First-order queries on structures of bounded degree are computable with constant delay

Arnaud Durand, Etienne Grandjean

▶ **To cite this version:**

Arnaud Durand, Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. ACM Transactions on Computational Logic, 2007, 8 (4), pp.1-18. hal-00195016

HAL Id: hal-00195016
https://hal.science/hal-00195016

Submitted on 8 Dec 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

First-order queries on structures of bounded degree are computable with constant delay

ARNAUD DURAND
Université Paris 7 - Denis Diderot
and
ETIENNE GRANDJEAN
Université de Caen

A relational structure is d-degree-bounded, for some integer d, if each element of the domain belongs to at most d tuples. In this paper, we revisit the complexity of the evaluation problem of not necessarily Boolean first-order (**FO**) queries over d-degree-bounded structures. Query evaluation is considered here as a dynamical process. We prove that any **FO** query on d-degree-bounded structures belongs to the complexity class CONSTANT-DELAY$_{lin}$, i.e., can be computed by an algorithm that has two separate parts: it has a precomputation step of time linear in the size of the structure and then, it outputs all solutions (i.e. tuples that satisfy the formula) one by one with a constant delay (i.e. depending on the size of the formula only) between each. Seen as a global process, this implies that queries on d-degree-bounded structures can be evaluated in total time $f(|\varphi|).(|\mathcal{S}| + |\varphi(\mathcal{S})|)$ and space $g(|\varphi|).|\mathcal{S}|$ where $\mathcal{S}$ is the structure, φ is the formula, $\varphi(\mathcal{S})$ is the result of the query and f, g are some fixed functions.

Among other things, our results generalize a result of Seese on the data complexity of the model-checking problem for d-degree-bounded structures. Besides, the originality of our approach compared to related results is that it does not rely on the Hanf's model-theoretic technique and is simple and informative since it essentially rests on a quantifier elimination method.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computational Logic*; F.1.3 [**Computation by Abstract Devices**]: Complexity Measures and Classes

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Computational Complexity, Enumeration Problems, First-Order Logic

Introduction

Evaluating the expressive power of logical formalisms is an important task in theoretical computer science. It has many applications in numerous fields such as complexity theory, verification or databases. In this latter case, it often amounts

Author's Address : A. Durand, Équipe de Logique Mathématique - CNRS UMR 7056, UFR de mathématiques, Université Paris 7 - Denis Diderot, 2 place jussieu F-75251 Paris cedex 05, France. Email: durand@logique.jussieu.fr
E. Grandjean, GREYC - CNRS UMR 6072, Université de Caen - Campus 2, F-14032 Caen cedex - France. Email: grandjean@info.unicaen.fr
Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 20YY ACM 1529-3785/YY/00-0001 $5.00

to determining how difficult it is to compute a query written in a given language. In this vein, determining which fragments of first-order logic define tractable query languages has deserved much attention.

It is well known, that over an arbitrary signature, computing a first-order query can be done in time polynomial in the size of the structure (and even in logarithmic space and AC^0 [Vardi 1995; Libkin 2004]). However the exponent of this polynomial depends heavily on the formula size (more precisely, on the number of variables). Nevertheless, for particular kinds of structures or formulas the complexity bound can be substantially improved.

A *d-degree-bounded structure* is a relational structure $\langle D; R_1, \ldots, R_q \rangle$ where each element of D belongs to at most d tuples of each relation R_i. In [Seese 1996], it is proved that checking if a given first-order sentence φ is true (i.e., the Boolean query or model-checking problem) in a d-degree-bounded structure $\mathcal{S}$ can be done in linear time in the size of $\mathcal{S}$. The method used to prove this result relies on model-theoretic techniques (threshold equivalence of structures for isomorphism types and locality of first-order logic, see [Hanf 1965; Gaifman 1982; Libkin 2004]). It is perfectly constructive but hardly implementable. Later, still using such methods, several other tractability results have been shown for the complexity of the model-checking of first-order formulas over structures or formulas that admit nice (tree) decomposition properties (see [Flum et al. 2002]).

The main goal of this paper is to revisit the complexity of the evaluation problem of not necessarily Boolean first-order queries over d-degree bounded structures. We regard query evaluation as a *dynamical process*. Instead of considering the cost of the evaluation globally, we measure the delay between the outputs of two consecutive tuples, i.e., query problems are viewed as enumeration problems. This latter kind of problems appears widely in many areas of computer science (see for example [Eiter and Gottlob 1995; Eiter et al. 2003; Boros et al. 2000; Kavvadias et al. 2000; Goldberg 1994] or [Johnson et al. 1988] for basic complexity notions on enumeration). However, to our knowledge the relationship to query evaluation has not been investigated so far.

We prove that any query on d-degree-bounded structures belongs to the complexity class CONSTANT-DELAY$_{lin}$, i.e., can be computed by an algorithm that has two separate parts: it has a precomputation step of which time complexity is linear in the size of the structure and then, outputs all the solution tuples one by one with a constant delay (i.e., depending on the size of the formula only) between two successive tuples and between the last one and the signal indicating that the computation is terminated. Seen as a global process, this implies that queries on bounded degree structures can be evaluated in total time $f(|\varphi|).(|\mathcal{S}| + |\varphi(\mathcal{S})|)$ and space $g(|\varphi|).|\mathcal{S}|$ where $|\mathcal{S}|$ is the size of the structure $\mathcal{S}$, $|\varphi|$ is that of the formula φ, $|\varphi(\mathcal{S})|$ is the size of the result $\varphi(\mathcal{S})$ of the query and f, g are some fixed functions. As a corollary, it implies that the time complexity of the model-checking problem is $f(|\varphi|).|\mathcal{S}|$ thus providing an alternative proof of the result of [Seese 1996].

The main method used in this paper does not rely on model-theoretic techniques as previous results of the same kind (see, for example, [Seese 1996] or [Lindell 2005] for a generalization to least-fixed point formulas). Instead, we develop a very simple quantifier elimination method suitable for bijective unary functions and apply it to

obtain our complexity bound. An advantage of this method is that it is effective and informative. Another advantage is that our paper is completely self-contained.

Besides, the class CONSTANT-DELAY$_{lin}$ is interesting by itself and is, to our knowledge, a new complexity class for enumeration problems: as proved for linear time complexity (the class DLIN studied in [Grandjean and Schwentick 2002]) it can be shown that CONSTANT-DELAY$_{lin}$ is a robust class and is in some sense the minimal robust complexity class of enumeration problems.

The paper is organized as follows. First, basic definitions are given in Section 1. In particular, in Subsection 1.3, we recall definitions about enumeration problems, introduce the notion of constant delay computation and prove some basic properties about it. In Section 2, the quantifier elimination method is introduced and is applied to the evaluation problem of first-order formulas over structures with bijective unary functions, for short *bijective structures*, which are d-degree-bounded structures of a special kind. In Section 3, using classical logical interpretation techniques, this later problem is shown equivalent in linear time to the first-order query problem over d-degree-bounded structures thus providing the same bound for it. Finally, in Subsection 3.3, consequences about the complexity of the subgraph (resp. induced subgraph) isomorphism problem are given.

1. DEFINITIONS

1.1 Logical definitions and query problems

We suppose the reader to be familiar with the basic notions of first-order logic. A *signature* σ is a finite set of relational and functional symbols of given arities (0-ary function symbols are constants symbols). The arity of σ is the maximal arity of its symbols. The signature σ is *unary* if all its symbols are of arity at most one.

A (finite) σ-structure consists of a domain D together with an interpretation of each symbol of σ over D (the same notation is used here for each signature symbol and its interpretation).

In this paper, we will distinguish between two kinds of signatures on which semantical restrictions on their possible interpretation are imposed:

—Either σ is made of constant symbols, monadic (i.e., unary) relation symbols and unary function symbols of which interpretation is restricted to bijective functions (i.e., permutations),

—Or σ contains relation symbols only of which degrees are bounded by some given constant d (detailed definitions about d-degree bounded relations are delayed till section 3).

Class of structures defined by either of these semantical restrictions will be said to be of *bounded degree* [1].

In what follows we make precise notions and problems about first-order logic over bijective structures. The main motivation of our study of the first-order theory of bijective structures is that is allows a very simple quantifier elimination procedure.

[1]Note that bijective functions can be seen as special cases of d-bounded degree relations, so we have essentially one notion of bounded degree structure

DEFINITION 1.1. *Let $\sigma = \{c_1, \ldots, c_p, U_1, \ldots, U_q, f_1, \ldots, f_k\}$ be a unary signature consisting of constant symbols c_i, of monadic predicates U_i and of unary function symbols f_i, $i = 1, \ldots, k$. A* bijective σ-structure *is a σ-structure $\mathcal{S}$ of the form $\mathcal{S} = \langle D; c_1, \ldots, c_p, U_1, \ldots, U_q, f_1, \ldots, f_k \rangle$ where each f_i is a permutation of the domain D.*

This paper provides a quantifier elimination method for first-order formulas interpreted in bijective structures. As it is usual for such kind of result, the elimination will be done in a richer language. The following definition is required.

DEFINITION 1.2. *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}. Similarly, the term $\tau^{-1}(x)$ denotes the inverse of the term $\tau(x)$.*

A bijective atomic formula *is of one of the following four forms where $\tau(x), \tau_1(x), \tau_2(x)$ are bijective terms:*

—*either a bijective equality $\tau_1(x) = \tau_2(y)$,*

—*or $\tau(x) = c$ where c is a constant symbol,*

—*or $U(\tau(x))$ where U is a monadic predicate,*

—*or a cardinality statement $\exists_x^k \Psi(x)$ where the quantifier $\exists_x^k$ is interpreted as "there exist at least k values of x such that" and $\Psi(x)$ is a Boolean combination of bijective atomic formulas over variable x only.*

A bijective literal is a bijective atomic formula or its negation.

A bijective first-order σ-formula [2] *is a first-order formula built over bijective atomic formulas of some unary signature σ.*

It is essential to notice that this richer language is exactly as expressive as first-order logic on bijective structures. As the inverse of each function symbol can be used, each bijective equality $\tau(x) = \tau_1(y)$ can be rephrased as $\tau_2(x) = y$ where $\tau_2(x) = \tau_1^{-1}\tau(x)$.

Let $\varphi(\overline{t})$ and $\varphi'(\overline{t})$ be two σ-formulas with free variables in the list $\overline{t} = (t_1, \ldots, t_k)$. Formulas $\varphi(\overline{t})$ and $\varphi'(\overline{t})$ are *equivalent* if for all σ-structures $\mathcal{S}$ and all tuples $\overline{a} = (a_1, \ldots, a_k)$ of elements of the domain it holds that:

$$(\mathcal{S}, \overline{a}) \models \varphi(\overline{t}) \text{ iff } (\mathcal{S}, \overline{a}) \models \varphi'(\overline{t}).$$

One of the main problems studied in this paper is the following.

QUERY-BIJ(**FO**)
Input: a unary functional signature σ, a bijective σ-structure $\mathcal{S}$ of domain D and a first-order bijective σ-formula $\varphi(\overline{x})$ with free variables in the list $\overline{x} = (x_1, \ldots, x_k)$
Parameter: φ
Output: $\varphi(\mathcal{S}) = \{\overline{a} \in D^k : (\mathcal{S}, \overline{a}) \models \varphi(\overline{x})\}$.

The corresponding Boolean query problem (the subproblem where $k = 0$, i.e., φ is a sentence) is often called a model-checking problem. It will be denoted by

[2] Note that, of course, the notion of "bijective formula" is an abuse of language: this is motivated by the use of inverse function symbols f^{-1} which makes sense only if the interpretation is restricted to bijections

MC-BIJ(**FO**) here. As suggested by the formulation of the query problem, we are interested in its parameterized complexity and the complexity results given here consider the size of the query formula φ as the parameter (see [Downey and Fellows 1999]).

1.2 Model of computation and measure of time

The model of computation used in this paper is the Random Access Machine (RAM) with uniform cost measure (see [Aho et al. 1974; Grandjean and Schwentick 2002; Grandjean and Olive 2004; Flum et al. 2002]). Query problems are the main subject of this paper, an instance of such a problem always consists of two kinds of objects: a first-order structure and a first-order formula.

The *size* $|I|$ of an object I is the number of registers used to store I in the RAM in a natural way. If E is the set $\{1, \ldots, n\}$, $|E| = card(E) = n$. If $R \subseteq D^k$ is a k-ary relation over domain D, with $|D| = card(D)$, then $|R| = k.card(R)$: all the tuples $(x_1, \ldots, x_k)$ for which $R(x_1, \ldots, x_k)$ holds must be stored, each in a separate k-tuple of registers. Similarly, if f is a unary function from D to D, all the values $f(x)$ must be stored and, as a consequence $|f| = |D|$.

If φ is a first-order formula, $|\varphi|$ is the length of φ, i.e., the number of occurrences of variables, relation or function symbols and syntactic symbols: $\exists, \forall, \wedge, \vee, \neg, =$,"(","(")",",",".

All the problems we consider in this paper are parameterized problems: they take as input a list of objects made of a σ-structure $\mathcal{S}$ and a formula φ and as output the result of the query $\varphi(\mathcal{S})$. Due to the much larger size, in practice, of the structure $\mathcal{S}$ than the size of formula φ, $|\mathcal{S}| >> |\varphi|$, this latter one, $|\varphi|$, is considered here as the parameter.

A problem **P** is said to be computable in time $f(|\varphi|).T(|\mathcal{S}|, |\varphi(\mathcal{S})|)$ for some functions $f : \mathbb{N} \to \mathbb{N}$ and $T : \mathbb{N}^2 \to \mathbb{N}$ if there exists a RAM that computes **P** in time (i.e., the number of instructions performed) bounded by $f(|\varphi|).T(|\mathcal{S}|, |\varphi(\mathcal{S})|)$ using space, i.e., addresses and register contents also bounded by $f(|\varphi|).T(|\mathcal{S}|, |\varphi(\mathcal{S})|)$. The notation $O_\varphi(T(|\mathcal{S}|, |\varphi(\mathcal{S})|))$ is used as an abbreviation when one does not want to make precise the value of function f. It is also assumed that the function T is at least linear and at most polynomial. To give an example and to relate our complexity measure to the logarithmic cost measure, in case T is linear, i.e., $T(n, p) = O(n + p)$, it is easy to see that the number of bits manipulated by the RAM is really linear in the number of bits needed to encode the input and the output.

1.3 Enumeration algorithms and constant delay computations

In this section, A is a binary predicate (and is seen as an abstraction for a two-inputs algorithmic problem). Enumeration problems will be defined by reference to such a generic predicate.

DEFINITION 1.3. *Given a binary predicate A, the enumeration problem* ENUM·A *associated with A is defined as follows. For each input x:*

$$\text{ENUM}\cdot A(x) = \{y \ : \ A(x, y) \ \textit{holds} \ \},$$

i.e. ENUM$\cdot A(x)$ *is the solution set for input* x.

REMARK 1.4. *Query problems may be seen as enumeration problems. The input* x *consists of the structure* $\mathcal{S}$ *and the formula* $\varphi(\overline{x})$*, a witness* y *is a tuple* $\overline{a}$ *and evaluating predicate* A *amounts to checking whether* $(\mathcal{S}, \overline{a}) \models \varphi(\overline{x})$.

One may consider the delay between two consecutive solutions as an important point in the complexity of enumeration problems. In [Johnson et al. 1988] several complexity measures for enumeration have been defined. One of the most interesting ones is that of a *polynomial delay* algorithm. An algorithm $\mathcal{A}$ is said to compute an enumeration problem ENUM$\cdot A$ within *polynomial delay* if, for each input x, it computes the set ENUM$\cdot A(x)$ in any order with no repetition and no more than a (fixed) polynomial delay between two consecutive solutions it outputs (and no more than a polynomial delay to output the first solution and between the last solution and the end of the algorithm). *Polynomial delay* is often considered as the right notion of feasability for enumeration problems. In this paper, we introduce a more demanding complexity measure that forces *constant delay* between outputs.

DEFINITION 1.5. *An enumeration problem* ENUM$\cdot A$ *is said to be computable within constant delay and with linear precomputation, which is written* ENUM$\cdot A \in$ CONSTANT-DELAY$_{lin}$*, if there exists a RAM algorithm* $\mathcal{A}$ *which, for any input* x*, enumerates the set* ENUM$\cdot A(x)$ *and satisfies the following properties.*

(*1*) $\mathcal{A}$ *uses space* $O(|x|)$

(*2*) $\mathcal{A}$ *can be decomposed into the two following successive steps*
- (*a*) PRECOMP$(\mathcal{A})$ *which performs some precomputations in time* $O(|x|)$*, and*
- (*b*) ENUM$(\mathcal{A})$ *which outputs all solutions without repetition within a delay bounded by some constant* DELAY$(\mathcal{A})$ *which does not depend on input* x*. This delay applies between two consecutive solutions and after the last one.*

By allowing polynomial time precomputations (and polynomial space) instead of linear time (and linear space), one defines a similar but larger class called CONSTANT-DELAY$_{poly}$.

REMARK 1.6. *As proved for the linear time class* DLIN *(see [Grandjean and Schwentick 2002]), it can be shown that the complexity class* CONSTANT-DELAY$_{lin}$ *is robust, i.e., that it is not modified if the set of allowed operations and statements of the RAMs is changed in several reasonable ways. This is because linear time (and linear space) precomputations give the ability to precompute the tables of new allowed operations.*

REMARK 1.7 CONSTANT DELAY AND PARAMETERIZED PROBLEMS. *In this paper, we focus on query problems. As already written, the input formally consists of a structure* $\mathcal{S}$ *and a formula* φ *and the size of* φ *is considered as a parameter in the complexity evaluation of the query problem. The natural approach of constant delay enumeration for parameterized problems seems to be the following: the size of the input* x *is* $|\mathcal{S}|$ *and the constant delay depends on the parameter (i.e., on* $|\varphi|$*, for a query problem) only.*

The following result is immediate, it evaluates the total time cost of any constant delay algorithm.

LEMMA 1.8. *Let* ENUM$\cdot A$ *be an enumeration problem belonging to the class* CONSTANT-DELAY$_{lin}$*, then, for any input* x*, the set* ENUM$\cdot A(x)$ *can be computed in* $O(|x| + |\text{ENUM}\cdot A(x)|)$ *total time (i.e., in time linear in the sum of the sizes of the input and the output) and space* $O(|x|)$*.*

The two lemmas below give basic properties of constant delay computations.

LEMMA 1.9. *An enumeration problem* ENUM$\cdot A$ *computable in time* $O(|x|)$ *belongs to* CONSTANT-DELAY$_{lin}$*.*

PROOF. For any input x, one only has to compute and store the set ENUM$\cdot A(x)$. This step can be viewed as the precomputation part of the algorithm running in time and space $O(|x|)$. Then, one has to enumerate one by one the elements of the set ENUM$\cdot A(x)$. This is obviously a constant delay process. □

LEMMA 1.10. *Let* ENUM$\cdot A$ *and* ENUM$\cdot B$ *be two disjoint enumeration problems, i.e., such that, for any input* x*,* ENUM$\cdot A(x) \cap$ ENUM$\cdot B(x) = \emptyset$*. Define the (union) enumeration problem* ENUM$\cdot(A \cup B)$ *as follows:*

$$\textit{for any } x, \text{ ENUM}\cdot(A \cup B)(x) = \{y \ : \ A(x,y) \textit{ or } B(x,y) \textit{ holds } \}.$$

If ENUM$\cdot A$ *and* ENUM$\cdot B$ *belong to* CONSTANT-DELAY$_{lin}$ *then, the problem* ENUM$\cdot(A \cup B)$ *also belongs to* CONSTANT-DELAY$_{lin}$*.*

PROOF. Due to the disjointness of the two solution sets for any input, the proof is evident. Let $\mathcal{A}$ and $\mathcal{B}$ be the constant delay algorithms for problems ENUM$\cdot A$ and ENUM$\cdot B$ respectively. The following algorithm correctly computes the problem ENUM$\cdot(A \cup B)$.

Algorithm 1 Constant delay algorithm for ENUM$\cdot A \cup B$

1: **Input:** x
2: PRECOMP($\mathcal{A}$); PRECOMP($\mathcal{B}$)
3: ENUM($\mathcal{A}$); ENUM($\mathcal{B}$)

Obviously, the delay is bounded by the maximum of DELAY($\mathcal{A}$) and DELAY($\mathcal{B}$). □

REMARK 1.11. *Note that the disjointness condition in the lemma above is not always necessary. In case there exist a total ordering* $<$ *and constant delay enumeration algorithms for* ENUM$\cdot A$ *and* ENUM$\cdot B$ *that enumerate solutions in the common order* $<$ *then, it is easily seen that* ENUM$(A \cup B)$ *belongs also to* CONSTANT-DELAY$_{lin}$ *even if the disjointness condition is not satisfied.*

2. FIRST-ORDER QUERIES ON BIJECTIVE STRUCTURES

2.1 Quantifier elimination on bijective structures

The key result of this paper consists in a quantifier elimination method for first-order formulas on bijective structures.

THEOREM 2.1. *Each bijective first-order formula is equivalent to a Boolean combination of bijective atomic formulas.*

In the special case where the formula is a sentence (i.e., has no free variable) then, it is equivalent to a Boolean combination of cardinality statements.

PROOF. As universal quantification can be expressed in terms of existential quantification and negation, we only have to consider elimination of existentially quantified variables. W.l.o.g., we consider formulas in disjunctive normal form and, as existential quantification commutes with disjunction we may consider the case of the elimination of a single existentially quantified variable y in a formula of the form:

$$\varphi(\overline{x}) \equiv \exists y \ (\alpha_1 \wedge \ldots \wedge \alpha_r) \tag{1}$$

where each α_i is a bijective literal with variables among $\overline{x}$ and y. Literals depending on $\overline{x}$ only and cardinality statements need not be considered since they do not involve y, and so $\varphi(\overline{x})$ may be supposed to be of the following form:

$$\varphi(\overline{x}) \equiv \exists y \ [\psi(y) \wedge y =_{\epsilon_1} \tau_1(x_{i_1}) \wedge \ldots \wedge y =_{\epsilon_k} \tau_k(x_{i_k})] \tag{2}$$

where each formula $y =_{\epsilon_j} \tau_j(x_{i_j})$, with $\epsilon_j = \pm 1$, is $y = \tau_j(x_{i_j})$ if $\epsilon_j = 1$ and is $y \neq \tau_j(x_{i_j})$ if $\epsilon_j = -1$ and ψ is a conjunction of bijective literals. To eliminate the quantified variable y we consider two cases.

First, suppose that there is at least one index j such that $\epsilon_j = 1$. In this case, the equality $y = \tau_j(x_{i_j})$ is used to replace each occurrence of y in the formula by the term $\tau_j(x_{i_j})$. The process results in an equivalent quantifier-free formula $\varphi'(\overline{x})$ without variable y.

The second possibility leads to a more complicated replacement scheme. Suppose that for every j, $\epsilon_j = -1$. Then,

$$\varphi(\overline{x}) \equiv \exists y \ [\psi(y) \wedge \bigwedge_{j \leq k} y \neq \tau_j(x_j)] \tag{3}$$

(For simplicity of notations but w.l.o.g. we have supposed that $i_j = j$ for $j = 1, \ldots, k$). The basic idea is now the following : suppose $h \leq k$ is the number of distinct values among those of the k terms $\tau_j(x_j)$ such that $\psi(\tau_j(x_j))$ is true; then, formula $\varphi(\overline{x})$ is true if and only if the number of elements b such that $(\mathcal{S}, b) \models \psi(y)$ holds is strictly greater than h (i.e., $\exists_y^{h+1} \psi(y)$ is true). Introducing (new) cardinality statements in the formula, $\varphi(\overline{x})$ can be equivalently rephrased as the following Boolean combination of bijective atomic formulas:

$$\varphi(\overline{x}) \equiv \bigvee_{h=0}^{k} \bigvee_{P \subseteq [k], Q \subseteq P, |Q|=h} \left[\bigwedge_{j \in Q} \psi(\tau_j(x_j)) \wedge \bigwedge_{i \in P} \bigvee_{j \in Q} \tau_i(x_i) = \tau_j(x_j) \wedge \bigwedge_{j \in [k] \setminus P} \neg \psi(\tau_j(x_j)) \wedge \exists_y^{h+1} \psi(y) \right] \tag{4}$$

where $[k] = \{1, \ldots, k\}$. The formula in brackets is a conjunction of four parts: the first three parts express that there are *exactly h distinct* values $\tau_j(x_j)$ in the set $\psi(\mathcal{S})$.

More generally, starting from a prenex bijective first-order formula, one eliminates all quantified variables except one from the innermost to the outermost one. This will result in an equivalent Boolean combination of bijective *atomic* formulas. In the case where the formula is a sentence, it is easily seen that the elimination process results in a Boolean combination of cardinality statements (note that, of course, $\exists x \varphi(x)$ can be rewritten as $\exists^1_x \varphi(x)$). □

An interesting consequence of Theorem 2.1 is the following result.

COROLLARY 2.2 SEESE [SEESE 1996]. *The problem* MC-BIJ(**FO**) *i.e. of deciding whether a bijective structure $\mathcal{S}$ satisfies a first-order sentence φ is decidable in time $O_\varphi(|\mathcal{S}|)$.*

PROOF. From Theorem 2.1, we know that there exists a Boolean combination of cardinality statements over the same signature σ which is equivalent to φ. Given a cardinality statement $\exists^k_x \Psi(x)$, where Ψ is quantifier-free, one can test whether a given σ-structure $\mathcal{S}$ satisfies $\mathcal{S} \models \exists^k_x \Psi(x)$ in time $O_\Psi(|\mathcal{S}|)$: it suffices to enumerate all the elements a of the domain, test whether $(\mathcal{S}, a) \models \Psi(x)$ in constant time and count those for which the answer is positive. If this number is greater than or equal to k then $\exists^k_x \Psi(x)$ is true in $\mathcal{S}$. The final answer for φ is given by the Boolean combination of the answers for each cardinality statement. □

2.1.1 *Considerations on an efficient implementation of the algorithm.* Compared to the method of [Seese 1996], the proofs given in this paper are much simpler and more informative: on bijective structures any first-order sentence is equivalent to a Boolean combination of cardinality statements which are very simple "one variable" sentences. But, due to the case of Formula 3 in the proof of Theorem 2.1 which leads to the equivalent Formula 4 the whole process requires time $f(|\varphi|).|\mathcal{S}|$ for some function f that may be a tower of exponentials. It can be shown that it heavily depends on the number of variables and, more precisely, on the number of quantifier alternations of the formula [3]. However, the function f can be substantially reduced in case there are few quantifier alternations.

In this subsection, we revisit the method of the proof of Theorem 2.1 to prove a slightly different result in a specific case. We focus on formulas with existentially quantified variables only and show that the model-checking problem for such formulas can be efficiently solved. A first-order sentence is in $\mathbf{FO}^\exists$ if it is of the form $\exists \overline{y}\ \Psi(\overline{y})$, where $\Psi(\overline{y})$ is quantifier-free and in disjunctive normal form (DNF).

COROLLARY 2.3. *The model-checking problem for $\mathbf{FO}^\exists$ formulas on bijective structures (i.e. testing whether $\mathcal{S} \models \varphi$ for some sentence $\varphi \in \mathbf{FO}^\exists$ and some bijective structure $\mathcal{S}$) can be solved in time $O(|\varphi|^d.|\mathcal{S}|)$ where d is the number of distinct variables of φ.*

PROOF. The result obviously holds for $d = 1$. So, assume $d > 1$. For the same reason as in Theorem 2.1, we may consider any formula of the form:

[3] A careful examination shows that the height of the tower is linear in the number of quantifier alternations

$$\varphi(\overline{x}) \equiv \exists y\ (\alpha_1 \wedge \ldots \wedge \alpha_r) \tag{5}$$

where each α_i is a bijective literal [4] with variables among $\overline{x}$ and y. For sake of completeness here, we consider also bijective literals not containing y. Then, $\varphi(\overline{x})$ is of the form:

$$\varphi(\overline{x}) \equiv \exists y\ [\psi(y) \wedge y =_{\epsilon_1} \tau_1(x_{i_1}) \wedge \ldots \wedge y =_{\epsilon_k} \tau_k(x_{i_k}) \wedge \gamma(\overline{x})] \tag{6}$$

where notation ϵ_j is as in the proof of Theorem 2.1 and γ, ψ are conjunctions of bijective literals. Again, if $\epsilon_j = 1$, for some j, then all the occurrences of y are replaced by $\tau_j(x_{i_j})$ and $\varphi(\overline{x})$ is equivalent to a conjunction of literals without the variable y.

Suppose now that $\epsilon_j = -1$ for all $j \leq k$. Let $A = \{a \in D : (\mathcal{S}, a) \models \psi(y)\}$. Since $\psi(y)$ is quantifier-free, A can be computed in time $O(|\psi|.|\mathcal{S}|)$. Two cases need to be considered now. If $|A| > k$, since there are at most k different values $\tau_j(x_j)$ for $j = 1, \ldots, k$, then the conjunction $\exists y[\psi(y) \wedge y \neq \tau_1(x_{i_1}) \wedge \ldots \wedge y \neq \tau_k(x_{i_k})]$ is always true and $\varphi(\overline{x})$ is simply equivalent to $\gamma(\overline{x})$. If $|A| \leq k$ let $A = \{a_1, \ldots, a_h\}$, with $h \leq k$. Formula $\varphi(\overline{x})$ is replaced by the equivalent formula below over the richer signature $\sigma \cup \{a_1, \ldots, a_h\}$:

$$\bigvee_{i \leq h} (\bigwedge_{j \leq k} a_i \neq \tau_j(x_{i_j}) \wedge \gamma(\overline{x}))$$

In all cases, the formula obtained is also in DNF. Time $O(|\varphi|.|\mathcal{S}|)$ is needed to eliminate the variable y and the new formula is of size bounded by $O(k.|\varphi|)$, hence of size $O(|\varphi|^2)$. Elimination of all the d existentially quantified variables except the last one can be pursued from this new formula (without need for a normalization). In the worst case (where all literals are of the form $x_i \neq \tau_1(x_j)$), the process will result in a disjunction of less than $|\varphi|^{d-1}$ conjunctions of at most $|\varphi|$ literals. □

2.2 Constant delay algorithm for first-order queries on bijective structures

We are now ready to state the main result of this section.

THEOREM 2.4. *The problem* QUERY-BIJ(**FO**) *belongs to* CONSTANT-DELAY$_{lin}$. *In particular, it follows from Lemma 1.8 that this problem can be solved in time* $O_\varphi(|\mathcal{S}| + |\varphi(\mathcal{S})|)$ *and space* $O_\varphi(|\mathcal{S}|)$.

Before proving Theorem 2.4, we establish the following lemma.

LEMMA 2.5. *The following problem belongs to the class* CONSTANT-DELAY$_{lin}$*: given a conjunction* Ψ *of bijective literals and a bijective structure* $\mathcal{S}$ *over the same vocabulary, compute* $\Psi(\mathcal{S})$.

PROOF. The result is proved by induction on k the number of (free) variables of $\Psi(\overline{x})$ where $\overline{x} = (x_1, \ldots, x_k)$. We even assume that Ψ makes use of explicit constants from the domain D of $\mathcal{S}$, i.e. elements of D.

[4] In this proof, bijective literals do not involve cardinality statements

For the case $k = 1$, it is evident that the set $Q = \{a \in D : (\mathcal{S}, a) \models \Psi(x)\}$ can be evaluated in time [5] $O_\Psi(|D|) = O_\Psi(|\mathcal{S}|)$ and hence, by Lemma 1.9, belongs to CONSTANT-DELAY$_{lin}$.

The result is supposed to be true for k ($k \geq 1$) and is proved now for $k+1$. Let's consider the query:

$$Q = \{(\overline{a}, b) \in D^{k+1} : \mathcal{S} \models \Psi(\overline{x}, y)\}$$

where Ψ is a conjunction of bijective literals over variables $\overline{x} = (x_1, \ldots, x_k)$ and y. As for Theorem 2.1, two cases need to be distinguished.

(1) Ψ contains at least one literal of the form $\tau_1(y) = \tau_2(x_{i_0})$, $1 \leq i_0 \leq k$, that can also be rephrased as $y = \tau(x_{i_0})$ where $\tau(x_{i_0}) = \tau_1^{-1}\tau_2(x_{i_0})$;

(2) Ψ does not contain such a literal.

In the first case, Ψ can be rewritten as:

$$\Psi(\overline{x}, y) = \Psi_0(\overline{x}, y) \wedge y = \tau(x_{i_0}).$$

The query Q is then equivalent to:

$$Q = \{(\overline{a}, \tau(a_{i_0})) \in D^{k+1} : (\mathcal{S}, \overline{a}) \models \Psi_0(\overline{x}, \tau(x_{i_0}))\},$$

which is essentially the following k variable query Q':

$$Q' = \{\overline{a} \in D^k : (\mathcal{S}, \overline{a}) \models \Psi_0(\overline{x}, \tau(x_{i_0})\}.$$

To be precise, $Q = \{(\overline{a}, \tau(a_{i_0})) : \overline{a} \in Q'\}$. By the induction hypothesis, query Q' can be computed by some constant delay algorithm $\mathcal{A}'$. This provides a constant delay procedure for query Q obtained by replacing in algorithm $\mathcal{A}'$ each enumerated tuple $\overline{a}$ by the tuple $(\overline{a}, \tau(a_{i_0}))$.

Case 2 is a little more complicated. Formula Ψ can be put under the following form:

$$\Psi \equiv \Psi_1(\overline{x}) \wedge \Psi_2(y) \wedge \bigwedge_{1 \leq i \leq r} y \neq \tau_i(x_{j_i})$$

with $1 \leq j_i \leq k$ for $1 \leq i \leq r$. By the induction hypothesis, the k variable query:

$$Q_1 = \{\overline{a} \in D^k : (\mathcal{S}, \overline{a}) \models \Psi_1(\overline{x})\}$$

can be computed by an algorithm $\mathcal{A}_1$ on input $\mathcal{S}$ with constant delay. By the induction hypothesis, the k variable query Q_b over structure $(\mathcal{S}, b)$ defined for any $b \in D$ by:

$$Q_b = \{\overline{a} \in D^k : (\mathcal{S}, \overline{a}, b) \models \Psi(\overline{x}, y)\}\}$$

can be enumerated by an algorithm with constant delay. Let now Q_2 be the one variable query:

$$Q_2 = \{b \in D : (\mathcal{S}, b) \models \Psi_2(y)\},$$

[5] More precisely, for each fixed $a \in D, (\mathcal{S}, a) \models \Psi(\overline{x})$ can be checked in time $O(|\Psi|)$ by using addressing to the structure $\mathcal{S}$ (which is possible on a RAM)

that is computable in linear time. If $|Q_2| \leq r$ then, by Lemma 1.10, there exists an algorithm $\mathcal{A}_0$ which enumerates the disjoint union $\cup_{b \in Q_2} Q_b \times \{b\}$ with constant delay. Note that $\cup_{b \in Q_2} Q_b \times \{b\} = Q$. From what has been said Algorithm 2 below correctly computes query Q in that case.

Algorithm 2 Evaluating query Q

Input: $\mathcal{S}, \Psi$
Compute Q_2 and $|Q_2|$
if $|Q_2| \leq r$ **then** run $\mathcal{A}_0$
else
  PRECOMP($\mathcal{A}_1$) (*)
  for $\overline{a} \in$ ENUM($\mathcal{A}_1$) **do**
    for $b \in Q_2$ **do**
      if $(\mathcal{S}, \overline{a}, b) \not\models \bigvee_{1 \leq i \leq r} y = \tau_i(x_{j_i})$ **then** Output $(\overline{a}, b)$
      end if
    end for
  end for
end if

Up to step (*) of Algorithm 2, all can be done in linear time.

It remains to show that, in the case where $|Q_2| \geq r + 1$, the delay between two successive solutions is bounded by some constant. Since $|Q_2| \geq r + 1$ and the number of $b \in Q_2$ that verify $(\mathcal{S}, \overline{a}, b) \models \bigvee_{1 \leq i \leq r} y = \tau_i(x_{j_i})$ is trivially bounded by r, the algorithm outputs at least one $(\overline{a}, b)$ for each $\overline{a} \in Q_1$ (more precisely, it outputs at least $|Q_2| - r$ such tuples). As a consequence, the maximal delay between two successive outputs (resp. maximal delay after the last output) is bounded by $2r +$ DELAY($\mathcal{A}_1$) (resp. $r +$ DELAY($\mathcal{A}_1$)). Then, computing Q can be done with constant delay. □

PROOF OF THEOREM 2.4. Let $\mathcal{S}$ and $\varphi(\overline{x})$ be instances of the query problem QUERY-BIJ(**FO**). From Theorem 2.1, one can transform $\varphi(\overline{x})$ into the following equivalent formula in disjunctive normal form:

$$\varphi(\overline{x}) \equiv \Psi_1(\overline{x}) \vee \ldots \vee \Psi_q(\overline{x})$$

where each Ψ_i is a conjunction of bijective literals and for all i, j, $1 \leq i < j \leq q$, and all bijective structures $\mathcal{S}$, we have $\Psi_i(\mathcal{S}) \cap \Psi_j(\mathcal{S}) = \emptyset$. Theorem 2.4 immediately follows from Lemma 1.10 since the enumeration problem of each query $\mathcal{S} \mapsto \Psi_i(\mathcal{S})$, $1 \leq i \leq q$, belongs to CONSTANT-DELAY$_{lin}$ by Lemma 2.5. □

3. RELATIONAL STRUCTURES OF BOUNDED DEGREE

In this section, we apply the classical interpretability method due to Rabin [Rabin 1964]. This technique, designed originally for first-order theories, has been used and adapted by many authors including [Compton and Henson 1990; Arnborg et al. 1991; Seese 1992] and is also used in [Seese 1996]. Specifically, we present a very

simple interpretation of the first-order theory of d-degree-bounded structures into the first-order theory of bijective structures. This step is similar to the much more complex interpretation of the same theory into the first-order theory of k-degree bounded graphs presented in [Seese 1996] and is similarly computable in linear time.

3.1 Two equivalent definitions

Let $\rho = \{R_1, \ldots, R_q\}$ be a relational signature, i.e., a signature made of relational symbols R_i each of arity a_i. Recall that $arity(\rho) = max_{1 \leq i \leq q}(a_i) = m$.

Let $\mathcal{S} = \langle D; R_1, \ldots, R_q \rangle$ be a ρ-structure. For each $i \leq q$, $R_i \subseteq D^{a_i}$. Define the *degree* of an element x in $\mathcal{S}$, denoted $degree_{\mathcal{S}}(x)$ as the total number of tuples of relations R_i to which x belongs. One defines the degree of a structure as $degree(\mathcal{S}) = max_{x \in D}(degree_{\mathcal{S}}(x))$.

REMARK 3.1. *In [Seese 1996] a different definition of the degree of a structure is given. For each x, let $degree^1_{\mathcal{S}}(x)$ denote the number of elements $y \neq x$ adjacent to x, i.e., that appear in some tuple in which x also occurs and $degree^1(\mathcal{S}) = max_{x \in D}(degree^1_{\mathcal{S}}(x))$. This is the degree of the Gaifman graph of the structure.*

Since each tuple containing x contains at most $m-1$ elements different from x, it is easily seen that:

$$degree^1(\mathcal{S}) \leq (m-1).degree(\mathcal{S}) \text{ where } m = arity(\rho).$$

Conversely, if, for each x, there exist at most d elements $y \in D$ adjacent to x then, the number of distinct tuples involving x and y is bounded by $q.m.d^{m-1}$. Hence,

$$degree(\mathcal{S}) \leq q.m.(degree^1(\mathcal{S}))^{m-1}.$$

So, the two measures yield the same families of bounded degree relational structures.

We are interested in the complexity of the following query problem for d-bounded degree structures (which is independent of either measure of degree we choose).

QUERY-DEG(**FO**)
Input: an integer d, a relational signature ρ, a ρ-structure $\mathcal{S}$ with $degree(\mathcal{S}) \leq d$ and a first-order ρ-formula $\varphi(\overline{x})$ with free variables in the list $\overline{x} = (x_1, \ldots, x_k)$
Parameter: d, φ
Output: $\varphi(\mathcal{S}) = \{\overline{a} \in D^k : (\mathcal{S}, \overline{a}) \models \varphi(\overline{x})\}$.

3.2 Interpreting a structure of bounded degree into a bijective structure

In this section, we present a reduction from QUERY-DEG(**FO**) to QUERY-BIJ(**FO**) which is obtained by interpreting any d-degree-bounded structure into a bijective one.

Let $\mathcal{S} = \langle D; R_1, \ldots, R_q \rangle$ be a ρ-structure of domain D, of arity $m = max_{1 \leq i \leq q} arity(R_i)$ and of degree bounded by some constant d. One associates to $\mathcal{S}$ a bijective σ-structure $\mathcal{S}' = \langle D'; D, T_1, \ldots, T_q, g, f_1, \ldots, f_m \rangle$ of domain D' where $D, T_1, \ldots, T_q$ are pairwise disjoint unary relations (i.e. subsets of D') and $g, f_1, \ldots, f_m$ are permutations of D'. The structure $\mathcal{S}'$ is precisely defined as follows:

—D corresponds to the domain of $\mathcal{S}$.

—T_i $(1 \leq i \leq q)$ is a set of elements, each representing a tuple of R_i (hence, $card(T_i) = card(R_i)$).
The new domain D' is $D \cup (D \times \{1, \ldots, d\}) \cup T_1 \cup \ldots \cup T_q$. The sets forming this union are pairwise disjoint. In particular, the sets $T_1, \ldots, T_q$ are defined as disjoint sets, and they are disjoint from D, even if they correspond to unary relations. Let us use the following convenient abbreviations: $U = D \cup (D \times \{1, \ldots, d\})$ and $T = \bigcup_{1 \leq i \leq q} T_i$.

—g creates a cycle that relates d copies of each element x of the domain. More precisely, for each $x \in D$, it holds that $g(x) = (x, 1)$, $g((x, i)) = (x, i + 1)$ for $1 \leq i < d$, and $g((x, d)) = x$. We also set $g(x) = x$ for all other x $(x \in T)$.

—Each f_i is an involutive permutation and essentially represents a projection of T into D as follows. Let $R_i(x_1, \ldots, x_k)$ be true in $\mathcal{S}$ for some relation R_i of arity $k \leq m$ and some k-tuple $(x_1, \ldots, x_k) \in D^k$. Suppose $R_i(x_1, \ldots, x_k)$ is represented by an element $t \in T_i$, then, for each $j \leq k$, we set $f_j(t) = (x_j, h)$ and we set the inverse $f((x_j, h)) = t$ if $R(x_1, \ldots, x_k)$ is the h^{th} tuple in which x_j appears (with $h \leq d$). The construction is completed by setting $f_j(x) = x$ for all other $x \in D'$.

Figure 1 details the reduction on an example.

It is clear that, by construction, $\mathcal{S}'$ is a bijective structure and that we have the following interpretation Lemma.

LEMMA 3.2. *Let θ_i be the σ-formula:*

$$\theta_i(x_1, \ldots, x_k) \equiv \exists t(T_i(t) \wedge \bigwedge_{1 \leq j \leq k} \bigvee_{1 \leq h \leq d} f_j(t) = g^h(x_j)).$$

Then, for all $(a_1 \ldots, a_k) \in D^k$:

$$(\mathcal{S}, a_1, \ldots, a_k) \models R_i(x_1, \ldots, x_k) \iff (\mathcal{S}', a_1, \ldots, a_k) \models \theta_i(x_1, \ldots, x_k).$$

To each first-order ρ-formula $\varphi(x_1, \ldots, x_p)$, is associated the σ-formula $\varphi''(x_1, \ldots, x_p)$ obtained by replacing each quantification $\exists v$ (resp. $\forall v$) by the relativized quantification $(\exists v D(v))$ (resp. $(\forall v D(v))$) and by replacing each subformula $R_i(x_1, \ldots, x_k)$ by $\theta_i(x_1, \ldots, x_k)$.

The following proposition and lemma express that our reduction is correct and linear in $|\mathcal{S}|$. By using Lemma 3.2, one can easily prove Proposition 3.3 by induction on formula φ.

PROPOSITION 3.3 INTERPRETATION OF $\mathcal{S}$ INTO $\mathcal{S}'$. *Let $\varphi(x_1, \ldots, x_p)$ be any relational ρ-formula and $\varphi''(x_1, \ldots, x_p)$ be its translation into a bijective σ-formula. Similarly, let $\mathcal{S}$ and $\mathcal{S}'$ be any ρ-structure (of domain D) and its corresponding bijective σ-structure, respectively. For all $(a_1 \ldots, a_p) \in D^p$:*

$$(\mathcal{S}, a_1, \ldots, a_p) \models \varphi(x_1, \ldots, x_p) \iff (\mathcal{S}', a_1, \ldots, a_p) \models \varphi''(x_1, \ldots, x_p).$$

In other words: $\varphi(\mathcal{S}) = \varphi''(\mathcal{S}') \cap D^p$. Then, setting $\varphi'(x_1, \ldots, x_p) \equiv \varphi''(x_1, \ldots, x_p) \wedge \bigwedge_{i \leq p} D(x_i)$, it holds: $\varphi(\mathcal{S}) = \varphi'(\mathcal{S}')$

LEMMA 3.4. *Computing $\mathcal{S}'$ from $\mathcal{S}$ can be done in linear time $O_{\rho,d}(|\mathcal{S}|)$.*

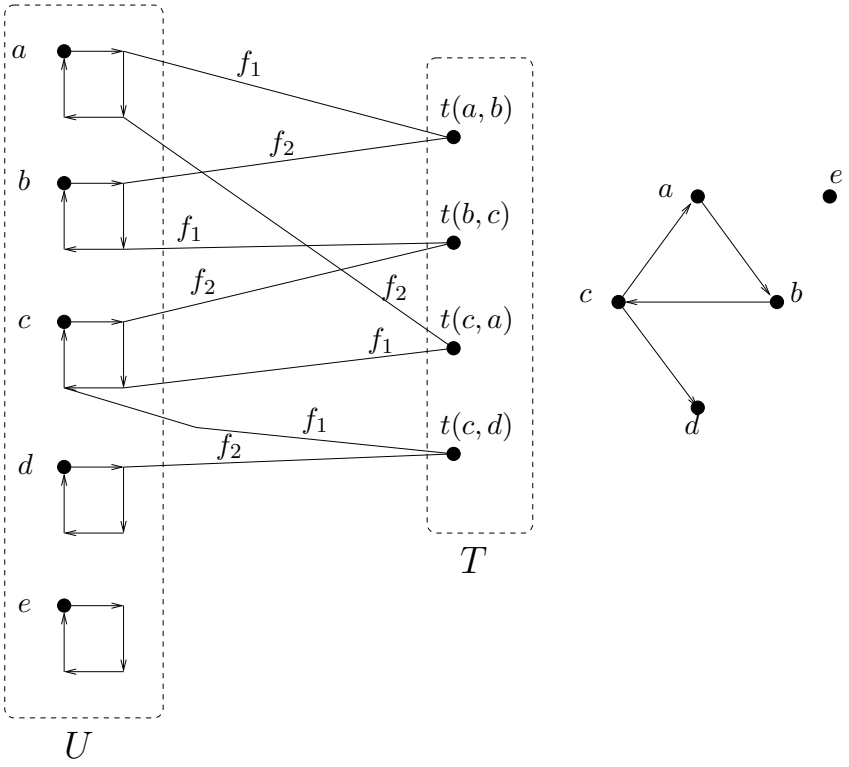

Fig. 1. Our reduction on an example: the original structure (digraph) of degree 3 is on the right side of the picture

PROOF. As computing $\mathcal{S}'$ from $\mathcal{S}$ is easy, one has only to compare the size of the two structures. In the following, $\Theta_k(n)$ means $\Theta(f(k).n)$ for some function f. The size of $\mathcal{S}$ is:

$$|\mathcal{S}| = \Theta(|D| + \sum_{i=1}^{q} card(R_i).arity(R_i)) = \Theta_\rho(|D| + \sum_{i=1}^{q} card(R_i)).$$

For $\mathcal{S}'$, by construction, it holds that:

$$|D'| = (d+1).|D| + \sum_{i=1}^{q} card(R_i) = \Theta_{d,\rho}(|\mathcal{S}|).$$

Hence, $|\mathcal{S}'| = \Theta(m|D'|) = \Theta_{d,\rho}(|\mathcal{S}|)$. □

We are now ready to state and prove the main result of this section.

THEOREM 3.5. *The problem* QUERY-DEG(**FO**) *belongs to* CONSTANT-DELAY$_{lin}$.

PROOF. Let $\mathcal{A}$ be a constant delay algorithm that computes the results of the problem QUERY-BIJ(**FO**). Proposition 3.3, yields that the algorithm given below evaluates correctly the results of a problem in QUERY-DEG(**FO**).

Algorithm 3 Evaluating QUERY-DEG(**FO**)

1: **Input:** $\mathcal{S}, d, \varphi$
2: Compute the σ-formula $\varphi'(\overline{x})$ associated with φ (and d)
3: Compute the bijective σ-structure $\mathcal{S}'$ associated with $\mathcal{S}$ (and d)
4: Run $\mathcal{A}$ on input $\mathcal{S}'$, φ'

The cost of instruction 2 is constant in $|\varphi|$ and d, that of instruction 3 is $O_{\varphi,d}(|\mathcal{S}|)$ (by Lemma 3.4) and the cost of the precomputation part of algorithm $\mathcal{A}$ (included in instruction 4) is $O_{\varphi'}(|\mathcal{S}'|)$ (hence $O_{\varphi,d}(|\mathcal{S}|)$) by Theorem 2.4. These steps form a precomputation phase of time complexity $O_{\varphi,d}(|\mathcal{S}|)$. Finally, the effective enumeration of $\varphi(\mathcal{S}) = \varphi'(\mathcal{S}')$ is handled on $\mathcal{S}', \varphi'$ by $\mathcal{A}$ and is performed with constant delay. □

3.3 Complexity of subgraphs problems

In this part, we present a simple application of our result to a well-known graph problem. 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 $u, v \in V_H$, $E(g(u), g(v))$ holds if (resp. if and only if) $E_H(u, v)$ holds.

GENERATE SUBGRAPH (resp. GENERATE INDUCED SUBGRAPH)
Input: any graph H and a graph G of degree bounded by d
Parameter: $|H|, d$.
Output: All the subgraphs (resp. induced subgraphs) of G isomorphic to H.

The treewidth of a graph G is the maximal size minus one of a node in a tree decomposition of G (see, [Robertson and Seymour 1983; 1984] or [Downey and Fellows 1999]). In [Plehn and Voigt 1990] it is proved that for graphs H of treewidth at most w, testing if a given graph H is an induced subgraph of a graph G of degree at most d can be done in time $f(|H|, d).|G|^{w+1}$. In what follows, we show that there is no reason to focus on graphs of bounded treewidth and that a better bound can be obtained for *any* graph H (provided G is of bounded degree). In the result below, we prove that not only the complexity of this decision problem is $f(|H|, d).|G|$ but that generating all the (induced) subgraphs isomorphic to H can be done with constant delay.

COROLLARY 3.6. *The problem* GENERATE SUBGRAPH *(resp.* GENERATE INDUCED SUBGRAPH*) belongs to* CONSTANT-DELAY$_{lin}$.

PROOF. The proof is given for the problem GENERATE INDUCED SUBGRAPH. Let $G = \langle V; E\rangle$ and $H = \langle V_H = \{h_1, \ldots, h_k\}; E_H\rangle$ ($|V_H| = k$) be the two inputs of the problem. 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 φ be the following formula:

$$\varphi(x_1, \ldots, x_k) \equiv \bigwedge_{i<j\leq k} x_i \neq x_j \wedge \bigwedge_{\neg E_H(h_i,h_j)} \neg E(x_i, x_j) \wedge \bigwedge_{E_H(h_i,h_j)} E(x_i, x_j).$$

Note that the formula φ depends only on H. The result follows now from Theorem 3.5. □

4. CONCLUSION

In this paper, we study the complexity of evaluating first-order queries on bounded degree structures and consider this evaluation as a dynamical process, i.e., as an enumeration problem. Our main contributions are two-fold. First, we define a simple quantifier elimination method suitable for first-order formulas which have to be evaluated in bijective structures. Second, we define a new complexity class, called CONSTANT-DELAY$_{lin}$. It can be seen as the minimal robust complexity class for enumeration problems and we prove that our query problems on bounded degree structures belong to this class.

There are several interesting directions for further research. Among them, the following series of questions seem worth to be studied:

—Which "natural" query problems belong to CONSTANT-DELAY$_{lin}$? More generally, which kind of combinatorial or algorithmic enumeration problems admit constant delay procedures ? is it the case, for example, of first-order queries on trees of unbounded degree or on relations of locally bounded tree-width (see [Frick and Grohe 2001]) ?
The same questions can be asked for the strictly larger class CONSTANT-DELAY$_{poly}$ of constant delay enumeration problems for which polynomial time (instead of linear time) precomputations are allowed.

—What are the structural properties of the complexity classes CONSTANT-DELAY$_{lin}$ and CONSTANT-DELAY$_{poly}$? Do they have complete problems ? Under which kind of reductions ? Could they be proved to be different from the classes of enumeration problems solvable with linear or polynomial delay, respectively ?

Acknowledgments. We thank Ron Fagin for a very fruitful email exchange that lead us to define complexity notions about constant delay computations. Thanks to Bruno Courcelle for many helpful discussions and suggestions. Finally, we are pleased to thank the referees for their very careful reading of the manuscript and for their useful remarks that help us to notably improve the readability of the paper.

REFERENCES

AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

ARNBORG, S., LAGERGREN, J., AND SEESE, D. 1991. Easy problems for tree-decomposable graphs. *Journal of Algorithms 12,* 2, 308–340.

BOROS, E., GURVICH, V., KHACHIYAN, L., AND MAKINO, K. 2000. Generating partial and multiple transversals of a hypergraph. In *Proceedings 27th International Conference on Automata, Languages, and Programming (ICALP 2000), Geneva (Switzerland)*, U. Montanari, J. D. P. Rolim, and E. Welzl, Eds. Lecture Notes in Computer Science, vol. 1853. Springer-Verlag, 588–599.

COMPTON, K. AND HENSON, C. 1990. A uniform method for proving lower bounds on the computational complexity of logical theories. *Annals of Pure and Applied Logic 48*, pp.1–79.

DOWNEY, R. G. AND FELLOWS, M. R. 1999. *Parameterized complexity*. Springer-Verlag.

Eiter, T. and Gottlob, G. 1995. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing 24,* 6, 1278–1304.

Eiter, T., Gottlob, G., and Makino, K. 2003. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing 32,* 2, 514–537.

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

Frick, M. and Grohe, M. 2001. Deciding first-order properties of locally tree decomposable structures. *Journal of the ACM 48*, 1184–1206.

Gaifman, H. 1982. On local and nonlocal properties. In *Logic Colloquium'81*, J. Stern, Ed. North-Holland, 105–135.

Goldberg, L. A. 1994. Listing graphs that satisfy first order sentences. *Journal of Computer and System Sciences 49,* 2, 408–424.

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

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

Hanf, W. 1965. Model-theoretic methods in the study of elementary logic. In *The Theory of Models*, L. H. J. Addison and A. Tarski, Eds. North-Holland, 132–145.

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

Kavvadias, D. J., Sideri, M., and Stavropoulos, E. C. 2000. Generating all maximal models of a boolean expression. *Information Processing Letters 74,* 3-4, 157–162.

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

Lindell, S. 2005. Computing monadic fixed-point in linear-time on doubly-linked data structures. In *7th Workshop on Logic and Computational Complexity (LCC'05).*

Plehn, J. and Voigt, B. 1990. Finding minimally weighted subgraphs. In *16th workshop on graph theoretic concepts in computer science*, Springer, Ed. Lecture Notes in Computer Science, vol. 484. 18–29.

Rabin, M. 1964. A simple method of undecidability proofs and some applications. In *Logic, Methodology and Phylosophy of Science*, Bar-Hillel, Ed. Vol. 1. North-Holland, Amsterdam, 58–68.

Robertson, N. and Seymour, P. 1983. Graph minors I. excluding a forest. *Journal of Combinatorial Theory Ser. B 35*, 39–61.

Robertson, N. and Seymour, P. 1984. Graph minors III. planar tree-width. *Journal of Combinatorial Theory Ser. B 41*, 92–114.

Seese, D. 1992. Interpretability and tree automata: a simple way to solve algorithmic problems on graphs closely related to trees. In *Tree Automata and Languages*, M. Nivat and A. Podelski, Eds. Elsevier Science, 83–114.

Seese, D. 1996. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science 6,* 6 (December), 505–526.

Vardi, M. Y. 1995. On the complexity of bounded-variable queries. In *ACM Symposium on Principles of Database Systems.* ACM Press, 266–276.

Received July 2005; revised September 2005; accepted November 2005.