psi(\overline{x}) \in \textsc{id}$;
(iii) $\psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i\leq m} y \neq \tau_i(\overline{x})$, where $\delta, \psi \in \textsc{id}$ and $\psi(\overline{x})$ logically implies $\textsc{pwd}_{i\in[m]}(\tau_i(\overline{x})) \wedge \bigwedge_{i\in[m]} \delta(\tau_i(\overline{x}))$.

Moreover, although it is not emphasized in the statement of Proposition 3.5, this rewriting of γ in $\bigcurlyvee\textsc{id}$ can be done in such a way that in each disjunct that fits the form (ii) or (iii) above, the variable y singularized in both these forms can be arbitrarily chosen[3]. In particular, when we write $\exists z \gamma(\overline{x}, z)$ as $\exists z \bigvee_i \gamma_i$, with the γ_i's in ID, we can ensure that in each γ_i of the form (ii) or (iii), the singularized variable y coincides with z. Consequently, the formula $\exists z \gamma(\overline{x}, z)$ can be written $\bigvee_i \exists y \gamma_i(\overline{x}, y)$, where each γ_i fits one of the forms (i), (ii) or (iii) of Definition 3.4, with y as the singularized variable.

Given a bijective structure S, it is now easy to translate each formula $\exists y \gamma_i(\overline{x}, y)$ into a quantifier-free formula $\tilde{\gamma}_i(\overline{x})$, equivalent on S. This is done according to the form of γ_i:

(i) $\exists y \delta(y)$ becomes false if $\delta(S) = \emptyset$, true otherwise;
(ii) $\exists y\, (\psi(\overline{x}) \wedge y = \tau(\overline{x}))$ becomes $\psi(\overline{x})$;
(iii) $\exists y \left(\psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i \le m} y \neq \tau_i(\overline{x})\right)$ becomes false if $\delta(S)$ contains at most m elements, $\psi(\overline{x})$ otherwise.

The time complexity of this procedure amounts to that of computing $card(\delta(S))$ for some first-order formula $\delta(y)$ of arity 1, which is $O(|\delta|.|S|)$. Hence, the time of the transformation is $O(|S|)$, for a fixed formula. So, φ can be transformed into a quantifier-free formula φ' so that $\varphi(S) = \varphi'(S)$, and, clearly, one gets $|\varphi'| \le g(|\varphi|)$, for some function g. We conclude by applying again Proposition 3.5. □

4. Complexity results on bijective structures

The enumeration, counting and jth solution problems are very easy for queries defined by inductive descriptions as the following proposition asserts.

Proposition 4.1. *For each function $\varphi \in ID$,*

1. COUNT(φ, BIJ) *is computable in linear time ;*
2. Jth$(\varphi, \text{BIJ}, <_{lex}) \in$ CONSTANT-TIME$_{\text{lin}}$ *;*
3. ENUM$(\varphi, \text{BIJ}, <_{lex}) \in$ CONSTANT-DELAY$_{\text{lin}}$.

Proof. Thanks to Lemma 2.4, we just have to prove *1* and *2*. The algorithms described in this proof are recursive. Their behaviour on an input (φ, S) (resp. (φ, S, j)) is directed by the shape of the inductive description φ. That is, the algorithms operate according to modalities that depend on the form (i), (ii) or (iii) of Definition 3.4 that φ matches.

1. COUNT(φ, BIJ). The following recursive procedure computes $card(\varphi(S))$ in time $O(|S|)$ for any $\varphi \in$ ID and $S \in$ BIJ. As mentioned above, it uses the inductive definition of ID.

[3]This possibility is explicitly mentioned in the first paragraph of the proof of Proposition 3.5: when arity$(\varphi) > 1$, the normalization procedure "distinguishes one (arbitrary) free variable y and writes $\varphi \equiv \varphi(\overline{x}, y)$ with $y \notin \overline{x}$".

```
procedure COUNT($\varphi, S$)
    if arity($\varphi$) = 1 then
        compute $\varphi(S)$ ; return $card(\varphi(S))$
    if $\varphi \equiv \psi(\overline{x}) \wedge y = \tau(\overline{x})$ then
        return COUNT($\psi, S$)
    if $\varphi \equiv \psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i<m} y \neq \tau_i$ then
        return COUNT($\psi, S$) $*$ (COUNT($\delta, S$) $- m$)
```

The last instruction of this procedure is justified by the fact that ψ fulfils the implication $\psi(\overline{x}) \rightarrow \text{PWD}(\tau_i, i \leq m) \wedge \bigwedge_{i \leq m} \delta(\tau_i)$.

2. Jth$(\varphi, \text{BIJ}, <_{\text{lex}})$. We design an algorithm that, given $\varphi \in \text{ID}$ and $S \in \text{BIJ}$, performs a preprocessing step in time $O(|S|)$ in such a way that $\mathsf{jth}(\varphi, S, <_{\text{lex}}, j)$ can be afterwards computed in constant time for any $j \in \mathbb{N}$. Both the preprocessing and the output phase are recursive. The following easy considerations will make the proof of the algorithm immediate.

(i) If arity$(\varphi) = 1$, then computing and sorting $\varphi(S)$ can be done in time $O(|S|)$. Afterwards, $\mathsf{jth}(\varphi, S, <_{\text{lex}}, j)$ can be returned in constant time for any j.

(ii) If $\varphi(\overline{x}, y) \equiv \psi(\overline{x}) \wedge y = \tau(\overline{x})$, then $\mathsf{jth}(\varphi, S, <_{\text{lex}}, j) = (\overline{a}, \tau(\overline{a}))$ where $\overline{a} = \mathsf{jth}(\psi, S, <_{\text{lex}}, j)$.

(iii) If $\varphi(\overline{x}, y) \equiv \psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i \leq m} y \neq \tau_i(\overline{x})$, then the result rests on the two following remarks:

• If $(A, <_A)$ and $(B, <_B)$ are two linear orders and $<_{\text{lex}}$ is the associated lexicographic ordering for $A \times B$, then for any $j < card(A).card(B)$:

$$\mathsf{jth}(A \times B, <_{\text{lex}}, j) = (\mathsf{jth}(A, <_A, q), \mathsf{jth}(B, <_B, r)),$$

where $q = j \operatorname{div} card(B)$ and $r = j \bmod card(B)$.

• Let $(A, <)$ be a linear order and F a subset of A. Denote by $a_0, \ldots, a_{p-1}$ (resp. $f_0, \ldots, f_{m-1}$) the strictly increasing enumeration of A (resp. F) according to $<$. Then for any $r < p - m$:

$$\mathsf{jth}(A \setminus F, <, r) = \mathsf{jth}(A, <, s),$$

where s is the result of the following procedure:

$$s \leftarrow r;\ i \leftarrow 0;\ \textbf{while}\ (f_i \leq a_s \text{ and } i < m)\{i\text{++};\ s\text{++}\};\ \textbf{return}\ s.$$

The precomputation and output steps of our algorithm, denoted JTH, for the problem Jth$(\varphi, \text{BIJ}, <_{\text{lex}})$ are described below. We use some instruction of the form $T \leftarrow \mathsf{sort}\, X$. It means "sort the set X and store it in increasing order in a table $T[0, \ldots, card(X) - 1]$". The soundness of JTH proceeds from the above remarks. Besides, it is clearly a CONSTANT-TIME$_{\text{lin}}$ algorithm.

```
procedure JTH.precomp($\varphi, S$)
    compute $card(\varphi(S))$
    if $\varphi$ is a formula $\delta$ of arity 1 then
        compute $\delta(S)$ ; $\Delta \leftarrow \mathsf{sort}\,\delta(S)$
    if $\varphi \equiv \psi(\overline{x}) \wedge y = \tau(\overline{x})$ then
        JTH.precomp($\psi, S$)
    if $\varphi \equiv \psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i<m} y \neq \tau_i$ then
        JTH.precomp($\delta, S$) ; JTH.precomp($\psi, S$)

procedure JTH.output($\varphi, S, j$)
    if $j \geq card(\varphi(S))$ then return error
    else
        if $\varphi$ is a formula $\delta$ of arity 1 then return $\Delta[j]$
        if $\varphi \equiv \psi(\overline{x}) \wedge y = \tau(\overline{x})$ then
            $\overline{a} \leftarrow$ JTH.output($\psi, S, j$) ; return $(\overline{a}, \tau(\overline{a}))$
        if $\varphi \equiv \psi(\overline{x}) \wedge \delta(y) \wedge \bigwedge_{i<m} y \neq \tau_i$ then
            $\beta \leftarrow card(\delta(S)) - m$
            $\overline{a} \leftarrow$ JTH.output($\psi, S, j \operatorname{div} \beta$)
            $\Theta \leftarrow \mathsf{sort}\,\{\tau_1(\overline{a}), \ldots, \tau_m(\overline{a})\}$
            $i \leftarrow 0$ ; $s \leftarrow j \bmod \beta$ ;
            while $\Theta[i] \leq \Delta[s]$ and $i < m$ do {$i$++, $s$++}
            return $(\overline{a}, \Delta[s])$
```

□

Corollary 4.2. *For each* $\varphi \in$ FO, ENUM$(\varphi, \text{BIJ}, <_{lex}) \in$ CONSTANT-DELAY$_{\text{lin}}$.

Proof. Derives from Corollary 3.6, Lemma 2.5 and Proposition 4.1. □

Remark 4.3. This corollary improves Theorem 7 of [4] which states a similar result for some unspecified linear order $<$ instead of $<_{\text{lex}}$.

We are not able to prove that, for each $\varphi \in$ FO, problem Jth$(\varphi, \text{BIJ}, <_{\text{lex}})$ belongs to the class CONSTANT-TIME$_{\text{lin}}$ and we conjecture it does not. However, a similar result holds for a less canonical linear order that depends on the input formula φ.

Definition 4.4. Let $\varphi(\overline{x}) \equiv \bigcurlyvee_{i=1}^{s} \varphi_i(\overline{x})$ be an exclusive disjunction of inductive descriptions $\varphi_i(\overline{x})$ of signature σ and let S be a σ-structure. Let $<_{\text{lex}}^{\varphi}$ denote the "lexicographical" order on $\varphi(S)$ defined as follows: for any $\overline{a}, \overline{b} \in \varphi(S)$: $\overline{a} <_{\text{lex}}^{\varphi} \overline{b}$, if

- either $\overline{a} \in \varphi_i(S)$ and $\overline{b} \in \varphi_j(S)$ for some $i < j \leq s$,
- or $\overline{a}, \overline{b} \in \varphi_i(S)$ for some $i \leq s$ and $\overline{a} <_{\text{lex}} \overline{b}$.

Corollary 4.5. *Let* $\varphi \in \bigcurlyvee$ ID. *Then* Jth$(\varphi, \text{BIJ}, <_{\text{lex}}^{\varphi}) \in$ CONSTANT-TIME$_{\text{lin}}$.

Proof. For any exclusive disjunction of inductive descriptions $\varphi \equiv \bigcurlyvee_{i=1}^{s} \varphi_i$ and any $S \in$ BIJ, it holds: $\mathsf{jth}(\varphi, S, <_{\text{lex}}^{\varphi}, j) = \mathsf{jth}(\varphi_i, S, <_{\text{lex}}, k)$, where i is the smallest index such that $j < \sum_{u=1}^{i} card(\varphi_u(S))$ and where $k = j - \sum_{u=1}^{i-1} card(\varphi_u(S))$.

Together with the algorithm JTH for $\textsc{jth}(\varphi, \textsc{bij}, <_{\text{lex}})$ described above (see p. 161) for $\varphi \in \textsc{id}$, this remark justifies the following algorithm $\mathcal{A}$ for $\textsc{jth}(\varphi, \textsc{bij}, <^{\varphi}_{\text{lex}})$ with $\varphi \in \curlyvee \textsc{id}$:

```
procedure $\mathcal{A}$.precomp($\varphi, S$)
    write $\varphi$ under the form $\curlyvee_{i=1}^{s} \varphi_i$, with each $\varphi_i \in \textsc{id}$
    for $i = 1$ to $s$ do
        JTH.precomp($\varphi_i, S$)
        compute $v_i = \sum_{j=1}^{i} card(\varphi_j(S))$
procedure $\mathcal{A}$.output($\varphi, S, j$)
    if $j \geq v_s$ then return error
    else
        Find $i \leq s$ such that $v_{i-1} \leq j < v_i$
        return JTH.output($\varphi_i, S, j - v_{i-1}$)
```

It is easily seen that $\mathcal{A}$ runs in Constant-Time$_{\text{lin}}$. □

Corollary 4.6. *For each* $\varphi \in \textsc{fo}$*,* $\textsc{random}(\varphi, \textsc{bij}) \in \text{AVG} - \text{CONST} - \text{TIME}_{\text{lin}}$.

Proof. Immediate, from Corollaries 3.6, 4.5 and Lemma 2.4. □

5. Complexity results on structures of bounded degree

Let S be a structure of domain D over a relational signature σ. We say that the *degree* of S is *bounded by* d if, for each symbol $R \in \sigma$, each $a \in D$ belongs to at most d tuples of R. We denote by $\textsc{deg}_d$ the class of relational structures of degree bounded by d. The complexity results we obtain for our query problems on bijective structures immediately imply the same results on structures of bounded degree because of the following lemma of [4]:

Lemma 5.1 (Durand, Grandjean)**.** *Let* $d > 0$ *be a fixed integer. For each* $(\varphi, S) \in \textsc{fo} \times \textsc{deg}_d$*, there exists* $(\varphi', S') \in \textsc{fo} \times \textsc{bij}$ *such that* $\varphi(S) = \varphi'(S')$*. Moreover, the size* $|\varphi'|$*only depends on* $|\varphi|$ *and* d*, and* (φ', S') *is computable from* (φ, d, S) *in time* $O(|S|)$*, that means, in time* $\leq f(|\varphi|, d).|S|$*, for some function* f*.*

Proof. See [4], Proposition 10 and Lemma 11. □

Corollary 5.2. *For each* $d > 0$ *and each* $\varphi \in \textsc{fo}$*, we have:*

(1) $\textsc{enum}(\varphi, \textsc{deg}_d, <_{lex}) \in$ Constant-Delay$_{\text{lin}}$*;*
(2) $\textsc{jth}(\varphi, \textsc{deg}_d, <^{\varphi,d}) \in$ Constant-Time$_{\text{lin}}$*, for some linear order* $<^{\varphi,d}$*;*
(3) $\textsc{random}(\varphi, \textsc{deg}_d) \in$ *AVG− CONST− TIME*$_{\text{lin}}$.

Let $H = (V_H, E_H)$ be a fixed graph over the set $V_H = \{1, \ldots, k\}$. For any graph $G = (V_G, E_G)$, the *H-subgraphs of G* are the tuples $(x_1, \ldots, x_k) \in V_G^k$ such that $\forall i, j \in V_H$: $ij \in E_H \rightarrow x_i x_j \in E_G$. Similarly, the *H-induced subgraphs of G* are the tuples $(x_1, \ldots, x_k) \in V_G^k$ such that $ij \in E_H \leftrightarrow x_i x_j \in E_G$. The results above can be applied to prove that the H-(induced) subgraphs of any graph G of degree $\leq d$ can be enumerated with constant delay after a linear precomputation. More precisely, denote by $\textsc{graph}_d$ the class of graphs of degree bounded by d, and by $\textsc{subgraph}_d^H$ the problem relation (see p. 149) whose input space is $\textsc{graph}_d$ and whose set of solutions on a given input G is the set of H-subgraphs of G. Define similarly the counterpart problem $\textsc{ind-subgraph}_d^H$ related to induced subgraphs. Given $H, G \in \textsc{graph}_d$, the set of tuples $\textsc{subgraph}_d^H(G)$ and $\textsc{ind-subgraph}_d^H(G)$ are easily expressed by the respective first-order formulas:

$$\varphi_H(x_1, \ldots, x_k) \equiv \textsc{pwd}(x_1, \ldots, x_k) \wedge \bigwedge_{ij \in E_H} E(x_i, x_j) \quad \text{and}$$

$$\psi_H(x_1, \ldots, x_k) \equiv \textsc{pwd}(x_1, \ldots, x_k) \wedge \bigwedge_{ij \in E_H} E(x_i, x_j) \wedge \bigwedge_{ij \notin E_H} \neg E(x_i, x_j),$$

where $k = card(V_H)$. Therefore, the results of Corollary 5.2 apply to the enumeration, jth solution and random solution problems associated with the problem relations $\textsc{subgraph}_d^H$ and $\textsc{ind-subgraph}_d^H$.

Corollary 5.3. *For each fixed $d > 0$ and each fixed graph H of degree $\leq d$, we have:*

(1) $\textsc{enum}(\textsc{subgraph}_d^H, <_{lex}) \in \textsc{Constant-Delay}_{\text{lin}}$*,*
(2) $\text{Jth}(\textsc{subgraph}_d^H, <^{H,d}) \in \textsc{Constant-Time}_{\text{lin}}$*, for some linear order* $<^{H,d}$*,*
(3) $\textsc{random}(\textsc{subgraph}_d^H) \in \text{AVG} - \text{CONST} - \text{TIME}_{\text{lin}}$*,*

and the same results hold for $\textsc{ind-subgraph}_d^H$*.*

References

[1] G. Bagan, Mso queries on tree decomposable structures are computable with linear delay, in *Proc. 15th Annual Conference of the EACSL (CSL'06). Lect. Notes Comput. Sci.* **4207** (2006) 167–181.

[2] G. Bagan, A. Durand and E. Grandjean, On acyclic conjunctive queries and constant delay enumeration, in *Proc. 16th Annual Conference of the EACSL (CSL'07). Lect. Notes Comput. Sci.* (2007) 208–222.

[3] B. Courcelle, Linear delay enumeration and monadic second-order logic. *Discrete Appl. Math.* (2007) (to appear).

[4] A. Durand and E. Grandjean, First-order queries on structures of bounded degree are computable with constant delay. *ACM T. Comput. Logic* (2007) 1–18.

[5] A. Durand and F. Olive, First-order queries over one unary function, in *Proc. 15th Annual Conference of the EACSL (CSL'06). Lect. Notes Comput. Sci.* **4207** (2006) 334–348.

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

[7] M. Frick and M. Grohe, Deciding first-order properties of locally tree decomposable structures. *J. ACM* **48** (2001) 1184–1206.

[8] G. Gottlob, N. Leone and F. Scarcello, The complexity of acyclic conjunctive queries. *J. ACM* **48** (2001) 431–498.

[9] S. Grigorieff, Décidabilité et complexité des théories logiques. *Collection Didactique INRIA* **8** (1991) 7–97.

[10] E. Grandjean and T. Schwentick, Machine-independent characterizations and complete problems for deterministic linear time. *SIAM J. Comput.* **32** (2002) 196–230.
[11] L. Libkin, *Elements of finite model theory.* EATCS Series, Springer (2004).
[12] S. Lindell, A normal form for first-order logic over doubly-linked data structures. *Int. J. Found. Comput. Sci.* (2006) (to appear).
[13] R. Motwani and P. Raghavan, *Randomized algorithms.* Cambridge University Press (1995).
[14] C. Papadimitriou and M. Yannakakis, On the complexity of database queries. *J. Comput. Syst. Sci.* **58** (1999) 407–427.
[15] M.Y. Vardi, On the complexity of bounded-variable queries. *Proc. Principles of Databases Systems (PODS'95)*, ACM Press (1995) 266–276.
[16] D. Seese, Linear time computable problems and first-order descriptions. *Math. Structures Comput. Sci.* **6** (1996) 505–526.
[17] M. Yannakakis, Algorithms for acyclic database schemes. *Proc. Very Large Data Bases Conference (VLDB'81)* (1981) 82–94.