Enumeration of First-Order Queries on Classes of Structures With Bounded Expansion

Wojciech Kazana*
INRIA and ENS Cachan
kazana@lsv.ens-cachan.fr

Luc Segoufin
INRIA and ENS Cachan
http://pages.saclay.inria.fr/luc.segoufin/

ABSTRACT

We consider the evaluation of first-order queries over classes of databases with *bounded expansion*. The notion of bounded expansion is fairly broad and generalizes bounded degree, bounded treewidth and exclusion of at least one minor. It was known that over a class of databases with bounded expansion, first-order sentences could be evaluated in time linear in the size of the database. We first give a different proof of this result. Moreover, we show that answers to first-order queries can be enumerated with constant delay after a linear time preprocessing. We also show that counting the number of answers to a query can be done in time linear in the size of the database.

1. INTRODUCTION

Query evaluation is certainly the most important problem in databases. Given a query q and a database $\mathbf{D}$ it is to compute the set $q(\mathbf{D})$ of all tuples in the output of q on $\mathbf{D}$. However, the set $q(\mathbf{D})$ may be larger than the database itself as it can have a size of the form n^l where n is the size of the database and l the arity of the query. It can therefore require too many of the available resources to compute it entirely.

There are many solutions to overcome this problem. For instance one could imagine that a small subset of $q(\mathbf{D})$ can be quickly computed and that this subset will be enough for the user needs. Typically one could imagine computing the top-ℓ most relevant answers relative to some ranking function or to provide a sampling of $q(\mathbf{D})$ relative to some distribution. One could also imagine computing only the number of solutions $|q(\mathbf{D})|$ or providing an efficient test for whether a given tuple belongs to $q(\mathbf{D})$ or not.

In this paper we consider a scenario consisting in enumerating $q(\mathbf{D})$ with constant delay. Intuitively, this means that there is a two-phase algorithm working as follows: a preprocessing phase that works in time linear in the size of the database, followed by an enumeration phase outputting one by one all the elements of $q(\mathbf{D})$ with a constant delay between any two consecutive outputs. In particular, the first answer is output after a time linear in the size of the database and once the enumeration starts a new answer is being output regularly at a speed independent from the size of the database. Altogether, the set $q(\mathbf{D})$ is entirely computed in time $f(q)(n + |q(\mathbf{D})|)$ for some function f depending only on q and not on $\mathbf{D}$.

One could also view a constant delay enumeration algorithm as follows. The preprocessing phase computes in linear time an index structure representing the set $q(\mathbf{D})$ in a compact way (of size linear in n). The enumeration algorithm is then a streaming decompression algorithm.

One could also require that the enumeration phase outputs the answers in some given order. Here we will consider the lexicographical order based on a linear order on the domain of the database.

There are many problems related to enumeration. The main one is the model checking problem. This is the case when the query is boolean, i.e. outputs only 0 or 1. In this case a constant delay enumeration algorithm is a Fixed Parameter Linear (FPL) algorithm for the model checking problem of q, i.e. it works in time $f(q)n$. This is a rather strong constraint as even the model checking problem for conjunctive queries is not FPL (modulo some hypothesis in parametrized complexity) [19]. Hence, in order to obtain constant delay enumeration algorithms, we need to make restrictions on the queries and/or on the databases. Here we consider first-order (FO) queries over classes of structures having "bounded expansion".

The notion of class of graphs with bounded expansion was introduced by Nešetřil and Ossona de Mendez in [16]. Its precise definition can be found in Section 2.2. At this point it suffices to know that it contains the class of graphs of bounded degree, the class of graphs of bounded treewidth, the class of planar graphs, and any class of graphs excluding at least one minor. This notion is generalized to classes of structures via their Gaifman graphs or adjacency graphs.

For the class of structures with bounded degree and FO queries the model checking is in FPL [20] and there also are constant delay enumeration algorithms [9, 13]. In the case of structures of bounded treewidth and FO queries (actually even MSO queries with first-order free variables) the model checking is also in FPL [8] and there are constant de-

*This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant Webdam, agreement 226513. http://webdam.inria.fr/

lay enumeration algorithms [4, 14]. For classes of structures with bounded expansion the model checking problem for FO queries was recently shown to be in FPL [10, 12].

Our results can be summarized as follows. For FO queries and any class of structures with bounded expansion:

- we provide a new proof that the model checking problem can be solved in FPL,
- we show that the set of solutions to a query can be enumerated with constant delay,
- we show that computing the number of solutions can be done in FPL,
- we show that, after a preprocessing in time linear in the size of the database, one can test on input $\bar{a}$ whether $\bar{a} \in q(\mathbf{D})$ in constant time.

Concerning model checking, our method uses a different technique than the previous ones. There are several characterizations of classes having bounded expansion [16]. Among them we find the "low tree depth coloring" and the "transitive fraternal augmentations". The previous methods were based on the low tree depth coloring characterization while ours is based on transitive fraternal augmentations. We argue that the use of transitive fraternal augmentations gives a simpler proof. The reason is that it gives a useful normal form on quantifier-free formulas that will be the core of our algorithms for constant delay enumeration and for counting the number of solutions. As for the previous proofs, we exhibit a quantifier elimination method, also based on our normal form. Our quantifier elimination method results in a quantifier-free formula but over a recoloring of a functional representation of a "fraternal and transitive augmentation" of the initial structure.

Our other algorithms (constant delay enumeration, counting the number of solution or testing whether a tuple is a solution or not) start by eliminating the quantifiers as for the model checking algorithm. Note that for all these problems, the quantifier-free case is already non trivial and require the design and the computation of new index structures. For instance consider the simple query $R(x, y)$. Given a pair (a, b) we would like to test whether (a, b) is a tuple of the database in constant time. In general, index structures can do this with $\log n$ time. We will see that we can do constant time, assuming bounded expansion.

In the presence of a linear order on the domain of the database, our constant delay algorithm can output the answers in the corresponding lexicographical order.

Related work.

We make use of a functional representation of the initial structures. Without this functional representations we would not be able to eliminate first-order quantifiers. Indeed, with this functional representation we can talk of a node at distance 2 from x using the quantifier-free term $f(f(x))$, avoiding the existential quantification of the middle point. This idea was already taken in [9] for eliminating first-order quantifiers over structures of bounded degree. Our approach differs from theirs in the fact that in the bounded degree case the functions can be assumed to be permutations (in particular they are invertible) while this is no longer true in our setting, complicating significantly the combinatorics.

Once we have a quantifier-free formula, constant delay enumeration could also be obtained using the characterization of bounded expansion based on low tree depth colorings. Indeed, using this characterization one can easily show that enumerating a quantifier-free formula over structures of bounded expansion amounts in enumerating an MSO query over structures of bounded tree-width and for those known algorithms exist [4, 14]. However, the known enumeration algorithms of MSO over structures of bounded treewidth are rather complicated while our direct approach is fairly simple. Actually, our proof shows that constant delay enumeration of FO queries over structures of bounded treewidth can be done using simpler algorithms than for MSO queries. Moreover, it gives a constant delay algorithm outputting the solutions in lexicographical order. No such algorithms were known for FO queries over structures of bounded treewidth. In the bounded degree case, both enumeration algorithms of [9, 13] output their solutions in lexicographical order.

Similarly, counting the number of solutions of a quantifier-free formula over structures of bounded expansion reduces to counting the number of solutions of a MSO formula over structures of bounded treewidth. This latter problem is known to be in FPL [3]. We give here a direct and simple proof of this fact for FO queries over structures of bounded expansion.

2. PRELIMINARIES

In this paper a database is a finite relational structure. A *relational signature* is a tuple $\sigma = (R_1, \ldots, R_l)$, each R_i being a relation symbol of arity r_i. A *relational structure* over σ is a tuple $\mathbf{D} = \left(D, R_1^{\mathbf{D}}, \ldots, R_l^{\mathbf{D}}\right)$, where D is *the domain* of $\mathbf{D}$ and $R_i^{\mathbf{D}}$ is a subset of D^{r_i}. We fix a reasonable encoding of structures by words over some finite alphabet, as in [1] for instance. The *size* of $\mathbf{D}$ is denoted by $\|\mathbf{D}\|$ and is the length of the encoding of $\mathbf{D}$.

By *query* we mean a formula written in the first-order logic, FO, built from atomic formulas of the form $x = y$ or $R_i(x_1, \ldots, x_{r_i})$ for some relation R_i, and closed under the usual Boolean connectives $(\neg, \vee, \wedge)$ and existential and universal quantifications $(\exists, \forall)$. We write $\phi(\bar{x})$ to denote a query whose free variables are $\bar{x}$, and the number of free variables is called the *arity of the query*. A *sentence* is a query of arity 0. Given a structure $\mathbf{D}$ and a query ϕ, an *answer* to q in $\mathbf{D}$ is a tuple $\bar{a}$ of elements of $\mathbf{D}$ such that $\mathbf{D} \models \phi(\bar{a})$. We write $\phi(\mathbf{D})$ for the set of answers to q in $\mathbf{D}$, i.e. $\phi(\mathbf{D}) = \{\bar{a} \mid \mathbf{D} \models \phi(\bar{a})\}$. By $|\phi(\mathbf{D})|$ we denote the cardinality of the set $\phi(\mathbf{D})$. As usual, $|\phi|$ denotes the size of ϕ.

Let $\mathcal{C}$ be a class of structures. The model checking problem of FO over $\mathcal{C}$ is the computational problem of given a

sentence $q \in$ FO and a database $\mathbf{D} \in \mathcal{C}$ to test whether $\mathbf{D} \models q$ or not.

We now introduce our running examples.

EXAMPLE A-1. *The first query has arity 2 and returns pairs of nodes at distance at most two in a graph. We use the classical notion of distance that ignores the possible orientation of the edges. The query is of the form* $\exists z E(x,z) \wedge E(z,y)$*, where* E *is the symmetric closure of the input relation.*

Testing the existence of a solution to this query can be easily done in time linear in the size of the database. For instance one can go trough all nodes of the database and check whether it has degree two. The degree of each node can be computed in linear time by going through all edges of the database and incrementing the degree counters associated with its endpoints.

EXAMPLE B-1. *The second query has arity 3 and returns triples* (x,y,z) *such that* y *is connected to* x *and* z *via an edge but* x *is not connected to* z*. The query is of the form* $E(x,y) \wedge E(y,z) \wedge \neg E(x,z)$*, where* E *is the symmetric closure of the input relation.*

It is not clear at all how to test the existence of a solution to this query in time linear in the size of the database. The problem is similar to the one of finding a triangle in a graph, for which the best known algorithm has complexity even slightly worse than matrix multiplication [2]. If the degree of the input structure is bounded by a constant d*, we can test the existence of a solution in linear time by the following algorithm. We first go through all edges* (x,y) *of the database and add* y *to a list associated with* x *and* x *to a list associated with* y*. It remains now to go through all nodes* y *of the database, consider all pairs* (x,z) *of nodes in the associated list (the number of such pairs is bounded by* d^2*) and then test whether there is an edge between* x *and* z *(by testing whether* x *is in the list associated with* z*).*

We aim at generalizing this kind of reasoning to structures with bounded expansion.

Given a query q, we care about "enumerating" $q(\mathbf{D})$ efficiently. Let $\mathcal{C}$ be a class of structures. For a query $q(\bar{x})$, *the enumeration problem of* q *over* $\mathcal{C}$ is, given a database $\mathbf{D} \in \mathcal{C}$, to output the elements of $q(\mathbf{D})$ one by one with no repetition. The maximal time between any two consecutive outputs of elements of $q(\mathbf{D})$ is called *the delay*. The definition below requires a constant time between any two consecutive outputs. We formalize these notions in the forthcoming sections.

2.1 Model of computation and enumeration

We use Random Access Machines (RAM) with addition and uniform cost measure as a model of computation. For further details on this model and its use in logic see [9]. In the sequel we assume without loss of generality that the input relational structure comes with a linear order on its domain (if not, we use the one induced by the encoding of the database as a word). Whenever we iterate through all nodes of the domain, the iteration is with respect to the initial linear order.

We say that the enumeration problem of q over a class $\mathcal{C}$ of structures is in the class CONSTANT-DELAY$_{lin}$, or equivalently that we can enumerate q over $\mathcal{C}$ with constant delay[1], if it can be solved by a RAM algorithm which, on input $\mathbf{D} \in \mathcal{C}$, can be decomposed into two phases:

- a precomputation phase that is performed in time $O(\|\mathbf{D}\|)$,
- an enumeration phase that outputs $q(\mathbf{D})$ with no repetition and a constant delay between two consecutive outputs. The enumeration phase has full access to the output of the precomputation phase but can use only a constant total amount of extra memory.

Notice that if we can enumerate q with constant delay, then all answers can be output in time $O(\|\mathbf{D}\| + |q(\mathbf{D})|)$ and the first output is computed in time linear in $\|\mathbf{D}\|$. In the particular case of boolean queries, the associated model checking problem must be solvable in time linear in $\|\mathbf{D}\|$.

We may in addition require that the enumeration phase outputs the answers to q using the lexicographical order. We then say that we can enumerate q over $\mathcal{C}$ with constant delay in lexicographical order.

EXAMPLE A-2. *Over the class of all graphs, we cannot enumerate pairs of nodes at distance 2 with constant delay unless the Boolean Matrix Multiplication problem can be solved in quadratic time [6]. However, over the class of graphs of degree* d*, there is a simple constant delay enumeration algorithm. During the preprocessing phase, we associate with each node the list of all its neighbors at distance 2. This can be done in time linear in the database as in Example B-1. We then color in blue all nodes having a non empty list and make sure each blue node points to the next blue node (according to the linear order on the domain). This also can be done in time linear in the database and concludes the preprocessing phase. The enumeration phase now goes through all blue nodes* x *using the pointer structure and, for each of them, outputs all pairs* (x,y) *where* y *is in the list associated with* x*.*

EXAMPLE B-2. *Over the class of all graphs, the query of this example cannot be enumerated in constant delay because, as mentioned in Example B-1, testing whether there is one solution is already non linear. Over the class of graphs of bounded degree, there is a simple constant delay enumeration algorithm, similar to the one from Example A-2.*

Note that in general constant delay enumeration algorithms are not closed under any boolean operations. For instance

[1]For readability we use the term "enumerate with constant delay", but technically speaking it should read "enumerate with constant delay after linear preprocessing". The reader should keep in mind that the linear preprocessing, although not explicitly mentioned, always precedes the enumeration process.

it is not because we can enumerate q and q' with constant delay, that we can enumerate $q \vee q'$ with constant delay as enumerating one query after the other would break the "no repetition" requirement. However, if we can enumerate with constant delay in the lexicographical order, then a simple argument that resembles the problem of merging two sorted lists shows closure under union:

LEMMA 1. *If both queries $q(\bar{x})$ and $q'(\bar{x})$ can be enumerated in lexicographical order with constant delay then the same is true for $q(\bar{x}) \vee q'(\bar{x})$.*

It will follow from our results that the enumeration problem of FO over the class of structures with "bounded expansion" is in CONSTANT-DELAY$_{lin}$. The notion of bounded expansion was defined in [16] for graphs and then it was generalized to structures via their Gaifman or Adjacency graphs. We start with defining it for graphs.

2.2 Graphs with bounded expansion and augmentation

In this paper a graph is a directed graph with colors on vertices. We can then view a graph as a relational structure $\mathbf{G} = (V, E, P_1, \ldots, P_l)$, where V is the set of nodes, $E \subseteq V^2$ is the set of oriented edges and, for each $1 \leq i \leq l$, P_i is a predicate of arity 1. A pair $(u, v) \in E$ represents an edge from node u to node v. The *in-degree* of a node v is the number of nodes u such that $(u, v) \in E$. By $\Delta^-(\mathbf{G})$ we mean the maximal in-degree of a node of $\mathbf{G}$.

In [16] several equivalent definitions of bounded expansion were shown. We will not use here the initial definition but the one exploiting the notion of "augmentations". The interested reader can find in Appendix 8.1 the initial definition of bounded expansion.

Let $\mathbf{G}$ be a graph. A 1-*transitive fraternal augmentation of* $\mathbf{G}$ is any graph $\mathbf{H}$ with the same vertex set as $\mathbf{G}$ and the same colors of vertices, including all edges of $\mathbf{G}$ (with their orientation) and such that for any three vertices x, y, z of $\mathbf{G}$ we have the following:

(transitivity) if (x, y) and (y, z) are edges in $\mathbf{G}$, then (x, z) is an edge in $\mathbf{H}$,

(fraternity) if (x, z) and (y, z) are edges in $\mathbf{G}$, then at least one of the edges: (x, y), (y, x) is in $\mathbf{H}$,

(strictness) moreover, if $\mathbf{H}$ contains an edge that was not present in $\mathbf{G}$, then it must have been added by one of the previous two rules.

Note that the notion of 1-transitive fraternal augmentation is not a deterministic operation. Although transitivity induces precise edges, fraternity implies nondeterminism and thus there can possibly be many different 1-transitive fraternal augmentations. We care here about choosing the orientations of the edges resulting from the fraternity rule in order to minimize the maximal in-degree.

Following [17] we fix a deterministic algorithm computing a "good" choice of orientations of the edges induced by the fraternity property. The precise definition of the algorithm is not important for us, it only matters here that the algorithm runs in time linear in the size of the input graph (see Lemma 2 below). With this algorithm fixed, we can now speak of **the** 1-transitive fraternal augmentation of $\mathbf{G}$.

Let $\mathbf{G}$ be a graph. The *transitive fraternal augmentation* of $\mathbf{G}$ is the sequence $\mathbf{G} = \mathbf{G}_0 \subseteq \mathbf{G}_1 \subseteq \mathbf{G}_2 \subseteq \ldots$ such that for each $i \geq 1$ the graph $\mathbf{G}_{i+1}$ is the 1-transitive fraternal augmentation of $\mathbf{G}_i$. We will say that $\mathbf{G}_i$ is the i-th augmentation of $\mathbf{G}$.

DEFINITION 1. *[16] Let $\mathcal{C}$ be a class of graphs. $\mathcal{C}$ has bounded expansion if there exists a function $\Gamma_{\mathcal{C}} : \mathbb{N} \to \mathbb{R}$ such that for each graph $\boldsymbol{G} \in \mathcal{C}$ the transitive fraternal augmentation $\boldsymbol{G} = \boldsymbol{G}_0 \subseteq \boldsymbol{G}_1 \subseteq \boldsymbol{G}_2 \subseteq \ldots$ of $\boldsymbol{G}$ is such that for each $i \geq 0$ we have $\Delta^-(\boldsymbol{G}_i) \leq \Gamma_{\mathcal{C}}(i)$.*

Consider for instance a graph of degree d. Notice that the 1-transitive fraternal augmentation introduces an edge between nodes that were at distance at most 2 in the initial graph. Hence, when starting with a graph of degree d, we end up with a graph of degree at most d^2. This observation shows that the class of graphs of degree d has bounded expansion as witnessed by the function $\Gamma(i) = d^{2^i}$. Exhibiting the function Γ for the other examples of classes with bounded expansion mentioned in the introduction: bounded treewidth, planar graphs, graphs excluding at least one minor, requires more work [16].

The following lemma shows that within a class $\mathcal{C}$ of bounded expansion the i-th augmentation of $\mathbf{G} \in \mathcal{C}$ can be computed in linear time.

LEMMA 2. *[17] Let $\mathcal{C}$ be a class of bounded expansion. For each $\boldsymbol{G} \in \mathcal{C}$ and each i, $\boldsymbol{G}_i$ is computable from $\boldsymbol{G}_{i-1}$ in time $O(\|\boldsymbol{G}_{i-1}\|)$.*

In particular Lemma 2 implies that for each i, given $\mathbf{G} \in \mathcal{C}$, $\mathbf{G}_i$ is computable from $\mathbf{G}$ in time $O(\|\mathbf{G}\|)$.

2.3 Graphs of bounded in-degree as functional structures

For the rest of this section we fix a class $\mathcal{C}$ of graphs with bounded expansion and let $\Gamma_{\mathcal{C}}$ be the function given by Definition 1. For any graph $\mathbf{G} \in \mathcal{C}$ its transitive fraternal augmentation $\mathbf{G} = \mathbf{G}_0 \subseteq \mathbf{G}_1 \subseteq \mathbf{G}_2 \subseteq \ldots$ is such that for all i, $\mathbf{G}_i$ has in-degree bounded by $\Gamma_{\mathcal{C}}(i)$. From the definition of bounded expansion it follows that the maximal in-degree of the graphs we will manipulate is always bounded by a number independent of the graph. We will use this property by constantly referring to the $1^{st}, 2^{nd} \ldots$ predecessor of a node. It will therefore be convenient for us to represent the graphs $\mathbf{G}_i$ as functional structures where this predecessors are images of the current node via some suitable functions. As mentioned in the introduction, this functional representation is also useful for eliminating some quantifiers.

A *functional signature* is a tuple $\sigma = (f_1, \ldots, f_l, P_1, \ldots, P_m)$, each f_i being a functional symbol of arity 1 and each P_i being an unary predicate. A *functional structure* over σ is then

defined as for relational structures. FO is defined as usual over the functional signature. In particular, it can use atoms of the form $f(f(f(x)))$, which is crucial for the quantifier elimination step of Section 3 as the usual relational representation would require existential quantification for denoting the same element. A graph $\mathbf{G}$ of in-degree l and colored with m colors can be represented as a functional structure $\vec{\mathbf{G}}$, where the unary predicates encode the various colors and $v = f_i(u)$ if v is the i^{th} element (according to some arbitrary order that will not be relevant in the sequel) such that (v, u) is an edge of $\mathbf{G}$. We call such node v the *i^{th} predecessor* of u (where "i^{th} predecessor" should really be viewed as an abbreviation for "the node v such that $f_i(u) = v$" and not as a reference to the chosen order). If we do not care about i and we only want to say that v is the image of u under some function, we call it a *predecessor* of u. Given $\mathbf{G} \in \mathcal{C}$ we define $\vec{\mathbf{G}}$ to be the functional representation of $\mathbf{G}$ as described above. Note that $\vec{\mathbf{G}}$ is computable in time linear in $\|\mathbf{G}\|$ and that for each first order query $\phi(\bar{x})$ one can easily compute a first order query $\psi(\bar{x})$ such that $\phi(\mathbf{G}) = \psi(\vec{\mathbf{G}})$.

EXAMPLE A-3. *With the functional point of view, the query computing nodes at distance at most two is of the form:*

$$\bigvee_{f,g\in\sigma} f(g(x)) = y \ \vee \ g(f(y)) = x \ \vee \ f(x) = g(y) \vee \exists z \ \ f(z) = x \wedge g(z) = y$$

where there is one disjunct per possible orientation of the edges on the path from x to y. We have removed the inner node z whenever this was possible.

EXAMPLE B-3. *Similarly, the query of Example B-1 is equivalent to:*

$$\begin{aligned}\bigvee_{f,g\in\sigma} \quad & \bigwedge_{h\in\sigma} (h(x) \neq z \wedge h(z) \neq x) \\ & \wedge \ [(f(x) = y \wedge g(y) = z) \\ & \quad \vee (x = f(y) \wedge g(y) = z) \\ & \quad \vee (f(x) = y \wedge y = g(z)) \\ & \quad \vee (x = f(y) \wedge y = g(z))].\end{aligned}$$

Recall that the augmentation steps only introduce new edges and do not affect the vertex set. It will be convenient for us to be able to recover $\mathbf{G}_i$ from $\mathbf{G}_{i+1}$. For this we use extra function symbols denoting the edges resulting from an augmentation step. The definition of bounded expansion guarantees that the number of required new symbols is bounded by $\Gamma_{\mathcal{C}}(i+1)$ and does not depend on the graph.

From this it follows that we have functional signatures $\sigma_{\mathcal{C}}(0) \subseteq \sigma_{\mathcal{C}}(1) \subseteq \sigma_{\mathcal{C}}(2) \subseteq \ldots$, where $\sigma_{\mathcal{C}}(0)$ is the initial signature and $\sigma_{\mathcal{C}}(i+1)$ is $\sigma_{\mathcal{C}}(i)$ plus the $\Gamma_{\mathcal{C}}(i+1)$ extra symbols needed for the extra augmentation step, such that for any graph $\mathbf{G} \in \mathcal{C}$ and for all i:

1. $\vec{\mathbf{G}}_i$ is a functional structure over $\sigma_{\mathcal{C}}(i)$,
2. $\vec{\mathbf{G}}_i \subseteq \vec{\mathbf{G}}_{i+1}$ and $\vec{\mathbf{G}}_{i+1}$ is computable in linear time from $\vec{\mathbf{G}}_i$,
3. for every FO query $\phi(\bar{x})$ over $\sigma_{\mathcal{C}}(i)$ and every $j \geq i$ we have that $\phi(\vec{\mathbf{G}}_i) = \phi(\vec{\mathbf{G}}_j)$.

We denote by $\alpha_{\mathcal{C}}(i)$ the number of function symbols of $\sigma_{\mathcal{C}}(i)$. It follows from the discussion above that $\alpha_{\mathcal{C}}(i) = \Sigma_{j\leq i}\Gamma_{\mathcal{C}}(j)$. It would be tempting to reduce this number by reusing function symbols, but that would then be problematic to enforce 3. (See Appendix 8.2.)

We say that a functional signature σ' is a *recoloring* of σ if it extends σ with some extra unary predicates (colors), while the functional part remains unchanged. Similarly, a functional structure $\vec{\mathbf{G}}'$ over σ' is a *recoloring* of $\vec{\mathbf{G}}$ over σ if σ' is a recoloring of σ and $\vec{\mathbf{G}}'$ is a σ'-expansion of $\vec{\mathbf{G}}$ (i.e. it does not differ from $\vec{\mathbf{G}}$ on the predicates in σ). We write *ϕ is over a recoloring of σ* if ϕ is over σ' and σ' is a recoloring of σ.

For each $p \geq 0$ we define $\mathcal{C}_p$ to be the class of all recolorings $\vec{\mathbf{G}}'_p$ of $\vec{\mathbf{G}}_p$ for some $\mathbf{G} \in \mathcal{C}$. In other words $\mathcal{C}_p$ is the class of functional representations of all recolorings of all p-th augmentations of graphs from $\mathcal{C}$. Note that all graphs from $\mathcal{C}_p$ are recolorings of a structure in $\sigma_{\mathcal{C}}(p)$, hence they use at most $\alpha_{\mathcal{C}}(p)$ function symbols.

From now on we assume that all graphs from $\mathcal{C}$ and all queries are in their functional representation. It follows from the discussion above that this is without loss of generality.

2.4 From structures to graphs

The *adjacency graph* of a relational structure $\mathbf{D}$, denoted by Adjacency($\mathbf{D}$), is a functional graph defined as follows. The set of vertices of Adjacency($\mathbf{D}$) is $D \cup T$ where T is the set of tuples occurring in some relation of $\mathbf{D}$. For each relation R_i in the schema of $\mathbf{D}$, there is a unary symbol P_{R_i} coloring the elements of T belonging to R_i. For each tuple $t = (a_1, \cdots, a_{r_i})$ such that $\mathbf{D} \models R_i(t)$ for some relation R_i of arity r_i, we have an edge $f_j(t) = a_j$ for all $j \leq r_i$.

OBSERVATION 1. *It is immediate to see that for every relational structure $\boldsymbol{D}$ we can compute Adjacency($\boldsymbol{D}$) in time $O(\|\boldsymbol{D}\|)$.*

Let $\mathcal{C}$ be a class of relational structures. We say that $\mathcal{C}$ has *bounded expansion* if the class $\mathcal{C}$' of adjacency graphs of structures from $\mathcal{C}$ has bounded expansion.

REMARK 1. *In the literature, for instance [10, 12], a class $\mathcal{C}$ of relational structures is said to have bounded expansion if the class of their Gaifman graphs has bounded expansion. Our definition can be shown to be equivalent to the usual one (see Appendix 8.3). As it gives directly an oriented graph, it is more convenient for us.*

Let $\Gamma_{\mathcal{C}'}$ be the function given by Definition 1 for $\mathcal{C}$'. The following lemma is immediate.

LEMMA 3. *Let $\mathcal{C}$ be a class of relational structures with bounded expansion and let $\mathcal{C}$' be the underlying class of adjacency graphs. Let $\phi(\bar{x}) \in$ FO. In time linear in the size*

of ϕ we can find a query $\psi(\bar{x})$ over $\sigma_{\mathcal{C}'}(0)$ such that for all $\boldsymbol{D} \in \mathcal{C}$ we have $\phi(\boldsymbol{D}) = \psi(\mathit{Adjacency}(\boldsymbol{D}))$.

As a consequence of Lemma 3 it follows that model checking, enumeration and counting of first-order queries over relational structures reduce to the graph case. Therefore in the rest of the paper we will only concentrate on the graph case (viewed as a functional structure), but the reader should keep in mind that all the results stated over graphs extend to relational structures via this lemma.

2.5 Normal form for quantifier-free first-order queries

We conclude this section by proving a normal form on quantifier-free FO formulas. This normal form will be the ground for all our algorithms later on. It basically says that, modulo performing some extra augmentation steps, a quantifier-free formula has a very simple form.

Fix class $\mathcal{C}$ of graphs with bounded expansion. Recall that we are now implicitly assuming that graphs are represented as functional structures.

A formula is *simple* if it does not contain atoms of the form $f(g(x))$, i.e. it does not contain any compositions of functions. Observe that, modulo augmentations, any formula can be transformed into a simple one.

LEMMA 4. *Let $\psi(\bar{x})$ be a formula over a recoloring of $\sigma_{\mathcal{C}}(p)$. Then, for $q = p + |\psi|$, there is a simple formula $\psi'(\bar{x})$ over a recoloring of $\sigma_{\mathcal{C}}(q)$ such that:*

for all $\vec{\boldsymbol{G}} \in \mathcal{C}_p$ there is a $\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ computable in time linear in $\|\vec{\boldsymbol{G}}\|$ such that $\psi(\vec{\boldsymbol{G}}) = \psi'(\vec{\boldsymbol{G}}')$.

PROOF. This is a simple consequence of transitivity. Any composition of two functions in $\vec{\mathbf{G}}$ represents a transitive pair of edges and becomes a single edge in the 1-augmentation $\vec{\mathbf{H}}$ of $\vec{\mathbf{G}}$. Then $f(g(x))$ over $\vec{\mathbf{G}}$ is equivalent to $h(x) \wedge P_{f,g,h}(x)$ over $\vec{\mathbf{H}}$, where the newly introduced color $P_{f,g,h}$ holds for those nodes v, for which the $f(g(v)) = h(v)$. As the nesting of compositions of functions is at most $|\psi|$, the result follows. The linear time computability is immediate from Lemma 2. □

We make one more observation before proving the normal form:

LEMMA 5. *Let $\vec{\boldsymbol{G}} \in \mathcal{C}_p$. Let u be a node of $\vec{\boldsymbol{G}}$. Let S be all the predecessors of u in $\vec{\boldsymbol{G}}$ and set $q = p + \Gamma_{\mathcal{C}}(p)$. Let $\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ be the $(q-p)$-th augmentation of $\vec{\boldsymbol{G}}$. There exists a linear order $<$ induced on S by $\vec{\boldsymbol{G}}'$, such that for all $v, v' \in S$, $v < v'$ implies $v' = f(v)$ is an edge of $\vec{\boldsymbol{G}}'$ for some function f from $\sigma_{\mathcal{C}}(q)$.*

PROOF. This is because all nodes of S are fraternal and the size of S is at most $\Gamma_{\mathcal{C}}(p)$. Hence, after one step of augmentation, all nodes of S are pairwise connected and, after at most $\Gamma_{\mathcal{C}}(p) - 1$ further augmentation steps, if there is a directed path from one node u of S to another node v of S, then there is also a directed edge from u to v. By induction on $|S|$ we show that there exists a node $u \in S$ such that for all $v \in S$ there is an edge from v to u. If $|S| = 1$ there is nothing to prove. Otherwise fix $v \in S$ and let $S' = S \setminus \{v\}$. By induction we get a u in S' satisfying the properties. If there is an edge from v to u, u also works for S and we are done. Otherwise there must be an edge from u to v. But then there is a path of length 2 from any node of S' to v. By transitivity this means that there is an edge from any node of S' to v and v is the node we are looking for.

We then set u as the minimal element of our order on S and we repeat this argument with $S \setminus \{u\}$. □

Lemma 5 justifies the following definition.

DEFINITION 2. *A p-type $\tau_p(x)$ is a quantifier-free conjunctive formula expressing all the relations between predecessors of a node x in some graph $\vec{\boldsymbol{G}} \in \mathcal{C}_p$ in the $(q-p)$-th augmentation $\vec{\boldsymbol{G}}'$ of $\vec{\boldsymbol{G}}$, where q is given by Lemma 5. More precisely, for every functions $f_i, f_j \in \sigma_{\mathcal{C}}(p)$, $\tau_p(x)$ contains at least one of the conjuncts $h_{i,j}(f_i(x)) = f_j(x)$ or $h_{j,i}(f_j(x)) = f_i(x)$, where $h_{i,j}$ and $h_{j,i}$ are function symbols from $\sigma_{\mathcal{C}}(q)$.*

In particular, a p-type τ induces a linear order on the predecessors of x as described by Lemma 5 ($f_i(x) < f_j(x)$ whenever $h_{i,j}(f_i(x)) = f_j(x)$ is a conjunct of τ) and moreover specifies all the relations between these predecessors in $\vec{\mathbf{G}}'$. Note that for a given p there are only finitely many possible p-types and that each of them can be specified with a conjunctive formula over $\sigma_{\mathcal{C}}(q)$.

We now state the normal form result.

PROPOSITION 1. *Let $\phi(\bar{x}y)$ be a simple quantifier-free query over a recoloring of $\sigma_{\mathcal{C}}(p)$. There exists q that depends only on p and ϕ and a quantifier-free query ψ over a recoloring of $\sigma_{\mathcal{C}}(q)$ that is a disjunction of formulas:*

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y), \tag{1}$$

where $\tau(y)$ contains a p-type of y; $\Delta^{=}(\bar{x}y)$ is either empty or contains one clause of the form $y = f(x_i)$ or one clause of the form $f(y) = g(x_i)$ for some suitable i, f and g; and $\Delta^{\neq}(\bar{x}y)$ contains arbitrarily many clauses of the form $y \neq f(x_i)$ or $f(y) \neq g(x_j)$. Moreover, ψ is such that:

for all $\vec{\boldsymbol{G}} \in \mathcal{C}_p$ there is a $\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ computable in time linear in $\|\vec{\boldsymbol{G}}\|$ with $\phi(\vec{\boldsymbol{G}}) = \psi(\vec{\boldsymbol{G}}')$.

PROOF. Set q as given by Lemma 5. We first put ϕ into a disjunctive normal form (DNF) and in front of each such disjunct we add a big disjunction over all possible p-types of y (recall that a type can be specified as a conjunctive formula). Let ϕ' be the resulting formula.

We deal with each disjunct of ϕ' separately.

Note that each disjunct is a query over $\sigma_{\mathcal{C}}(q)$ of the form:

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y),$$

where all sub-formulas except for $\Delta^{=}$ are as desired. Moreover, $\psi_1(\bar{x})$, $\Delta^{=}(\bar{x}y)$ and $\Delta^{\neq}(\bar{x}y)$ are in fact queries over $\sigma_{\mathcal{C}}(p)$. At this point $\Delta^{=}$ contains arbitrarily many clauses of the form $y = f(x_i)$ or $f(y) = g(x_i)$. If it contains at least one clause of the form $y = f(x_i)$, we can replace each other occurrence of y by $f(x_i)$ and we are done.

Assume now that $\Delta^{=}$ contains several conjuncts of the form $f_i(y) = g(x_k)$. Assume wlog that τ is such that $f_1(y) < f_2(y) < \cdots$, where $f_1(y), f_2(y), \cdots$ are all the predecessors of y from $\sigma_{\mathcal{C}}(p)$. Let i_0 be the smallest index i such that a clause of the form $f_i(y) = g(x_k)$ belongs to $\Delta^{=}$. We have $f_{i_0}(y) = g(x_k)$ in $\Delta^{=}$ and observe that τ specifies for $i < j$ a function $h_{i,j}$ in $\sigma_{\mathcal{C}}(q)$ such that $h_{i,j}(f_i(y)) = f_j(y)$. Then, as y is of type τ, a clause of the form $f_j(y) = h(x_{k'})$ with $i_0 < j$ is equivalent to $h_{i_0,j}(g(x_k)) = h(x_{k'})$.

Let ψ be the result of performing this operation on each disjunct of ϕ'.

Now, given $\vec{\mathbf{G}} \in \mathcal{C}_p$, let $\vec{\mathbf{G}}' \in \mathcal{C}_q$ be the $(q-p)$-th augementation of $\vec{\mathbf{G}}$. It is computable in time linear in $\vec{\mathbf{G}}$ by Lemma 2. By Lemma 5 we have $\phi(\vec{\mathbf{G}}) = \phi'(\vec{\mathbf{G}}')$. By construction we have $\psi(\vec{\mathbf{G}}') = \phi'(\vec{\mathbf{G}}')$ and the result follows. □

EXAMPLE A-4. *Let us see what Lemma 4 and the normalization algorithm do for $p = 0$ and some of the disjuncts of the query of Example A-3:*

In the case of $f(g(x)) = y$ note that by transitivity, in the augmented graph, this clause is equivalent to one of the form $y = h(x) \wedge P_{f,g,h}(x)$ (this case is handled by Lemma 4).

Consider now $\exists z \;\; f(z) = x \wedge g(z) = y$. It will be convenient to view this query when z plays the role of y in Proposition 1. Notice that in this case it is not in normal form as $\Delta^{=}$ contains two elements. However, the two edges $f(z) = x$ and $g(z) = y$ are fraternal. Hence, after one augmentation step, a new edge is added between x and y and we either have $y = h(x)$ or $x = h(y)$ for some h in the new signature.

Let $\tau_{h,f,g}(z)$ be a 0-type stating that $h(f(z)) = g(z)$ and $\tau_{h,g,f}(z)$ be a 0-type stating that $h(g(z)) = f(z)$. It is now easy to see that the query $\exists z \;\; f(z) = x \wedge g(z) = y$ is equivalent, in the augmented graph, to

$$\exists z \bigvee_h \; y = h(x) \wedge \tau_{h,f,g}(z) \wedge f(z) = x \;\; \vee \\ x = h(y) \wedge \tau_{h,g,f}(z) \wedge f(z) = x$$

3. MODEL CHECKING

In this section we show that the model checking problem of FO over a class of structures with bounded expansion can be done in time linear in the size of the structure. This gives a new proof of the result of [10]. Recall that by Lemma 3 it is enough to consider oriented graphs viewed as functional structures.

THEOREM 1. *[10] Let $\mathcal{C}$ be a class of graphs with bounded expansion and let ψ be a sentence of* FO. *Then, for all $\vec{\boldsymbol{G}} \in \mathcal{C}$, testing whether $\vec{\boldsymbol{G}} \models \psi$ can be done in time $O(\|\vec{\boldsymbol{G}}\|)$.*

The proof of Theorem 1 is done using a quantifier elimination procedure: given a query $\psi(\bar{x}y)$ with at least one free variable we can compute a quantifier-free query $\phi(\bar{x})$ that is "equivalent" to $\exists y \psi(\bar{x}y)$. Again, the equivalence should be understood modulo some augmentation steps for a number of augmentation steps depending only on $\mathcal{C}$ and $|\psi|$. When starting with a sentence ψ we end-up with ϕ being a boolean combination of formulas with one variable. Those can be easily tested in linear time in the size of the augmented structure, which in turns can be computed in time linear from the initial structure by Lemma 2. The result follows. We now state precisely the quantifier elimination step:

PROPOSITION 2. *Let $\mathcal{C}$ be a class of graphs with bounded expansion witnessed by the function $\Gamma_{\mathcal{C}}$. Let $\psi(\bar{x}y)$ be a quantifier-free formula over a recoloring of $\sigma_{\mathcal{C}}(p)$. Then one can compute a q and a quantifier-free formula $\phi(\bar{x})$ over a recoloring of $\sigma_{\mathcal{C}}(q)$ such that:*

for all $\vec{\boldsymbol{G}} \in \mathcal{C}_p$ there is a $\vec{\boldsymbol{G}}' \in \mathcal{C}_q$ such that:

$$\phi(\vec{\boldsymbol{G}}') = (\exists y \psi)(\vec{\boldsymbol{G}})$$

Moreover, $\vec{\boldsymbol{G}}'$ is computable in time $O(\|\vec{\boldsymbol{G}}\|)$.

Before going into details, we start with an outline of the proof. The reasoning is going to be as follows:

• Using Lemma 4 and Proposition 1 we argue that it suffices to show the quantifier elimination procedure only for $\psi(\bar{x}y)$ being of the special form given by (1), that is:

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y).$$

• In order to eliminate the existentially quantified variable y we somehow need to encode its existence in terms of properties of $\bar{x}$.

• In the easy case when ψ contains conjunct of the form $f(x_i) = y$, we can replace each occurrence of y with $f(x_i)$ and we are done.

• The most interesting case is when ψ contains conjunct of the form $f(y) = g(x_i)$. Then the algorithm proceeds as follows:

◦ it iterates through all nodes v of the graph (think of v as of a candidate for substituting the existentially quantified variable y) and in a sense "registers" its existence to node $f(v)$,

◦ given tuple $\bar{u}$ to be substituted for for $\bar{x}$ it is enough to only check nodes from the "list of registrants" of $g(u_i)$ as the possible candidates for y,

◦ unfortunately the above procedure could produce "lists of registrants" of arbitrary lengths, so we have to be more careful,

◦ therefore we limit the "registration" process and allow new nodes to register only if they are "different enough" (in terms of the sets of their predecessors) from the nodes that already registered,

◦ this way we define so called WITNESS sets that are of constant (i.e. independent from the size of $\vec{\mathbf{G}}$) sizes and such

that if there exists a valid node for y, there also exists such a node inside $\text{WITNESS}(g(u_i))$,

$\circ$ the rest of the argument is a way of encoding WITNESS sets by only recoloring the structure and not altering its functional part.

We now formalize the above approach:

PROOF OF PROPOSITION 2. Wlog (modulo augmentations, see Lemma 4 for details) we assume that ψ is simple.

We apply Proposition 1 to ψ and p and obtain a q and an equivalent formula in DNF, where each disjunct has the special form given by (1). As disjunction and existential quantification commute, it is enough to treat each part of the disjunction separately.

We thus assume that $\psi(\bar{x}y)$ is a quantifier-free conjunctive formula over a recoloring of $\sigma_{\mathcal{C}}(q)$ of the form (1):

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y).$$

We assume wlog that τ contains a p-type enforcing $f_1(y) < f_2(y) < \cdots$, where $f_1(y), f_2(y), \cdots$ are all the images of y by a function from $\sigma_{\mathcal{C}}(p)$. Moreover, for each $i < j$, τ contains an atom of the form $h_{i,j}(f_i(y)) = f_j(y)$ for some function $h_{i,j} \in \sigma_{\mathcal{C}}(q)$.

If $\Delta^{=}$ is $y = g(x_k)$ for some function g and some k, then we replace y with $g(x_k)$ everywhere in $\psi(\bar{x}y)$ resulting in a formula $\phi(\bar{x})$ having obviously the desired properties.

Assume now that $\Delta^{=}$ is $f(y) = g(x_i)$. Wlog assume that f is f_{i_0} in the order specified by the p-type τ and that $i = 1$. Hence we have $f_{i_0}(y) = g(x_1)$ in $\Delta^{=}$.

We will introduce extra colors in order to simulate all interactions between y and $\bar{x}$.

Let $\vec{\mathbf{G}}''$ be the $(q-p)$-th augmentation of $\vec{\mathbf{G}}$. We construct in time linear in $\|\vec{\mathbf{G}}''\|$ a set $\text{WITNESS}(v)$ for each v of $\vec{\mathbf{G}}'$ such that for all tuples $\bar{v}$ of $\vec{\mathbf{G}}''$, if $\vec{\mathbf{G}}'' \models \psi(\bar{v}u)$ for some node u, then there is a node $u' \in \text{WITNESS}(g(v_1))$ such that $\vec{\mathbf{G}}' \models \psi(\bar{v}u')$. Moreover, for all v, $|\text{WITNESS}(v)| \leq N$ where N is a number depending only on p. We then encode these witness sets using suitable extra colors.

Computation of the Witness function.

We start by initializing $\text{WITNESS}(v) = \emptyset$ for all v.

We then successively investigate all nodes u of $\vec{\mathbf{G}}''$ and do the following. If $\vec{\mathbf{G}}'' \models \neg\tau(u)$ then we move on to the next u. If $\vec{\mathbf{G}}'' \models \tau(u)$ then let $u_1, \cdots, u_l$ be the current value of $\text{WITNESS}(f_{i_0}(u))$.

Let β_p be $\alpha_{\mathcal{C}}(p)(\alpha_{\mathcal{C}}(p)+1)|\bar{x}|+1$.

Let i be minimal such that there exists j with $f_i(u_j) = f_i(u)$ and set $i = \alpha_{\mathcal{C}}(p)+1$ if such an i does not exists. Let $S_i = \{f_{i-1}(u_j) \mid f_i(u_j) = f_i(u)\}$, where $f_0(u_j)$ is u_j in the case where $i = 1$. If $|S_i| \leq \beta_p$ then we add u to $\text{WITNESS}(f_{i_0}(u))$.

The algorithm is linear time and the size of $\text{WITNESS}(v) \leq (\beta_p+1)^{\beta_p+1}$. It remains to show that it has the desired properties.

Analysis of the Witness function.

Assume $\vec{\mathbf{G}}'' \models \psi(\bar{v}u)$. If $u \in \text{WITNESS}(g(v_1))$ we are done. Otherwise note that $f_{i_0}(u) = g(v_1)$ and that $\vec{\mathbf{G}}'' \models \tau(u)$. Let i and S_i be as described in the algorithm when investigating u. As u was not added to $\text{WITNESS}(f_{i_0}(u))$, we must have $|S_i| > \beta_p$. Let $S_i = \{u_{i_1}, \cdots, u_{\beta_p}, \cdots\}$ be the corresponding elements of $\text{WITNESS}(g(v_1))$. Among these data values, for each j at most $\alpha_{\mathcal{C}}(p)$ of them may be a predecessor of v_j. Similarly, for each $i' \leq i$ and each j, at most $\alpha_{\mathcal{C}}(p)$ of them may be such that their image by $f_{i'}$ is a predecessor of v_j. For each $i' > i$ their image is exactly $f_{i'}(u)$ and it does not falsify any inequality conjuncts of ψ. Hence, at most $\alpha_{\mathcal{C}}(p)(\alpha_{\mathcal{C}}(p)+1)|\bar{v}|$ of them may falsify at least one of the inequality conjuncts of ψ. We can therefore find in $\text{WITNESS}(g(v_1))$ at least one element satisfying the formula, as $|S_i| > \alpha_{\mathcal{C}}(p)(\alpha_{\mathcal{C}}(p)+1)|\bar{v}|$.

Recoloring of $\vec{\mathbf{G}}''$.

Based on WITNESS we recolor $\vec{\mathbf{G}}''$ as follows. Let $\gamma_p = (\beta_p+1)^{\beta_p+1}$. For each $v \in \vec{\mathbf{G}}''$ we order $\text{WITNESS}(v)$. We can now speak of the i^{th} witness of v.

For each $i \leq \gamma_p$ we introduce a new unary predicate P_i and for each $u \in \vec{\mathbf{G}}''$ we set $P_i(u)$ if $\text{WITNESS}(u)$ contains at least i elements.

For each $i \leq \gamma_p$ and each $h, h' \in \alpha_{\mathcal{C}}(q)$ we introduce a new unary predicate $P_{i,h,h'}$ and for each $v \in \vec{\mathbf{G}}''$ we set $P_{i,h,h'}(v)$ if the i^{th} witness of $h(v)$ is an element u with $h'(u) = v$.

For each $i \leq \gamma_p$, $h \in \alpha_{\mathcal{C}}(q)$ we introduce a new unary predicate $Q_{i,h}$ and for each $v \in \vec{\mathbf{G}}''$ we set $Q_{i,h}(v)$ if the i^{th} witness of $h(v)$ is v.

We denote by $\vec{\mathbf{G}}'$ the resulting graph and notice that it can be computed in linear time from $\vec{\mathbf{G}}$.

Finally, note that if y is the i^{th} witness of $g(x_1)$, the equality $f_j(y) = h(x_k)$ with $j < i_0$ is equivalent over $\vec{\mathbf{G}}'$ to $h_{j,i_0}(h(x_k)) = g(x_1) \wedge P_{i,h_{j,i_0},f_j}(h(x_k))$ and the equality $y = h(x_k)$ is equivalent over $\vec{\mathbf{G}}'$ to $f_{i_0}(h(x_k)) = g(x_1) \wedge Q_{i,f_{i_0}}(h(x_k))$. From the definition of p-type, the equality $f_j(y) = h(x_k)$ with $j > i_0$ is equivalent to $h_{i_0,j}(g(x_1)) = h(x_k)$.

Computation of ϕ.

In view of the analysis above, $\psi(\bar{x}y)$ is equivalent to a formula:

$$\bigvee_{i \leq \gamma_p} \psi_1(\bar{x}) \wedge \psi^i(\bar{x})$$

where $\psi^i(\bar{x})$ checks that the i^{th} witness of $g(x_1)$ makes the initial formula true. In view of the above, this formula $\psi^i(\bar{x})$

is defined by

$$P_i(g(x_1)) \quad \wedge \bigwedge_{\substack{f_j(y)\neq h(x_k)\in\Delta^{\neq}\\ j<i_0}} \neg\big(h_{j,i_0}(h(x_k)) = g(x_1) \wedge P_{i,h_{j,i_0},f_j}(h(x_k))\big)$$
$$\wedge \bigwedge_{\substack{f_j(y)\neq h(x_k)\in\Delta^{\neq}\\ j\geq i_0}} h_{i_0,j}(g(x_1)) \neq h(x_k)$$
$$\wedge \bigwedge_{y\neq h(x_k)\in\Delta^{\neq}} \neg\big(f_{i_0}(h(x_k)) = g(x_1) \wedge Q_{i,f_{i_0}}(h(x_k))\big)$$

The special case when $\Delta^{=}$ is empty is a simpler version of the previous case, only this time it is enough to construct a set WITNESS which does not depend on v. For details see Appendix 8.4. □

EXAMPLE A-5. *Consider one of the quantified formulas as derived by Example A-4:*

$$\exists z \;\; y = h(x) \wedge \tau_{h,f,g}(z) \wedge f(z) = x$$

The resulting quantifier-free query has the form:

$$P(x) \wedge h(x) = y$$

where $P(x)$ is a newly introduced color saying "$\exists z \;\; \tau_{h,f,g}(z) \wedge f(z) = x$". The key point is that this new predicate can be computed in linear time by iterating through all nodes z, testing whether $\tau_{h,f,g}(z)$ is true and, if this is the case, coloring $f(z)$ with color P.

Applying the quantifier elimination process from inside out using Proposition 2 for each step and then applying Lemma 4 to the result yields:

THEOREM 2. *Let $\mathcal{C}$ be a class of graphs with bounded expansion. Let $\psi(\bar{x})$ be a query of* FO *over a recoloring of $\sigma_{\mathcal{C}}(0)$ with at least one free variable. Then one can compute a p and a simple quantifier-free formula $\phi(\bar{x})$ over a recoloring of $\sigma_{\mathcal{C}}(p)$ such that:*

for all $\vec{\boldsymbol{G}} \in \mathcal{C}$, we can construct in time $O(\|\vec{\boldsymbol{G}}\|)$ a graph $\vec{\boldsymbol{G}}' \in \mathcal{C}_p$ such that

$$\phi(\vec{\boldsymbol{G}}') = \psi(\vec{\boldsymbol{G}})$$

We will make use of the following useful consequence of Theorem 2:

COROLLARY 1. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\psi(\bar{x})$ be a formula of* FO *with at least one free variable. Then, for all $\vec{\boldsymbol{G}} \in \mathcal{C}$, after a preprocessing in time $O(\|\vec{\boldsymbol{G}}\|)$, we can test, given $\bar{u}$ as input, whether $\vec{\boldsymbol{G}} \models \psi(\bar{u})$ in constant time.*

PROOF. By Theorem 2 it is enough to consider quantifier-free simple queries. Hence it is enough to consider a query consisting of a single atom of either $P(x)$ or $P(f(x))$ or $x = f(y)$ or $f(x) = g(y)$.

During the preprocessing phase we associate with each node v of the input graph a list $L(v)$ containing all the predicates satisfied by v and all the images of v by a function symbol from the signature. This can be computed in linear time by enumerating all relations of the database and updating the appropriate lists with the corresponding predicate or the corresponding image.

Now, because we use the RAM model, given u we can in constant time recover the list $L(u)$. Using those lists it is immediate to check all atoms of the formula in constant time. □

Theorem 1 is a direct consequence of Theorem 2 and Corollary 1: Starting with a sentence, and applying Theorem 2 for eliminating quantifiers from inside out we end up with a Boolean combination of formulas with one variable. Each such formula can be tested in $O(\|\vec{\mathbf{G}}\|)$ by iterating through all nodes v of $\vec{\mathbf{G}}$ and in constant time (using Corollary 1) checking if v can be substituted for the sole existentially quantified variable.

On top of Theorem 1 the following corollary is immediate from Theorem 2 and Corollary 1:

COROLLARY 2. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\psi(x)$ be a formula of* FO *with one free variable. Then, for all $\vec{\boldsymbol{G}} \in \mathcal{C}$, computing the set $\psi(\vec{\boldsymbol{G}})$ can be done in time $O(\|\vec{\boldsymbol{G}}\|)$.*

4. ENUMERATION

In this section we consider first-order formulas with free variables and show that we can enumerate their answers over any class with bounded expansion with constant delay. Moreover, assuming a linear order on the domain of the input structure, we will see that the answers can be output in the lexicographical order. As before we only state the result for graphs, but it immediately extends to arbitrary structures by Lemma 3. Recall that we assumed (without loss of generality) the presence of a linear order of the domain.

THEOREM 3. *Let $\mathcal{C}$ be a class of graphs with bounded expansion and let $\phi(\bar{x})$ be a first-order query over $\sigma_{\mathcal{C}}(0)$. Then the enumeration problem of ϕ over $\mathcal{C}$ is in* CONSTANT-DELAY$_{lin}$. *Moreover the answers to ϕ can be output in lexicographical order.*

Before going into details, we start with an outline of the proof. The reasoning is going to be as follows:

- The proof is by induction on the number of free variables.
- The case $k = 1$ is done by Corollary 2.
- For $k > 1$, using the normalization and quantification procedures of the previous sections, it is enough to consider quantifier-free queries $\psi(\bar{x}y)$ of the form:

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y).$$

We further set $\psi''(\bar{x})$ the formula $\exists y \psi(\bar{x}y)$.

- In the easy case when ψ contains conjunct of the form $f(x_i) = y$, we enumerate $\psi''(\bar{x})$ by induction and append $f(x_i)$ to each resulting tuple.

• The most interesting case is when ψ contains conjunct of the form $f(y) = g(x_i)$. Then the algorithm proceeds as follows:

◦ It enumerates all the solutions of $\psi''(\bar{x})$ by induction and appends to it all the relevant y.

◦ For this it computes, during the preprocessing phase, several successor functions among nodes, such that for each $\bar{x}$, at least one of them will enumerate the associated y.

◦ The key point is that only finitely many successor functions need to be precomputed and that the suitable one can be found by looking only at $\bar{x}$.

We now formalize the above approach:

PROOF. Fix a class $\mathcal{C}$ of graphs with bounded expansion and a query $\phi(\bar{x})$ with k free variables. Let $\vec{\mathbf{G}}$ be the input graph and V be its set of vertices.

The proof is by induction on the number of free variables. The case $k = 1$ is done by Corollary 2.

Assume now that $k > 1$ and that $\bar{x}$ and y are the free variables of ϕ, where $|\bar{x}| = k - 1$.

We apply Theorem 2 to get a simple quantifier-free query $\varphi(\bar{x}y)$ and a structure $\vec{\mathbf{G}}' \in \mathcal{C}_p$, for some p that does not depend on $\vec{\mathbf{G}}$, such that $\varphi(\vec{\mathbf{G}}') = \phi(\vec{\mathbf{G}})$ and $\vec{\mathbf{G}}'$ can be computed in linear time from $\vec{\mathbf{G}}$.

We normalize the resulting simple quantifier-free query using Proposition 1, and obtain an equivalent quantifier-free formula ψ and a structure $\vec{\mathbf{G}}'' \in \mathcal{C}_q$, where q depends only on p and φ, $\vec{\mathbf{G}}''$ can be computed in linear time from $\vec{\mathbf{G}}'$, $\varphi(\vec{\mathbf{G}}') = \psi(\vec{\mathbf{G}}'')$ and ψ is a disjunction of formulas of the form (1):

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y),$$

where $\Delta^{=}(\bar{x}y)$ is either empty or contains one clause of the form $y = f(x_i)$ or one clause of the form $f(y) = g(x_i)$ for some suitable i, f and g; and $\Delta^{\neq}(\bar{x}y)$ contains arbitrarily many clauses of the form $y \neq f(x_i)$ or $f(y) \neq g(x_j)$.

By Lemma 1 it is enough to show that we can enumerate each disjunct separately. In the sequel we then assume that ψ has the form described in (1). We let $\psi'(y)$ be the formula $\exists \bar{x} \psi(\bar{x}y)$ and $\psi''(\bar{x})$ the formula $\exists y \psi(\bar{x}y)$.

If $\Delta^{=}$ contains an equality of the form $y = f(x_i)$ then we replace y by $f(x_i)$ in τ and $\Delta^{\neq}$, enumerate by induction the formula ψ'' and replace each of its output $\bar{a}$ with $(\bar{a} f(a_i))$ in order to obtain the desired constant delay enumeration algorithm. We therefore now assume that $\Delta^{=}$ does not contain such equality.

We now define two functions $L : V \to 2^V$ and $W : V^{k-1} \to V$ depending on whether $\Delta^{=}$ is empty or consists of a single clause of the form $f(y) = g(x_i)$. If $\Delta^{=}$ is empty we pick an arbitrary node w in $\vec{\mathbf{G}}''$ and set $L(w) = \psi'(\vec{\mathbf{G}}'')$, $L(v) = \emptyset$ for $v \neq w$, and $W(\bar{v}) = w$ for all tuples $\bar{v}$. If $\Delta^{=} = \{f(y) = g(x_i)\}$ we set $W(\bar{v}) = g(v_i)$ for all tuples $\bar{v}$ and define L using the following procedure. We initialize $L(v)$ to $\emptyset$ for each $v \in V$. Then, for each $v \in \psi'(\vec{\mathbf{G}}'')$, we add v to the set $L(f(v))$.

Notice that L can be computed in time linear in $\|\vec{\mathbf{G}}''\|$ (using Corollary 2), that each list $L(v)$ is sorted with respect to the linear order on the domain and that, given $\bar{v}$, $W(\bar{v})$ can be computed in constant time. Moreover, for each $\bar{v}u$, $\vec{\mathbf{G}}'' \models \psi(\bar{v}u)$ implies $u \in L(W(\bar{v}))$ and if $u \in L(W(\bar{v}))$ then $\Delta^{=}(\bar{v}u)$ is true.

By induction we can enumerate $\psi''(\bar{x})$ with constant delay.

On top of the linear time preprocessing necessary for enumerating ψ'' we do the following extra preprocessing. We first compute $L(v)$ for all $v \in V$. Then, for each $v \in V$, we perform the following procedure on $L(v)$. Each procedure will work in time linear in the size of $L(v)$, hence the total preprocessing will take time $O(|V|)$.

Fix v and set $L = L(v)$. We denote by $<$ the order on L. (Recall that this order is consistent with the initial order on the domain.)

For $S_1, \ldots, S_{\alpha_{\mathcal{C}}(q)} \subseteq V$ we define $\text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$ to be the first element $w \geq u$ of L such that $f_1(w) \notin S_1, \ldots,$ and $f_{\alpha_{\mathcal{C}}(q)}(w) \notin S_{\alpha_{\mathcal{C}}(q)}$. If such w does not exist, the value of $\text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$ is NULL. When all S_i are empty, we write $\text{next}_{\emptyset}(u)$ and by the above definitions we always have $\text{next}_{\emptyset}(u) = u$. We denote such functions as *shortcut pointers of* u. We write $\text{NEXT}_{f_1, S'_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S'_{\alpha_{\mathcal{C}}(q)}}(u) \preceq \text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$ if for each $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ we have $S'_i \subseteq S_i$. Note that for a given u the $\preceq$ relation is a partial order on the set of shortcut pointers of u. A trivial observation is that if $\text{NEXT}_{f_1, S'_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S'_{\alpha_{\mathcal{C}}(q)}}(u) \preceq \text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$, then $\text{NEXT}_{f_1, S'_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S'_{\alpha_{\mathcal{C}}(q)}}(u) \leq \text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$. The *size* of a shortcut pointer $\text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$ is the sum of sizes of the sets S_i.

In order to avoid writing too long expressions containing shortcut pointers, we introduce the following abbreviations:

- $\text{NEXT}_{f_1, S_1, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$ is denoted with $\text{NEXT}_{\vec{S}}(u)$,
- $\text{NEXT}_{f_1, S_1, \ldots, f_i, S_i \cup \{u_i\}, \ldots, f_{\alpha_{\mathcal{C}}(q)}, S_{\alpha_{\mathcal{C}}(q)}}(u)$ is denoted with $\text{NEXT}_{\vec{S}[S_i += \{u_i\}]}(u)$.

Set $\beta_q = (k-1) \cdot \alpha_{\mathcal{C}}(q)^2$.

Computing all shortcut pointers of size β_q would take more than linear time. We therefore compute a subset of those, denoted SC_L, that will be sufficient for our needs. SC_L is defined in an inductive manner. For all u, $\text{next}_{\emptyset}(u) \in \text{SC}_L$. Moreover, if the shortcut pointer NULL $\neq \text{NEXT}_{\vec{S}}(u) \in \text{SC}_L$ and has a size smaller than β_q, then, for each i, $\text{NEXT}_{\vec{S}[S_i += \{u_i\}]}(u) \in \text{SC}_L$, where $u_i = f_i(\text{NEXT}_{\vec{S}}(u))$. We then say that $\text{NEXT}_{\vec{S}}(u)$ is the *origin* of $\text{NEXT}_{\vec{S}[S_i += \{u_i\}]}(u)$. Note that SC_L contains all the shortcut pointers of the form $\text{NEXT}_{f_i, \{f_i(u)\}}(u)$ for $u \in L$ and these are exactly the shortcut pointers of u of size 1. By $\text{SC}_L(u) \subseteq \text{SC}_L$ we denote the shortcut pointers of u that are in SC_L.

The set SC_L has the following properties:

CLAIM 1. *Let* $\text{NEXT}_{\vec{S}}(u)$ *be a shortcut pointer of size not greater than* β_q. *Then there exists* $\text{NEXT}_{\vec{S}'}(u) \in \text{SC}_L$ *such that* $\text{NEXT}_{\vec{S}}(u) = \text{NEXT}_{\vec{S}'}(u)$. *Moreover, such* $\text{NEXT}_{\vec{S}'}(u)$ *can be found in constant time.*

PROOF. The desired shortcut pointer is $\text{NEXT}_{\vec{S}'}(u) \in \text{SC}_L$ that is maximal in terms of size shortcut pointer of u such that $\text{NEXT}_{\vec{S}'}(u) \preceq \text{NEXT}_{\vec{S}}(u)$. (See Appendix 8.5.) □

CLAIM 2. *There exists a constant* $\zeta(q,k)$ *such that for every node* u *we have* $|\text{SC}_L(u)| \leq \zeta(q,k)$.

PROOF. The proof is a direct consequence of the recursive definition of $\text{SC}_L(u)$. (See Appendix 8.5.) □

The following claim guarantees that SC_L can be computed in linear time and has therefore a linear size.

CLAIM 3. SC_L *can be computed in time linear in* $|L|$.

PROOF. SC_L can be constructed in an inductive manner starting from the last node on the list L and moving backward. Claim 1 plays the key role in constructing each shortcut pointer in constant time, while Claim 2 guarantees that the total size of SC_L is linear in $|L|$. (See Appendix 8.5.) □

The computation of SC_L concludes the preprocessing phase and it follows from Claim 3 that it can be done in linear time. We now turn to the enumeration phase.

We enumerate one by one the solutions to $\psi''(\bar{x})$ by simulating the enumeration algorithm obtained from the induction.

Having a solution $\bar{v}$ to ψ'' by construction we know that all nodes u such that $\vec{\mathbf{G}}'' \models \psi(\bar{v}u)$ are in $L = L(W(\bar{v}))$. Recall also that all elements $u \in L$ make $\tau(u) \wedge \Delta^{=}(\bar{v}u)$ true. For $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ we set $S_i = \{g(v_j) : g(x_j) \neq f_i(y)$ is a conjunct of $\Delta^{\neq}\}$. Starting with u the first node of the sorted list L, we apply the following procedure:

1. If $u =$ NULL, finish the nested enumeration procedure for $\bar{v}$. If not, let $\text{NEXT}_{\vec{S}'}(u)$ be the shortcut pointer from the application of Claim 1 to $\text{NEXT}_{\vec{S}}(u)$. Set $u' = \text{NEXT}_{\vec{S}'}(u)$. If $u' =$ NULL, finish the nested enumeration procedure for $\bar{v}$.
2. If $\vec{\mathbf{G}}'' \models \psi(\bar{v}u')$, output $(\bar{v}u')$.
3. Reinitialize u to the successor of u' in L and continue with Step 1.

We now show that the algorithm is correct, i.e. that it outputs all $\psi(\vec{\mathbf{G}}'')$ with no repetition.

The algorithm clearly outputs a subset of $\psi(\vec{\mathbf{G}}'')$ as it tests whether $\vec{\mathbf{G}}'' \models \psi(\bar{v}u')$ before outputting tuple $(\bar{v}u')$.

By the definition, list L contains no duplicates and as the algorithm moves only forward on that list, there are no repetitions during the output process.

By the definition of sets S_i and $\text{NEXT}_{\vec{S}}(u)$, for each $u \leq w < u'$ there is a suitable i and j such that $g(v_j) = f_i(w)$ and $g(x_j) \neq f_i(y)$ is a conjunct of $\Delta^{\neq}$. This way the algorithm does not skip any solutions at Step 1 and so it outputs exactly $\psi(\vec{\mathbf{G}}'')$.

It remain to show that there is a constant time between any two outputs.

By construction, for each $\bar{v}$, $L = L(W(\bar{v}))$ contains an element u such that $(\bar{v}u)$ is a solution. We therefore need to show that there is a constant time between any two outputs involving an element in L. Step 1 takes constant time due to Claim 1. From there the algorithm either immediately outputs a solution at Step 2 or jumps to Step 3. This means that $\vec{\mathbf{G}}'' \not\models \psi(\bar{v}u')$, but from the definitions of list L, sets S_i and shortcut pointers $\text{NEXT}_{\vec{S}}(u)$ it is only the $\Delta^{\neq}$ that is falsified and it is because of an inequality of the form $y \neq g(x_j)$ for some suitable g and j (where g may possibly be identity). This implies that $u' = g(v_j)$. As all the elements on L are distinct, the algorithm can skip over Step 2 up to $(k-1) \cdot (\alpha_{\mathcal{C}}(q)+1)$ times for each tuple $\bar{v}$ (there are up to that many different images of nodes from $\bar{v}$ under $\alpha_{\mathcal{C}}(q)$ different functions and the initial values of $\bar{v}$). This way the delay is bounded by up to $k \cdot (\alpha_{\mathcal{C}}(q)+1)$ consecutive applications of Claim 1 and is in fact constant.

As the list L was sorted with respect to the linear order on the domain, it is clear that the enumeration procedure outputs the set of solutions in lexicographical order.

This concludes the proof of the theorem. □

□

5. COUNTING

In this section we investigate the problem of counting the number of solutions to a query, i.e. computing $|q(\mathbf{D})|$. As usual we only state and prove our results over graphs but they generalize to arbitrary relational structures via Lemma 3. The proof goes by induction on the number of free variables and follows the same outline as for enumeration. It only replaces the step of enumeration pre-computing several successor functions by a combinatorial argument counting their number.

THEOREM 4. *Let* $\mathcal{C}$ *be class of graphs with bounded expansion and let* $\phi(\bar{x})$ *be a first-order formula. Then, for all* $\vec{\boldsymbol{G}} \in \mathcal{C}$, *we can compute* $|\phi(\vec{\boldsymbol{G}})|$ *in time* $O(\|\vec{\boldsymbol{G}}\|)$.

PROOF. The key idea is to prove a weighted version of the desired result. Assume $\phi(\bar{x})$ has exactly k free variables and for $1 \leq i \leq k$ we have functions $\#_i : V \to \mathbb{N}$. We will compute in time linear in $\|\vec{\mathbf{G}}\|$ the following number:

$$|\phi(\vec{\mathbf{G}})|_{\#} := \sum_{\bar{u} \in \phi(\vec{\mathbf{G}})} \prod_{1 \leq i \leq k} \#_i(u_i).$$

By setting all $\#_i$ to be constant functions with value 1 we get the regular counting problem. Hence Theorem 4 is an immediate consequence of the next lemma.

LEMMA 6. *Let $\mathcal{C}$ be class of graphs with bounded expansion and let $\phi(\bar{x})$ be a first-order formula with exactly k free variables.*
For $1 \leq i \leq k$ let $\#_i : V \to \mathbb{N}$ be functions such that for each v the value of $\#_i(v)$ can be computed in constant time. Then, for all $\vec{\mathbf{G}} \in \mathcal{C}$, we can compute $|\phi(\vec{\mathbf{G}})|_{\#}$ in time $O(\|\vec{\mathbf{G}}\|)$.

PROOF. The proof is by induction on the number of free variables.

The case $k = 1$ is trivial: in time linear in $\|\vec{\mathbf{G}}\|$ we compute $\phi(\vec{\mathbf{G}})$ using Corollary 2. By hypothesis, for each $v \in \phi(\vec{\mathbf{G}})$, we can compute the value of $\#_1(v)$ in constant time. Therefore the value

$$|\phi(\vec{\mathbf{G}})|_{\#} = \sum_{v \in \phi(\vec{\mathbf{G}})} \#_1(v)$$

can be computed in linear time as desired.

Assume now that $k > 1$ and that $\bar{x}$ and y are the free variables of ϕ, where $|\bar{x}| = k - 1$.

We apply Theorem 2 to get a simple quantifier-free query $\varphi(\bar{x}y)$ and a structure $\vec{\mathbf{G}}' \in \mathcal{C}_p$, for some p that does not depend on $\vec{\mathbf{G}}$, such that $\varphi(\vec{\mathbf{G}}') = \phi(\vec{\mathbf{G}})$ and $\vec{\mathbf{G}}'$ can be computed in linear time from $\vec{\mathbf{G}}$. Note that $|\phi(\vec{\mathbf{G}})|_{\#} = |\varphi(\vec{\mathbf{G}}')|_{\#}$, so it is enough to compute the latter value.

We normalize the resulting simple quantifier-free query using Proposition 1, and obtain an equivalent quantifier-free formula ψ and a structure $\vec{\mathbf{G}}'' \in \mathcal{C}_q$, where q depends only on p and φ, $\vec{\mathbf{G}}''$ can be computed in linear time from $\vec{\mathbf{G}}'$, $\varphi(\vec{\mathbf{G}}') = \psi(\vec{\mathbf{G}}'')$ and ψ is a disjunction of formulas of the form (1):

$$\psi_1(\bar{x}) \wedge \tau(y) \wedge \Delta^{=}(\bar{x}y) \wedge \Delta^{\neq}(\bar{x}y), \tag{1}$$

where $\Delta^{=}(\bar{x}y)$ is either empty or contains one clause of the form $y = f(x_i)$ or one clause of the form $f(y) = g(x_i)$ for some suitable i, f and g; and $\Delta^{\neq}(\bar{x}y)$ contains arbitrarily many clauses of the form $y \neq f(x_i)$ or $f(y) \neq g(x_j)$. Note that $|\varphi(\vec{\mathbf{G}}')|_{\#} = |\psi(\vec{\mathbf{G}}'')|_{\#}$, so it is enough to compute the latter value.

Observe that it is enough to solve the weighted counting problem for each disjunct separately, as we can then combine the results using a simple inclusion-exclusion reasoning. In the sequel we then assume that ψ has the form described in (1).

The proof now goes by induction on the number of inequalities in $\Delta^{\neq}$. While the inductive step turns out to be fairly easy, the difficult part is the base step of the induction.

We start with proving the inductive step. Let $g(y) \neq f(x_i)$ be an arbitrary inequality from $\Delta^{\neq}$ (where g might possibly be the identity). Let ψ^- be ψ with this inequality removed and $\psi^+ = \psi^- \wedge g(y) = f(x_i)$. Of course ψ and ψ^+ have disjoint sets of solutions and we have:

$$|\psi(\vec{\mathbf{G}}'')|_{\#} = |\psi^-(\vec{\mathbf{G}}'')|_{\#} - |\psi^+(\vec{\mathbf{G}}'')|_{\#}.$$

Note that ψ^- and ψ^+ have one less conjunct in $\Delta^{\neq}$. The problem is that ψ^+ is not of the form (1) as it may now contain two elements in $\Delta^{=}$. However it can be seen that the removal of the extra equality in $\Delta^{=}$ as described in the proof of Proposition 1 does not introduce any new elements in $\Delta^{\neq}$. See also Appendix 8.6. We can therefore remove the extra element in Δ^{+} and assume that ψ^+ has the desired form. We can now use the inductive hypothesis on the size of $\Delta^{\neq}$ to both ψ^- and ψ^+ in order to compute both $|\psi^-(\vec{\mathbf{G}}'')|_{\#}$ and $|\psi^+(\vec{\mathbf{G}}'')|_{\#}$ and derive $|\psi(\vec{\mathbf{G}}'')|_{\#}$.

It remains to show the base of the inner induction. In the following we assume that $\Delta^{\neq}$ is empty. The rest of the proof is a case analysis on the content of $\Delta^{=}$. Due to space limitations we analyze in full details only the situation when $\Delta^{=}$ consists of an atom of the form $y = f(x_1)$. Although this case is not the most difficult, we find it the most explanatory and still generic enough.

Assume then that $\Delta^{=}$ consists of an atom of the form $y = f(x_1)$.

Note that the solutions to ψ are of the form $(\bar{a}f(a_1))$. We have:

$$\begin{aligned}
|\psi(\vec{\mathbf{G}}'')|_{\#} &= \sum_{(\bar{u}v) \in \psi(\vec{\mathbf{G}}'')} \left(\#_k(v) \prod_{1 \leq i \leq k-1} \#_i(u_i) \right) \\
&= \sum_{(\bar{u}f(u_1)) \in \psi(\vec{\mathbf{G}}'')} \left(\#_k(f(u_1)) \prod_{1 \leq i \leq k-1} \#_i(u_i) \right) \\
&= \sum_{(\bar{u}f(u_1)) \in \psi(\vec{\mathbf{G}}'')} \left(\#_1(u_1)\#_k(f(u_1)) \prod_{2 \leq i \leq k-1} \#_i(u_i) \right)
\end{aligned}$$

In linear time we now iterate through all nodes u in $\vec{\mathbf{G}}''$ and set

$$\begin{aligned}
\#'_1(u) &:= \#_1(u) \cdot \#_k(f(u)) \\
\#'_i(u) &:= \#_i(u) \qquad \text{for } 2 \leq i \leq k-1.
\end{aligned}$$

Let $\vartheta(\bar{x})$ be ψ with all occurrences of y replaced with $f(x_1)$. We then have:

$$\begin{aligned}
|\psi(\vec{\mathbf{G}}'')|_{\#} &= \sum_{(\bar{u}f(u_1)) \in \psi(\vec{\mathbf{G}}'')} \left(\#'_1(u_1) \prod_{2 \leq i \leq k-1} \#'_i(u_i) \right) \\
&= \sum_{\bar{u} \in \vartheta(\vec{\mathbf{G}}'')} \prod_{1 \leq i \leq k-1} \#'_i(u_i) \\
&= |\vartheta(\vec{\mathbf{G}}'')|_{\#'}
\end{aligned}$$

By induction on the number of free variables, as $\#'_i(u)$ can be computed in constant time for each i and u, we can compute $|\vartheta(\vec{\mathbf{G}}'')|_{\#'}$ in time linear in $\|\vec{\mathbf{G}}''\|$ and we are done.

For the case when $\Delta^{=}$ consists of an atom $g(y) = f(x_1)$ we use the same approach, only this time we set:

$$\#'_1(u) := \#_1(u) \cdot \sum_{\substack{\{v \in (\exists \bar{x} \psi(\bar{x}y))(\vec{\mathbf{G}}'') \\ g(v) = u\}}} \#_k(v)$$

$$\#'_i(u) := \#_i(u) \qquad \text{for } 2 \leq i \leq k-1$$

and conclude with $|(\exists y \psi(\bar{x}y))(\vec{\mathbf{G}}'')|_{\#'} = |\psi(\vec{\mathbf{G}}'')|_{\#}$. For more details on this and the case when $\Delta^{=}$ is empty, see Appendix 8.6. □

As we said earlier, Theorem 4 is an immediate consequence of Lemma 6. □ □

6. CONCLUSIONS

Queries written in first-order logic can be efficiently processed over the class of structures having bounded expansion. We have seen that over this class the problems investigated in this paper can be computed in time linear in the size of the input structure. The constant factor however is not very good. The approach taken here, as well as the ones of [10, 12], yields a constant factor that is a tower of exponentials whose height depends on the size of the query. This nonelementary constant factor is unavoidable already on the class of unranked trees, assuming FPT≠AW[∗] [11]. In comparison, this factor can be triply exponential in the size of the query in the bounded degree case [20, 13].

It is possible that the results presented here can be generalized to a larger class of structures. In [18] the class of nowhere dense graphs was introduced and it generalizes the notion of bounded expansion. It seems that nowhere dense graphs do enjoy good algorithmic properties. However, we do not know yet whether the model checking problem of first-order logic can be done in linear time over nowhere dense structures. Actually, we do not even know whether the model checking problem is Fixed Parameter Tractable (FPT) over nowhere dense graphs.

The class of nowhere dense structures seems to be the limit for having good algorithmic properties for first-order logic. Indeed, it is known that the model checking problem of first-order logic over a class of structures that is not nowhere dense cannot be FPT [15] (modulo some complexity assumptions and closure of the class under substructures).

For structures of bounded expansion, an interesting open question is whether a sampling of the solutions can be performed in linear time. For instance: can we compute the j-th solution in constant time after a linear preprocessing? This can be done in the bounded degree case [7] and in the bounded treewidth case [5]. We leave the bounded expansion case for future research.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995.

[3] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy Problems for Tree-Decomposable Graphs. *J. of Algorithms*, 12(2):308–340, 1991.

[4] Guillaume Bagan. MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay. In *Conf. on Computer Science Logic (CSL)*, pages 167–181, 2006.

[5] Guillaume Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques*. PhD thesis, Université de Caen, 2009.

[6] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *Conf. on Computer Science Logic (CSL)*, pages 208–222, 2007.

[7] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth solution of a first-order query. *RAIRO Theoretical Informatics and Applications*, 42(1):147–164, 2008.

[8] Bruno Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

[9] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. on Computational Logic (ToCL)*, 8(4), 2007.

[10] Zdeněk Dvořák, Daniel Král, and Robin Thomas. Deciding First-Order Properties for Sparse Graphs. In *Symp. on Foundations of Computer Science (FOCS)*, pages 133–142, 2010.

[11] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

[12] Martin Grohe and Stephan Kreutzer. *Model Theoretic Methods in Finite Combinatorics*, chapter Methods for Algorithmic Meta Theorems. American Mathematical Society, 2011.

[13] Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science (LMCS)*, 7(2), 2011.

[14] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. on Computational Logic (ToCL)*, to appear.

[15] Stephan Kreutzer and Anuj Dawar. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:131, 2009.

[16] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008.

[17] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. Algorithmic aspects. *Eur. J. Comb.*, 29(3):777–791, 2008.

[18] Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European J. of Combinatorics*, 32(4):600–617, 2011.

[19] Christos H. Papadimitriou and Mihalis Yannakakis. On the Complexity of Database Queries. *J. on Computer and System Sciences (JCSS)*, 58(3):407–427, 1999.

[20] Detlef Seese. Linear Time Computable Problems and

First-Order Descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

8. APPENDIX

8.1 Graphs with bounded expansion

To avoid confusion with the notion of size of a structure, we use the following notion in the case of graphs: we write $|\mathbf{G}|_{\text{VERT}}$ to denote the number of nodes of $\mathbf{G}$ (i.e. the size of V from the sequel), while we write $|\mathbf{G}|_{\text{EDGE}}$ to denote the number of edges of $\mathbf{G}$ (i.e. the size of E from the sequel).

Let $\mathbf{G} = (V, E)$ be an uncolored graph. It is *unoriented* if for each $(u, v) \in E$ we also have that $(v, u) \in E$. Assume $\mathbf{G}$ is unoriented. For any node $v \in V$ and any $r \in \mathbb{N}$ we denote by $B_r(v)$ the r-ball around v, i.e. the set of nodes of $\mathbf{G}$ that are reachable from v by paths of lengths up to r. We say that a graph $\mathbf{H}$ is a r-minor of $\mathbf{G}$ if all the nodes $v_1, \ldots, v_k$ of $\mathbf{H}$ are also nodes of $\mathbf{G}$ and for $1 \leq i \leq k$ there exists $1 \leq r_i \leq r$, such that, inside $\mathbf{G}$, the balls $B_{r_1}(v_1), \ldots, B_{r_k}(v_k)$ are pairwise non-overlapping and there is an edge between v_i and v_j in $\mathbf{H}$ iff there is an edge in $\mathbf{G}$ from a node of $B_{r_i}(v_i)$ to a node of $B_{r_j}(v_j)$. The set of all r-minors of $\mathbf{G}$ is denoted by $\mathbf{G}\nabla_r$. For a graph $\mathbf{G}$ the *greatest reduced average density (grad) of $\mathbf{G}$ with rank r* is:

$$\nabla_r(\mathbf{G}) = \max_{\mathbf{H} \in \mathbf{G}\nabla_r} \frac{|\mathbf{H}|_{\text{EDGE}}}{|\mathbf{H}|_{\text{VERT}}}.$$

THEOREM 5. *[16] Let $\mathcal{C}$ be a class of graphs. The following conditions are equivalent:*

1. *there exists a computable function $f : \mathbb{N} \to \mathbb{R}$ such that for all graphs $\boldsymbol{G} \in \mathcal{C}$ and for all $r \in \mathbb{N}$ we have:*
$$\nabla_r(\boldsymbol{G}) \leq f(r),$$
2. *$\mathcal{C}$ has bounded expansion.*

In fact in [16] it is stated the other way around: the initial definition of class of graphs with bounded expansion is the one from point 1 from the above theorem and its equivalence with Definition 1 is a theorem there.

8.2 A remark about $\sigma_{\mathcal{C}}(i)$

It would be tempting to set $\sigma_{\mathcal{C}}(i)$ to be the functional structure with $\Gamma_{\mathcal{C}}(i)$ functional symbols that would then be used to encode up to $\Gamma_{\mathcal{C}}(i)$ predecessors of each node. We could then easily have properties 1 and 2, but it would not be the case for property 3. To see this consider the following simple example:

EXAMPLE 1. *$\mathcal{C}$ is such that $\Gamma_{\mathcal{C}}(i) = 2$ for all i and $\boldsymbol{G} \in \mathcal{C}$ is defined as $\boldsymbol{G} = (V = \{u, v, w\}, E = \{(u, w), (v, w)\})$. Wlog assume that the functional structure describing $\boldsymbol{G}$ is $\vec{\boldsymbol{G}}_1 = (V = \{u, v, w\}, \{f_1(w) = u\}, \{f_2(w) = v\})$ and so we need to show a transitive fraternal augmentation $\vec{\boldsymbol{G}} = \vec{\boldsymbol{G}}_0 \subseteq \vec{\boldsymbol{G}}_1 \subseteq \vec{\boldsymbol{G}}2 \subseteq \ldots$ with the desired properties 1, 2 and 3.*

Note that (u, v) is a fraternal pair of nodes in $\vec{\boldsymbol{G}}_1$ and so $\vec{\boldsymbol{G}}_2$ must describe an edge between u and v (in at least one of the directions). To match property 2, $\vec{\boldsymbol{G}}_2$ must contain $\vec{\boldsymbol{G}}_1$ and wlog we may assume that $\vec{\boldsymbol{G}}_2$ contains $(V = \{u, v, w\}, \{f_1(w) = u, f_1(u) = v\}, \{f_2(w) = v\})$.

Consider now the following query ϕ over $\sigma_{\mathcal{C}}(0)$:

$\phi(x, y) \equiv f_1(x) = y \vee f_2(x) = z$.

Clearly $(u, v) \in \phi(\vec{\boldsymbol{G}}_2)$, but $(u, v) \notin \phi(\vec{\boldsymbol{G}}_1)$ and although $\Delta^-(\vec{\boldsymbol{G}}_2) \leq 2$, two functional symbols in $\sigma_{\mathcal{C}}(1)$ are not enough to retain property 3.

The general idea behind the above example is that in order to have property 3, we cannot "re-use" functions used in $\vec{\mathbf{G}}_i$ to encode edges that appeared in $\vec{\mathbf{G}}_{i+1}$.

8.3 From structures to graphs

In this section we use the definition of bounded expansion from Theorem 5.

Recall the definition of Adjacency($\mathbf{D}$) from Section 2.4. In particular, nodes of Adjacency($\mathbf{D}$) are divided into two sets: D and T. Note that Adjacency($\mathbf{D}$) is a bipartite graph (neither any two nodes from D nor any two nodes from T are ever connected) and the maximal in-degree of a node from T is bounded by the maximal arity of a relation in $\mathbf{D}$. We call nodes from D *real nodes* and nodes from T *tuple nodes*.

The *Gaifman graph* of a relational structure $\mathbf{D}$, denoted by Gaifman($\mathbf{D}$), is defined as follows: the set of vertices of Gaifman($\mathbf{D}$) is D and there is an edge (a, b) in Gaifman($\mathbf{D}$) iff there exists a relation R_i and a tuple $t \in R_i$ such that both a and b occur in t.

In the literature, a class $\mathcal{C}$ of relational structures is said to have bounded expansion if the class $\mathcal{C}$' of Gaifman graphs of structures from $\mathcal{C}$ has bounded expansion. Our definition is more liberal (possibly equivalent).

Let $\mathbf{D}$ be a relational structure over signature σ with universe V, let R be a relation from σ of arity r and let $t \in R$ be a tuple of R in $\mathbf{D}$. The *effective arity of t* is the number of different elements in t.

LEMMA 7. *Let $\mathcal{C}$ be class of relational structures and let $\mathcal{C}$' be the underlying class of Gaifman graphs of structures from $\mathcal{C}$. If $\mathcal{C}$' has bounded expansion, then there exists a constant k such that for any structure $\boldsymbol{D} \in \mathcal{C}$ and for any tuple $t \in \boldsymbol{D}$ the effective arity of t is less than k.*

PROOF. Fix class $\mathcal{C}$ of structures and let $\mathcal{C}$' be the class of Gaifman graphs of structures from $\mathcal{C}$. Let f be the function from Theorem 5 witnessing the fact that $\mathcal{C}$' has bounded expansion.

Set $k = 2f(0)$. Let $\mathbf{D} \in \mathcal{C}$ and t be an arbitrary tuple from $\mathbf{D}$ with effective arity s. Let $A = \{a_1, \ldots, a_s\}$ be the set of different elements in t. By the definition of Gaifman($\mathbf{D}$) vertices from A are pairwise connected. Consider the 0-minor $\mathbf{H}$ of Gaifman($\mathbf{D}$) induced by A. We have that $\frac{|\mathbf{H}|_{\text{EDGE}}}{|\mathbf{H}|_{\text{VERT}}} = \frac{|A| \cdot (|A|-1)}{2|A|} = \frac{s-1}{2}$. By the definition $\nabla_0(\text{Gaifman}(\mathbf{D})) \geq \frac{|\mathbf{H}|_{\text{EDGE}}}{|\mathbf{H}|_{\text{VERT}}} \geq \frac{s-1}{2}$. On the other hand the definition of bounded expansion from Theorem 5 gives $f(0) \geq \nabla_0(\text{Gaifman}(\mathbf{D}))$ and we have $k > s$ as desired. □

PROPOSITION 3. *Let $\mathcal{C}$ be a class of structures such that the class $\mathcal{C}$' of Gaifman graphs of structures from $\mathcal{C}$ has bounded expansion. Then the class $\mathcal{C}$" of adjacency graphs of structures from $\mathcal{C}$ also has bounded expansion.*

It is a consequence of the following lemma.

LEMMA 8. *Let $\mathcal{C}$ be a class of structures such that the class $\mathcal{C}$' of Gaifman graphs of structures from $\mathcal{C}$ has bounded expansion. There exists a constant k such that for any structure $\boldsymbol{D} \in \mathcal{C}$ and for any natural number r we have that $\nabla_r(\mathit{Adjacency}(\boldsymbol{A})) \leq \nabla_r(\mathit{Gaifman}(\boldsymbol{A})) + k$.*

PROOF. Fix class $\mathcal{C}$ of structures such that the class $\mathcal{C}$' of Gaifman graphs of structures from $\mathcal{C}$ has bounded expansion.

Let k be the constant given by Lemma 7.

Let $\mathbf{D} \in \mathcal{C}$ and let r be a natural number and $\mathbf{H}$ be a r-minor of Adjacency($\mathbf{D}$). From $\mathbf{H}$ we construct a graph $\mathbf{H}'$ which is a r-minor of Gaifman($\mathbf{D}$) and such that:

$$\frac{|\mathbf{H}|_{\text{EDGE}}}{|\mathbf{H}|_{\text{VERT}}} \leq \frac{|\mathbf{H}'|_{\text{EDGE}}}{|\mathbf{H}'|_{\text{VERT}}} + k.$$

This immediately yields the result.

Recall from Section 2.4 that Adjacency($\mathbf{D}$) is a bipartite graph that contains *tuple* nodes and *real* nodes and such that neither any two tuple nodes nor any two real nodes are connected. By the definition of constant k from Lemma 7, each tuple node has up to k neighbors in Adjacency($\mathbf{D}$).

Consider a node v of $\mathbf{H}$. By construction, v is derived from a r_v-ball S_v of Adjacency($\mathbf{D}$), where $1 \leq r_v \leq r$.

If S_v contains no real nodes, then it simply is a single tuple node. As each tuple node has up to k neighbors in Adjacency($\mathbf{D}$), then if S_v contains no real nodes, v has at most k neighbors in $\mathbf{H}$. Let X be the set of all such nodes v in $\mathbf{H}$.

Otherwise, let S'_v be the set of real nodes of S_v. By definition S'_v is not empty and it is easy to verify that it forms a $\frac{r_v}{2}$-ball in Gaifman($\mathbf{D}$): for every $u \in S'_v$ the longest path from v to u in S_v is $v = u_1, t_{(1,2)}, u_2, t_{(2,3)}, \ldots, t_{(\frac{r_v}{2}-1,\frac{r_v}{2})}, u_{\frac{r_v}{2}} = u$, where each $t_{(i,i+1)}$ is a tuple node. By the definition of Gaifman($\mathbf{D}$) we have that u_i is connected to u_{i+1} (which is witnessed by $t_{(i,i+1)}$), which yields that $v = u_1, u_2, \ldots, u_{\frac{r_v}{2}} = u$ is a path in S'_v. Let $\mathbf{H}'$ be the r-minor of Gaifman($\mathbf{D}$) constructed from the elements S'_v, $v \notin X$.

By construction we have : $|\mathbf{H}'|_{\text{VERT}} + |X| = |\mathbf{H}|_{\text{VERT}}$.

Consider now an edge (u, v) in $\mathbf{H}$ where both u and v are not in X. This means that there is an edge (a, b) in Adjacency($\mathbf{A}$) with $a \in S_u$ and $b \in S_v$. As Adjacency($\mathbf{A}$) is bipartite, this means that a is a real node and b a tuple node (or vice versa). Wlog assume that a is the real node. As v is not in X, S_v contains a real node b' adjacent to b. Hence b witnesses that (a, b') is an edge in Gaifman($\mathbf{D}$) and so (u, v) is an edge in $\mathbf{H}'$. As we have seen that there are at most $k \cdot |X|$ edges (u, v) in $\mathbf{H}$ where either u or v belongs to X, we get: $|\mathbf{H}|_{\text{EDGE}} \leq |\mathbf{H}'|_{\text{EDGE}} + k|X|$.

Summing up we get:

$$\frac{|\mathbf{H}|_{\text{EDGE}}}{|\mathbf{H}|_{\text{VERT}}} \leq \frac{|\mathbf{H}'|_{\text{EDGE}}+k\cdot|X|}{|\mathbf{H}'|_{\text{VERT}}+|X|} \leq \frac{|\mathbf{H}'|_{\text{EDGE}}}{|\mathbf{H}'|_{\text{VERT}}+|X|} + \frac{k\cdot|X|}{|\mathbf{H}'|_{\text{VERT}}+|X|} \leq \frac{|\mathbf{H}'|_{\text{EDGE}}}{|\mathbf{H}'|_{\text{VERT}}} + k.$$

as desired. □

8.4 Model checking

We now give the details of the skipped part of the proof of Proposition 2, namely the case when $\Delta^=$ is empty.

In this case we construct a set WITNESS which does not depend on v. It is constructed as in the previous case and verifies: for all tuples $\bar{v}$ of $\vec{\mathbf{G}}''$, if $\vec{\mathbf{G}}'' \models \psi(\bar{v}u)$ for some node u, then there is a node $u' \in$ WITNESS such that $\vec{\mathbf{G}}'' \models \psi(\bar{v}u')$. Moreover, $|\text{WITNESS}| \leq \gamma_p$.

Recoloring of $\vec{\mathbf{G}}''$.

Based on WITNESS we recolor $\vec{\mathbf{G}}''$ as follows. Let $\gamma_p = (\beta_p + 1)^{\beta_p+1}$. We order WITNESS and we can now speak of the i^{th} witness.

For each $i \leq \gamma_p$ we introduce a new unary predicate P_i and for each $v \in \vec{\mathbf{G}}''$ we set $P_i(v)$ if WITNESS contains at least i elements.

For each $i \leq \gamma_p$ and each $h \in \sigma_{\mathcal{C}}(q)$ we introduce a new unary predicate $P_{i,h}$ and for each $v \in \vec{\mathbf{G}}''$ we set $P_{i,h}(v)$ if the i^{th} witness is a element u with $h(u) = v$.

For each $i \leq \gamma_p$, $h \in \sigma_{\mathcal{C}}(q)$ we introduce a new unary predicate Q_i and for each $v \in \vec{\mathbf{G}}''$ we set $Q_i(v)$ if the i^{th} witness is v.

We denote by $\vec{\mathbf{G}}'$ the resulting graph and notice that it can be computed in linear time from $\vec{\mathbf{G}}$.

Finally, note that if y is the i^{th} witness, the equality $f_j(y) = h(x_k)$ is equivalent over $\vec{\mathbf{G}}'$ to $P_{i,f_j}(h(x_k))$ and the equality $y = h(x_k)$ is equivalent over $\vec{\mathbf{G}}'$ to $Q_i(h(x_k))$.

The desired formula ϕ is computed as for the previous case when $\Delta^=$ was not empty.

8.5 Enumeration

We now present the omitted proofs from Section 4, namely the proofs of Claims 1, 2 and 3.

CLAIM 1 *Let $\text{NEXT}_{\vec{S}}(u)$ be a shortcut pointer of size not greater than β_q. Then there exists $\text{NEXT}_{\vec{S}'}(u) \in \text{SC}_L$ such that $\text{NEXT}_{\vec{S}}(u) = \text{NEXT}_{\vec{S}'}(u)$. Moreover, such $\text{NEXT}_{\vec{S}'}(u)$ can be found in constant time.*

PROOF. If $\text{NEXT}_{\vec{S}}(u) \in \text{SC}_L$, then we have nothing to prove. Assume then that $\text{NEXT}_{\vec{S}}(u) \notin \text{SC}_L$. Let $\text{NEXT}_{\vec{S}'}(u) \in \text{SC}_L$ be a maximal in terms of size shortcut pointer of u such that $\text{NEXT}_{\vec{S}'}(u) \preceq \text{NEXT}_{\vec{S}}(u)$ (recall that this means that for $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ we have $S'_i \subseteq S_i$). Such a shortcut pointer always exists as $\text{next}_{\emptyset}(u) \preceq \text{NEXT}_{\vec{S}}(u)$ and $\text{next}_{\emptyset}(u) \in \text{SC}_L$. Note that the size of $\text{NEXT}_{\vec{S}'}(u)$ is strictly smaller than the size of $\text{NEXT}_{\vec{S}}(u)$, so it is strictly smaller than β_q. Clearly, $\text{NEXT}_{\vec{S}'}(u)$ can be found in constant time. We claim that $\text{NEXT}_{\vec{S}}(u) = \text{NEXT}_{\vec{S}'}(u)$.

Let $v = \text{NEXT}_{\vec{S}'}(u)$. We know that $v \leq \text{NEXT}_{\vec{S}}(u)$. Assume now that there would exists $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ such that $u_i = f_i(v) \in S_j$. Then $u_i \notin S'_i$ and as the size of $\text{NEXT}_{\vec{S}'}(u)$ is smaller than β_q, we have that $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u) \in \text{SC}_L$. But $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$ has size strictly greater than $\text{NEXT}_{\vec{S}'}(u)$ and $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u) \preceq \text{NEXT}_{\vec{S}}(u)$, which contradicts the maximality of $\text{NEXT}_{\vec{S}'}(u)$. This means that such an i does not exist and concludes the fact that $\text{NEXT}_{\vec{S}}(u) = \text{NEXT}_{\vec{S}'}(u)$.

□

CLAIM 2 *There exists a constant $\zeta(q,k)$ such that for every node u we have $|\text{SC}_L(u)| \leq \zeta(q,k)$.*

PROOF. Fix u. Note that there is exactly 1 shortcut pointer of u of size 0 ($\text{next}_\emptyset(u)$) and $\alpha_{\mathcal{C}}(q)$ shortcut pointers of u of size 1. By the definition of SC_L, any shortcut pointer $\text{NEXT}_{\vec{S}}(u)$ can be an origin of up to $\alpha_{\mathcal{C}}(q)$ shortcut pointers of the form $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$, where $u_i = f_i(\text{NEXT}_{\vec{S}}(u))$ and the size of $\text{NEXT}_{\vec{S}[S_i+=\{u_i\}]}(u)$ is either the same as the size of $\text{NEXT}_{\vec{S}}(u)$ (if $u_i \in S_i$) or greater by 1. This way we see that $\text{SC}_L(u)$ contains up to $\alpha_{\mathcal{C}}(q)^2$ shortcut pointers of size 2 and, in general, up to $\alpha_{\mathcal{C}}(q)^s$ shortcut pointers of size s. As the maximal size of a computed shortcut pointer is bounded by β_q, we have $|\text{SC}_L(u)| \leq \sum_{0 \leq i \leq \beta_q} \alpha_{\mathcal{C}}(q)^i$. Both $\alpha_{\mathcal{C}}(q)$ and β_q depend only on q and k, which concludes the proof. □

CLAIM 3 *SC_L can be computed in time linear in $|L|$.*

PROOF. In linear time we set $\text{next}_\emptyset(u) = u$ for $u \in L$.

We first show how to compute shortcut pointers of size 1 of each node $u \in L$. We do it in an inductive manner, starting from the last node of L and moving backwards. Recall that these shortcut pointers are of the form $\text{NEXT}_{f_i,\{f_i(u)\}}(u)$. If u is the last node on L, then all these values are NULL. We now assume that u is not last on L and that for all $v > u$ all the shortcut pointers of v of size 1 were computed. We show how to compute shortcut pointers of u of size 1.

For each $1 \leq i \leq \alpha_{\mathcal{C}}(q)$ we compute $\text{NEXT}_{f_i,\{f_i(u)\}}(u)$. Let v be the node successor of u in L. If $f_i(u) \neq f_i(v)$, then $\text{NEXT}_{f_i,\{f_i(u)\}}(u) = v$. If $f_i(u) = f_i(v)$, then $\text{NEXT}_{f_i,\{f_i(u)\}}(u) = \text{NEXT}_{f_i,\{f_i(\text{next}(v))\}}(\text{next}(v))$ and the later shortcut pointer has already been computed.

Clearly all the shortcut pointers of size 1 are computed in time linear in the size of L.

We now turn to the computation of arbitrary $\text{NEXT}_{\vec{S}}(u) \in \text{SC}_L$ for $u \in L$. We again do it in an inductive manner starting from the last node on L and move backwards. If u is the last node on L then we are already done as all the shortcut pointers of u of size 1 are NULL and by definition there are no shortcut pointers of u of greater sizes in SC_L. We now assume that u is not last on L and that for all $v > u$ set $\text{SC}_L(v)$ is computed. We show how to compute $\text{SC}_L(u)$.

Consider now $\text{NEXT}_{\vec{S}}(u)$. If $\forall i \;\; f_i(u) \notin S_i$ then we are done, as $\text{NEXT}_{\vec{S}}(u) = u$. Otherwise $\exists i$ such that $f_i(u) \in S_i$. Let $v = \text{NEXT}_{f_i,\{f_i(u)\}}(u)$. Clearly $v \leq \text{NEXT}_{\vec{S}}(u)$ and $\text{NEXT}_{\vec{S}}(u) = \text{NEXT}_{\vec{S}}(v)$. We can conclude this case $\text{NEXT}_{\vec{S}}(v) = \text{NEXT}_{\vec{S}'}(v)$, where $\text{NEXT}_{\vec{S}'}(v) \in \text{SC}_L(v)$ is the shortcut pointer of v from the application of Claim 1 to $\text{NEXT}_{\vec{S}}(v)$. Claim 1 assures that we can find $\text{NEXT}_{\vec{S}'}(v)$ in constant time and thus $\text{NEXT}_{\vec{S}}(u)$ is computed in constant time. As Claim 2 shows that we only need to consider constantly many shortcut pointers for each u, the whole process takes time $O(|L|)$. □

8.6 Counting

CLAIM 4. . *There exists a query ψ^+_{NF} such that: its size depends only on the size of ψ^+, ψ^+_{NF} is in the normal form given by* (1)*, it contains an inequality conjunct $h(y) \neq g_1(x_i)$ (where h might possibly be identity) iff ψ^+ also contains such conjunct and $\psi^+_{\text{NF}}(\vec{\boldsymbol{G}}'') = \psi^+(\vec{\boldsymbol{G}}'')$. Moreover, ψ^+_{NF} can be constructed in time linear in the size of ψ^+.*

PROOF. The proof is a simple case analysis of the content of $\Delta^=$ of ψ.

If its empty, then ψ^+_{NF} is already in the desired form.

If it contains an atom of the form $y = h_2(x_j)$, then equality $g(y) = f(x_i)$ is equivalent to $g(h_2(x_j)) = f(x_i)$ and we are done.

If it contains an atom of the form $h_3(y) = h_2(x_j)$ and g is identity, then $h_3(y) = h_2(x_j)$ is equivalent to $h_3(f(x_i)) = h_2(x_j)$. If g is not identity, then $\tau(y)$ ensures us that either $g(y)$ determines $h_3(y)$ or vice versa. If we have $h_4(g(y)) = h_3(y)$, then $h_3(y) = h_2(x_j)$ is equivalent to $h_4(f(x_i)) = h_2(x_j)$. The other case is symmetric.

The fact that ψ^+_{NF} does not contain any additional inequalities, that it can be computed in time linear in the size of ψ^+ and that $\psi^+_{\text{NF}}(\vec{\mathbf{G}}'') = \psi^+(\vec{\mathbf{G}}'')$ follows from the above construction. □

LEMMA 6 *Let $\mathcal{C}$ be class of graphs with bounded expansion and let $\phi(\bar{x})$ be a first-order formula with exactly k free variables.*
For $1 \leq i \leq k$ let $\#_i : V \to \mathbb{N}$ be functions such that for each v the value of $\#_i(v)$ can be computed in constant time. Then for all $\vec{\boldsymbol{G}} \in \mathcal{C}$ we can compute $|\phi(\vec{\boldsymbol{G}})|_\#$ in time $O(\|\vec{\boldsymbol{G}}\|)$.

PROOF. We now give the omitted details from the proof of Lemma 6, that is the remaining cases of the analysis of the content of $\Delta^=$.

Assume now that $\Delta^=$ consists of an atom $g(y) = f(x_1)$. Let $\psi'(y)$ be the formula $\exists \bar{x} \psi(\bar{x}y)$ and $\psi''(\bar{x})$ the formula $\exists y \psi(\bar{x}y)$. We first compute set $\psi'(\vec{\mathbf{G}}'')$ in linear time using Corollary 2. We now define a function $\#'_k : V \to \mathbb{N}$ as:

$$\#'_k(u) := \sum_{\substack{\{v \in \psi'(\vec{\mathbf{G}}'') \\ g(v)=u\}}} \#_k(v).$$

Note that this function can be easily computed in linear time by going through all nodes v and adding $\#_k(v)$ to $\#'_k(g(v))$.

Finally we set:

$$\#'_1(u) := \#_1(u)\#'_k(f(u))$$
$$\#'_i(u) := \#_i(u) \qquad \text{for } 2 \leq i \leq k-1.$$

Let $u_1, u_2 \in \psi'(\vec{\mathbf{G}}'')$ be such that $g(u_1) = g(u_2)$. Because $\Delta^{\neq}$ is empty, observe that $\vec{\mathbf{G}}'' \models \forall \bar{x}(\psi(\bar{x}u_1) \leftrightarrow \psi(\bar{x}u_2))$. Based on this observation we now group the solutions to ψ according to their last $k-1$ values and get:

$$\begin{aligned}
|\psi(\vec{\mathbf{G}}'')|_{\#} &= \sum_{(\bar{u}v)\in\psi(\vec{\mathbf{G}}'')} \left(\#_k(v) \prod_{1\leq i\leq k-1} \#_i(u_i) \right) \\
&= \sum_{\bar{u}\in\psi''(\vec{\mathbf{G}}'')} \sum_{\substack{\{v\in\psi'(\vec{\mathbf{G}}'') \\ g(v)=f(u_1)\}}} \left(\#_k(v) \prod_{1\leq i\leq k-1} \#_i(u_i) \right) \\
&= \sum_{\bar{u}\in\psi''(\vec{\mathbf{G}}'')} \left(\sum_{\substack{\{v\in\psi'(\vec{\mathbf{G}}'') \\ g(v)=f(u_1)\}}} \#_k(v) \right) \prod_{1\leq i\leq k-1} \#_i(u_i) \\
&= \sum_{\bar{u}\in\psi''(\vec{\mathbf{G}}'')} \left(\#'_k(f(u_1)) \prod_{1\leq i\leq k-1} \#_i(u_i) \right) \\
&= \sum_{\bar{u}\in\psi''(\vec{\mathbf{G}}'')} \left(\#_1(u_1)\#'_k(f(u_1)) \prod_{2\leq i\leq k-1} \#'_i(u_i) \right) \\
&= \sum_{\bar{u}\in\psi''(\vec{\mathbf{G}}'')} \prod_{1\leq i\leq k-1} \#'_i(u_i) \\
&= |\psi''(\vec{\mathbf{G}}'')|_{\#'}
\end{aligned}$$

By induction on the number of free variables, as $\#'_i(u)$ can be computed in constant time for each i and u, we can compute $|\psi''(\vec{\mathbf{G}}'')|_{\#'}$ and we are done with this case.

The remaining case when $\Delta^{=}$ is empty is handled similarly to the previous one. We then have

$$\psi(\bar{x}y) = \psi_1(\bar{x}) \wedge \tau(y).$$

After setting

$$\#'_1(u) := \#_2(u) \cdot \sum_{v\in\tau(\vec{\mathbf{G}}'')} \#_1(v)$$
$$\#'_i(u) := \#_{i+1}(u) \qquad \text{for } 2 \leq i \leq k-1$$

we see that

$$|\psi(\vec{\mathbf{G}}'')|_{\#} = |\psi_1(\vec{\mathbf{G}}'')|_{\#'}$$

and we conclude again by induction on the number of free variables. $\square$