Answering FO+MOD Queries under Updates on Bounded Degree Databases

CHRISTOPH BERKHOLZ, JENS KEPPELER, and NICOLE SCHWEIKARDT,
Humboldt-Universität zu Berlin, Germany

We investigate the query evaluation problem for fixed queries over fully dynamic databases, where tuples can be inserted or deleted. The task is to design a dynamic algorithm that immediately reports the new result of a fixed query after every database update.

We consider queries in first-order logic (FO) and its extension with modulo-counting quantifiers (FO+MOD) and show that they can be efficiently evaluated under updates, provided that the dynamic database does not exceed a certain degree bound.

In particular, we construct a data structure that allows us to answer a Boolean FO+MOD query and to compute the size of the result of a non-Boolean query within constant time after every database update. Furthermore, after every database update, we can update the data structure in constant time such that afterwards we are able to test within constant time for a given tuple whether or not it belongs to the query result, to enumerate all tuples in the new query result, and to enumerate the difference between the old and the new query result with constant delay between the output tuples. The preprocessing time needed to build the data structure is linear in the size of the database.

Our results extend earlier work on the evaluation of first-order queries on static databases of bounded degree and rely on an effective Hanf normal form for FO+MOD recently obtained by Heimberg, Kuske, and Schweikardt (LICS 2016).

CCS Concepts: • **Theory of computation** → **Logic and databases**; *Database query processing and optimization (theory)*; *Finite Model Theory*;

Additional Key Words and Phrases: Dynamic databases, query enumeration, counting problem, first-order logic with modulo-counting quantifiers, Hanf locality

ACM Reference format:
Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2018. Answering FO+MOD Queries under Updates on Bounded Degree Databases. *ACM Trans. Database Syst.* 43, 2, Article 7 (August 2018), 32 pages.
https://doi.org/10.1145/3232056

1 INTRODUCTION

Query evaluation is a fundamental task in databases, and a vast amount of literature is devoted to the complexity of this problem. In this article, we study query evaluation on relational databases in the "dynamic setting," where the database may be updated by inserting or deleting tuples. In

This is the extended version of the conference contribution [3]. Funded by the German Research Foundation DFG under grant SCHW 837/5-1.

Authors' addresses: C. Berkholz, J. Keppeler, and N. Schweikardt, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099, Berlin, Germany; email: {berkholz, keppelej, schweikn}@informatik.hu-berlin.de.

https://doi.org/10.1145/3232056

this setting, an evaluation algorithm receives a query φ and an initial database D and starts with a preprocessing phase that computes a suitable data structure to represent the result of evaluating φ on D. After every database update, the data structure is updated so that it represents the result of evaluating φ on the updated database. The data structure shall be designed in such a way that it quickly provides the query result, preferably in constant time (i.e., independent of the database size). We focus on the following flavours of query evaluation:

- *Testing:* Decide whether a given tuple $\overline{a}$ is contained in $\varphi(D)$.
- *Counting:* Compute $|\varphi(D)|$ (i.e., the number of tuples that belong to $\varphi(D)$).
- *Enumeration:* Enumerate $\varphi(D)$ with a bounded delay between the output tuples.

Here, as usual, $\varphi(D)$ denotes the k-ary relation obtained by evaluating a k-ary query φ on a relational database D. For *Boolean* queries, all three tasks boil down to

- *Answering:* Decide if $\varphi(D) \neq \emptyset$.

Compared to the *dynamic descriptive complexity* framework introduced by Patnaik and Immerman [19], which focuses on the *expressive power* of first-order logic on dynamic databases and has led to a rich body of literature (see Reference [21] for a survey), we are interested in the *computational complexity* of query evaluation. The query language studied in this article is FO+MOD, the extension of first-order logic FO with modulo-counting quantifiers of the form $\exists^{i \bmod m} x\, \psi$, expressing that the number of witnesses x that satisfy ψ is congruent to i modulo m. FO+MOD can be viewed as a subclass of SQL that properly extends the relational algebra.

Following Reference [2], we say that a query evaluation algorithm is efficient if the update time is either constant or at most polylogarithmic ($\log^c n$) in the size of the database. As a consequence, efficient query evaluation in the dynamic setting is only conceivable if the static problem (i.e., the setting without database updates) can be solved for Boolean queries in linear or pseudo-linear ($n^{1+\varepsilon}$) time. Since this is not always possible, we provide a short overview on known results about first-order query evaluation on static databases and then proceed by discussing our results in the dynamic setting.

First-order Query Evaluation on Static Databases. The problem of deciding whether a given database D satisfies an FO-sentence φ is AW[$*$]-complete (parameterised by $\|\varphi\|$), and it is therefore generally believed that the evaluation problem cannot be solved in time $f(\|\varphi\|)\|D\|^c$ for any computable f and constant c (here, $\|\varphi\|$ and $\|D\|$ denote the size of the query and the database, respectively). For this reason, a long line of research focused on increasing classes of sparse instances ranging from databases of *bounded degree* [22] (where every domain element occurs only in a constant number of tuples in the database) to classes that are *nowhere dense* [10]. In particular, Boolean first-order queries can be evaluated on classes of databases of bounded degree in linear time $f(\|\varphi\|)\|D\|$, where the constant factor $f(\|\varphi\|)$ is threefold exponential in $\|\varphi\|$ [8, 22]; and Frick and Grohe [8] showed that the threefold exponential blow-up in terms of the query size is unavoidable assuming FPT $\neq$ AW[$*$].

Durand and Grandjean [6] and Kazana and Segoufin [13] considered the task of enumerating the result of a k-ary first-order query on bounded degree databases and showed that after a linear time preprocessing phase the query result can be enumerated with constant delay. This result was later extended to classes of databases of bounded expansion [14]. Kazana and Segoufin [14] also showed that counting the number of result tuples of a k-ary first-order query on databases of bounded expansion (and hence also on databases of bounded degree) can be done in time $f(\|\varphi\|)\|D\|$. Segoufin and Vigny [23] proved an analogous result for classes of locally bounded expansion and pseudo-linear time $f(\|\varphi\|)\|D\|^{1+\varepsilon}$, and they also presented an algorithm for enumerating the query result with constant delay after pseudo-linear time preprocessing. These results were recently generalised to all nowhere dense classes of databases by Grohe and Schweikardt [11] and Schweikardt, Segoufin, and Vigny [20]. Durand, Schweikardt, and Segoufin [7] obtained analogous results for classes of databases of low degree (i.e., degree at most $\|D\|^{o(1)}$).

Our Contribution. We extend the known linear time algorithms for first-order logic on classes of databases of bounded degree to the more expressive query language FO+MOD. Moreover, and more importantly, we lift the tractability to the dynamic setting and show that the result of FO and FO+MOD-queries can be maintained with constant update time. In particular, we obtain the following results. Let φ be a k-ary FO+MOD-query and d a degree bound on the databases under consideration.[1] Given an initial database D, we construct in linear time $f(\|\varphi\|, d)\|D\|$ a data structure that can be updated in constant time $f(\|\varphi\|, d)$ when a tuple is inserted into or deleted from a relation of D. After each update the data structure allows to

- immediately answer φ on D if φ is a Boolean query (Theorem 4.1),
- test for a given tuple $\overline{a} = (a_1, \ldots, a_k)$ whether $\overline{a} \in \varphi(D)$ in time $O(k^2)$ (Theorem 6.1),
- immediately output the number of result tuples $|\varphi(D)|$ in time $O(1)$ (Theorem 8.1),
- enumerate all tuples $(a_1, \ldots, a_k) \in \varphi(D)$ with $O(k^2)$ delay (Theorem 9.2), and
- enumerate all tuples $(a_1, \ldots, a_k)$ in $\varphi(D_{new}) \setminus \varphi(D_{old})$ and all tuples in $\varphi(D_{old}) \setminus \varphi(D_{new})$ with $O(k^2)$ delay (Theorem 11.1), where D_{old} and D_{new} denote the database before and after performing the update operation, respectively.

For fixed d, the parameter function $f(\|\varphi\|, d)$ is threefold exponential in terms of the query size, which is (by Frick and Grohe [8]) optimal assuming FPT $\neq$ AW[$*$]. We stress that while all these different types of evaluation turn out to be tractable on bounded degree databases, they are in general not equivalent. To the contrary, it has been shown in References [2, 4] that on unrestricted databases there are queries where the different tasks lead to different complexities.

Outline. Our dynamic query evaluation algorithm crucially relies on the locality of FO+MOD and in particular on an effective Hanf normal form for FO+MOD on databases of bounded degree recently obtained by Heimberg, Kuske, and Schweikardt [12]. After some basic definitions in Section 2, we briefly state their result in Section 3 and obtain a dynamic algorithm for Boolean FO+MOD-queries in Section 4. After some preparations for non-Boolean queries in Section 5, we present the algorithm for testing in Section 6. In Section 7, we reduce the task of counting and enumerating FO+MOD-queries in the dynamic setting to the problem of counting and enumerating independent sets in graphs of bounded degree. We use this reduction to provide efficient dynamic counting and enumeration algorithms in Sections 8 and 9, respectively. In Section 10, we generalise this to be able to efficiently enumerate particular subsets of the query result, and we use this in Section 11 to obtain an efficient dynamic algorithm for enumerating the difference between the old and the new query result. We conclude in Section 12.

2 PRELIMINARIES

We write $\mathbb{N}$ for the set of non-negative integers and let $\mathbb{N}_{\geqslant 1} := \mathbb{N} \setminus \{0\}$ and $[n] := \{1, \ldots, n\}$ for all $n \in \mathbb{N}_{\geqslant 1}$. By 2^M we denote the power set of a set M. For a partial function f, we write dom(f) and codom(f) for the domain and the codomain of f, respectively.

Databases. We fix a countably infinite set **dom**, the *domain* of potential database entries. Elements in **dom** are called *constants*. A *schema* is a finite set σ of relation symbols, where each $R \in \sigma$ is equipped with a fixed *arity* $\operatorname{ar}(R) \in \mathbb{N}_{\geqslant 1}$. Let us fix a schema $\sigma = \{R_1, \ldots, R_{|\sigma|}\}$. A *database* D of

[1]Both φ and d are assumed to be fixed, i.e., in contrast to the database they do not change. Moreover, although we make the dependence on $\|\varphi\|$ and d very explicit, we treat them as constants when elaborating on asymptotic complexity.

schema σ (σ-db, for short) is of the form $D = (R_1^D, \ldots, R_{|\sigma|}^D)$, where each R_i^D is a finite subset of $\mathbf{dom}^{\mathrm{ar}(R_i)}$. The *active domain* $\mathrm{adom}(D)$ of D is the smallest subset A of $\mathbf{dom}$ such that $R_i^D \subseteq A^{ar(R_i)}$ for each R_i in σ.

The *Gaifman graph* of a σ-db D is the undirected simple graph $G^D = (V, E)$ with vertex set $V := \mathrm{adom}(D)$, where there is an edge between vertices u and v whenever $u \neq v$ and there are $R \in \sigma$ and $(a_1, \ldots, a_{\mathrm{ar}(R)}) \in R^D$ such that $u, v \in \{a_1, \ldots, a_{\mathrm{ar}(R)}\}$. A σ-db D is called *connected* if its Gaifman graph G^D is connected; the *connected components* of D are the connected components of G^D. The *degree* of a database D is the degree of its Gaifman graph G^D, i.e., the maximum number of neighbours of a node of G^D.

Throughout this article, we fix a number $d \in \mathbb{N}$ and restrict attention to *d-bounded* databases, i.e., to databases of degree at most d.

Updates. We allow update of a given database of schema σ by inserting or deleting tuples as follows (note that both types of commands may change the database's active domain and the database's degree). A *deletion* command is of the form delete $R(a_1, \ldots, a_r)$ for $R \in \sigma$, $r = \mathrm{ar}(R)$, and $a_1, \ldots, a_r \in \mathbf{dom}$. When applied to a σ-db D, it results in the updated σ-db D' with $R^{D'} = R^D \setminus \{(a_1, \ldots, a_r)\}$ and $S^{D'} = S^D$ for all $S \in \sigma \setminus \{R\}$.

An *insertion* command is of the form insert $R(a_1, \ldots, a_r)$ for $R \in \sigma$, $r = \mathrm{ar}(R)$, and $a_1, \ldots, a_r \in \mathbf{dom}$. When applied to a σ-db D in the unrestricted setting, it results in the updated σ-db D' with $R^{D'} = R^D \cup \{(a_1, \ldots, a_r)\}$ and $S^{D'} = S^D$ for all $S \in \sigma \setminus \{R\}$. In this article, we restrict attention to databases of degree at most d. Therefore, when applying an insertion command to a σ-db D of degree $\leqslant d$, the command is carried out only if the resulting database D' still has degree $\leqslant d$; otherwise, D remains unchanged and instead of carrying out the insertion command, an error message is returned.

Queries. We fix a countably infinite set $\mathbf{var}$ of *variables.* We consider the extension FO+MOD of first-order logic FO with modulo-counting quantifiers. For a fixed schema σ, the set FO+MOD[σ] is built from atomic formulas of the form $x_1 = x_2$ and $R(x_1, \ldots, x_{\mathrm{ar}(R)})$, for $R \in \sigma$ and variables $x_1, x_2, \ldots, x_{\mathrm{ar}(R)} \in \mathbf{var}$, and is closed under Boolean connectives $\neg$, $\wedge$, existential first-order quantifiers $\exists x$, and modulo-counting quantifiers $\exists^{i \bmod m} x$, for a variable $x \in \mathbf{var}$ and integers $i, m \in \mathbb{N}$ with $m \geqslant 2$ and $i < m$. The intuitive meaning of a formula of the form $\exists^{i \bmod m} x\, \psi$ is that the number of witnesses x that satisfy ψ is congruent i modulo m. Note that FO+MOD is strictly more expressive than first-order logic without counting quantifiers, since it can express that the number of elements in a unary relation is even, which is not possible to express in FO (cf., e.g., Reference [16]). As usual, $\forall x$, $\vee$, $\rightarrow$, $\leftrightarrow$ will be used as abbreviations when constructing formulas. It will be convenient to add the quantifiers $\exists^{\geqslant m} x$, for $m \in \mathbb{N}_{\geqslant 1}$; a formula of the form $\exists^{\geqslant m} x\, \psi$ expresses that the number of witnesses x that satisfy ψ is $\geqslant m$. Though these quantifiers allow more succinct definitions, we will treat them as syntactic sugar, since they do not increase the expressive power of FO+MOD.

The *quantifier rank* $\mathrm{qr}(\varphi)$ of a FO+MOD-formula φ is the maximum nesting depth of quantifiers that occur in φ. By $\mathrm{free}(\varphi)$, we denote the set of all *free variables* of φ, i.e., all variables x that have at least one occurrence in φ that is not within a quantifier of the form $\exists x$, $\exists^{\geqslant m} x$, or $\exists^{i \bmod m} x$. A *sentence* is a formula φ with $\mathrm{free}(\varphi) = \emptyset$.

An *assignment* for φ is a partial mapping α from $\mathbf{var}$ to $\mathbf{dom}$, where $\mathrm{free}(\varphi) \subseteq \mathrm{dom}(\alpha)$. We write $(D, \alpha) \models \varphi$ to indicate that φ is satisfied when evaluated in D with respect to *active domain semantics* while interpreting every free occurrence of a variable x with the constant $\alpha(x)$. Recall from [1] that "active domain semantics" means that quantifiers are evaluated with respect to the database's active domain. In particular, $(D, \alpha) \models \exists x\, \psi$ iff there exists an $a \in \mathrm{adom}(D)$ such that $(D, \alpha\frac{a}{x}) \models \psi$, where $\alpha\frac{a}{x}$ is the assignment α' with $\alpha'(x) = a$ and $\alpha'(y) = \alpha(y)$ for all $y \in \text{dom}(\alpha) \setminus \{x\}$. Accordingly, $(D, \alpha) \models \exists^{\geqslant m} x\, \psi$ iff $|\{\, a \in \text{adom}(D) \,:\, (D, \alpha\frac{a}{x}) \models \psi \,\}| \;\geqslant\; m$, and $(D, \alpha) \models \exists^{i \bmod m} x\, \psi$ iff $|\{\, a \in \text{adom}(D) \,:\, (D, \alpha\frac{a}{x}) \models \psi \,\}| \;\equiv\; i \bmod m$.

A *k-ary* FO+MOD *query of schema* σ is of the form $\varphi(x_1, \ldots, x_k)$, where $k \in \mathbb{N}$, $\varphi \in$ FO+MOD$[\sigma]$, and $\text{free}(\varphi) \subseteq \{x_1, \ldots, x_k\}$. We will often assume that the tuple $(x_1, \ldots, x_k)$ is clear from the context and simply write φ instead of $\varphi(x_1, \ldots, x_k)$ and $(D, (a_1, \ldots, a_k)) \models \varphi$ instead of $(D, \frac{a_1, \ldots, a_k}{x_1, \ldots, x_k}) \models \varphi$, where $\frac{a_1, \ldots, a_k}{x_1, \ldots, x_k}$ denotes the assignment α with $\alpha(x_i) = a_i$ for all $i \in [k]$. When evaluated in a σ-db D, the k-ary query $\varphi(x_1, \ldots, x_k)$ yields the k-ary relation

$$\varphi(D) \quad := \quad \Big\{ (a_1, \ldots, a_k) \,\in\, \text{adom}(D)^k \;:\; \Big(D, \tfrac{a_1, \ldots, a_k}{x_1, \ldots, x_k}\Big) \;\models\; \varphi \Big\}\,.$$

Boolean queries are k-ary queries with $k = 0$. As usual, for Boolean queries we will write $\varphi(D) =$ no instead of $\varphi(D) = \emptyset$, and $\varphi(D) =$ yes instead of $\varphi(D) \neq \emptyset$; and we write $D \models \varphi$ to indicate that $(D, \alpha) \models \varphi$ for any assignment α.

Sizes and Cardinalities. The *size* $\|\sigma\|$ of a schema σ is the sum of the arities of its relation symbols. The size $\|\varphi\|$ of an FO+MOD query φ of schema σ is the length of φ when viewed as a word over the alphabet $\sigma \cup \mathbf{var} \cup \mathbb{N} \cup \{\, =, \wedge, \neg, \exists, {}^{\text{mod}}, {}^{\geqslant}, (,)\,\} \cup \{, \}$. For a k-ary query $\varphi(x_1, \ldots, x_k)$ and a σ-db D, the *cardinality of the query result* is the number $|\varphi(D)|$ of tuples in $\varphi(D)$. The *cardinality* $|D|$ of a σ-db D is defined as the number of tuples stored in D, i.e., $|D| := \sum_{R \in \sigma} |R^D|$. The *size* $\|D\|$ of D is defined as $\|\sigma\| + |\text{adom}(D)| + \sum_{R \in \sigma} \text{ar}(R) \cdot |R^D|$ and corresponds to the size of a reasonable encoding of D. Throughout the article, we let $f(\varphi, d)$ stand for a function of the form

$$f(\varphi, d) \quad = \quad 2^{d^{2^{O(\|\varphi\|)}}}. \tag{1}$$

Dynamic Algorithms for Query Evaluation. We adopt the framework for dynamic algorithms for query evaluation of [2]; the next paragraphs are taken almost verbatim from [2]. Following [5], we use Random Access Machines (RAMs) with $O(\log n)$ word-size and a uniform cost measure to analyse our algorithms. We will assume that the RAM's memory is initialised to 0. In particular, if an algorithm uses an array, we will assume that all array entries are initialised to 0, and this initialisation comes at no cost (in real-world computers this can be achieved by using the *lazy array initialisation technique*, cf., e.g., Reference [18]). A further assumption is that for every fixed dimension $k \in \mathbb{N}_{\geqslant 1}$ we have available an unbounded number of k-ary arrays A such that for given $(n_1, \ldots, n_k) \in \mathbb{N}^k$ the entry $\mathsf{A}[n_1, \ldots, n_k]$ at position $(n_1, \ldots, n_k)$ can be accessed in constant time.[2] For our purposes, it will be convenient to assume that $\mathbf{dom} = \mathbb{N}_{\geqslant 1}$.

Our algorithms will take as input a k-ary FO+MOD-query $\varphi(x_1, \ldots, x_k)$, a parameter d, and an initial σ-db D_0 of degree $\leqslant d$. For all query evaluation problems considered in this article, we aim at routines **preprocess** and **update** that achieve the following.

On input of $\varphi(x_1, \ldots, x_k)$ and D_0, **preprocess** builds a data structure D that represents D_0 (and that is designed in such a way that it supports the evaluation of φ on D_0). On input of a command update $R(a_1, \ldots, a_r)$ (with update $\in$ {insert, delete}), calling **update** modifies the data structure D such that it represents the updated database D. The *preprocessing time* $t_{\text{preprocess}}$ is the time used for performing **preprocess**; the *update time* t_{update} is the time used for performing an **update**. In this article, t_{update} will be independent of the size of the current database D. By **init**, we denote the particular case of the routine **preprocess** on input of a query $\varphi(x_1, \ldots, x_k)$ and the *empty* database $D_\emptyset$, where $R^{D_\emptyset} = \emptyset$ for all $R \in \sigma$. The *initialisation time* t_{init} is the time used for performing **init**. In all dynamic algorithms presented in this article, the **preprocess** routine for input of $\varphi(x_1, \ldots, x_k)$

[2]While this can be accomplished easily in the RAM-model, for an implementation on real-world computers one would probably have to resort to replacing our use of arrays by using suitably designed hash functions.

and D_0 will carry out the **init** routine for $\varphi(x_1, \ldots, x_k)$ and then perform a sequence of $|D_0|$ update operations to insert all the tuples of D_0 into the data structure. Consequently, $t_{\text{preprocess}} = t_{\text{init}} + |D_0| \cdot t_{\text{update}}$.

In the following, D will always denote the database that is currently represented by the data structure $\mathtt{D}$.

To solve the *enumeration problem under updates*, apart from the routines **preprocess** and **update**, we aim at a routine **enumerate** such that calling **enumerate** invokes an enumeration of all tuples (without repetition) that belong to the query result $\varphi(D)$. The *delay* t_{delay} is the maximum time used during a call of **enumerate**

- until the output of the first tuple (or the end-of-enumeration message EOE, if $\varphi(D) = \emptyset$),
- between the output of two consecutive tuples, and
- between the output of the last tuple and the end-of-enumeration message EOE.

To *test* if a given tuple belongs to the query result, instead of **enumerate** we aim at a routine **test** which on input of a tuple $\overline{a} \in \mathbf{dom}^k$ checks whether $\overline{a} \in \varphi(D)$. The *testing time* t_{test} is the time used for performing a **test**. To solve the *counting problem under updates*, instead of **enumerate** or **test** we aim at a routine **count** that outputs the cardinality $|\varphi(D)|$ of the query result. The *counting time* t_{count} is the time used for performing a **count**. To *answer* a *Boolean* query under updates, instead of **enumerate**, **test**, or **count** we aim at a routine **answer** that produces the answer yes or no of φ on D. The *answer time* t_{answer} is the time used for performing **answer**. Whenever speaking of a *dynamic algorithm*, we mean an algorithm that has routines **preprocess** and **update** and, depending on the problem at hand, at least one of the routines **answer**, **test**, **count**, and **enumerate**.

Throughout the article, we often adopt the view of *data complexity* and suppress factors that may depend on the query φ or the degree bound d but not on the database D. For example, "linear preprocessing time" means $t_{\text{preprocess}} \leqslant g(\varphi, d) \cdot \|D\|$ and "constant update time" means $t_{\text{update}} \leqslant g(\varphi, d)$, for a function g with codomain $\mathbb{N}$. When writing $poly(n)$ we mean $n^{O(1)}$.

3 HANF NORMAL FORM FOR FO+MOD

Our algorithms for evaluating FO+MOD queries rely on a decomposition of FO+MOD queries into *Hanf normal form*. To describe this normal form, we need some more notation.

Two formulas φ and ψ of schema σ are called *d-equivalent* (in symbols: $\varphi \equiv_d \psi$) if for all σ-dbs D *of degree* $\leqslant d$ and all assignments α for φ and ψ in D we have $(D, \alpha) \models \varphi \iff (D, \alpha) \models \psi$.

For a σ-db D and a set $A \subseteq \text{adom}(D)$, we write $D[A]$ to denote the restriction of D to the domain A, i.e., $R^{D[A]} = \{\overline{a} \in R^D : \overline{a} \in A^{\text{ar}(R)}\}$, for all $R \in \sigma$. For two σ-dbs D and D', an *isomorphism* $\pi : D \to D'$ is a bijection from $\text{adom}(D)$ to $\text{adom}(D')$ with $(b_1, \ldots, b_r) \in R^D \iff (\pi(b_1), \ldots, \pi(b_r)) \in R^{D'}$ for all $R \in \sigma$, for $r := \text{ar}(R)$, and for all $b_1, \ldots, b_r \in \text{adom}(D)$. For two k-tuples $\overline{a} = (a_1, \ldots, a_k)$ and $\overline{a}' = (a'_1, \ldots, a'_k)$ of elements in $\text{adom}(D)$ and $\text{adom}(D')$, respectively, we write $(D, \overline{a}) \cong (D', \overline{a}')$ to indicate that there is an isomorphism π from D to D' that maps a_i to a'_i for all $i \in [k]$.

The *distance* $dist^D(a, b)$ between two elements $a, b \in \text{adom}(D)$ is the minimal length (i.e., the number of edges) of a path from a to b in D's Gaifman graph G^D (if no such path exists, we let $dist^D(a, b) = \infty$; note that $dist^D(a, a) = 0$). For $r \geqslant 0$ and $a \in \text{adom}(D)$, the *r-ball* around a in D is the set

$$N_r^D(a) := \{b \in \text{adom}(D) : dist^D(a, b) \leqslant r\}.$$

For a σ-db D and a tuple $\overline{a} = (a_1, \ldots, a_k)$, we let $N_r^D(\overline{a}) := \bigcup_{i \in [k]} N_r^D(a_i)$. The *r-neighbourhood* around $\overline{a}$ in D is defined as the σ-db $\mathcal{N}_r^D(\overline{a}) := D[N_r^D(\overline{a})]$. For $r \geqslant 0$ and $k \geqslant 1$, a *type* τ (*over* σ) *with* k *centres and radius* r (for short: *r-type with k centres*) is of the form $(T, \overline{t})$, where T is a

σ-db, $\overline{t} \in \text{adom}(T)^k$, and $\text{adom}(T) = N_r^T(\overline{t})$. The elements in $\overline{t}$ are called the *centres* of τ. For a tuple $\overline{a} \in \text{adom}(D)^k$, the *$r$-type of $\overline{a}$ in D* is defined as the r-type with k centres $(\mathcal{N}_r^D(\overline{a}), \overline{a})$.

For a given r-type with k centres $\tau = (T, \overline{t})$ it is straightforward to construct a first-order formula $\text{sph}_\tau(\overline{x})$ (depending on r and τ) with k free variables $\overline{x} = (x_1, \ldots, x_k)$ that expresses that the r-type of $\overline{x}$ is isomorphic to τ, i.e., for every σ-db D and all $\overline{a} = (a_1, \ldots, a_k) \in \text{adom}(D)^k$ we have

$$(D, \overline{a}) \models \text{sph}_\tau(\overline{x}) \iff \left(\mathcal{N}_r^D(\overline{a}), \overline{a}\right) \cong (T, \overline{t}).$$

The formula $\text{sph}_\tau(\overline{x})$ is called a *sphere-formula* (over σ and $\overline{x}$); the numbers r and k are called *locality radius* and *arity*, respectively, of the sphere-formula.

A *Hanf-sentence* (over σ) is a sentence of the form $\exists^{\geqslant m} x \ \text{sph}_\tau(x)$ or $\exists^{i \bmod m} x \ \text{sph}_\tau(x)$, where τ is an r-type (over σ) with 1 centre, for some $r \geqslant 0$. The number r is called *locality radius* of the Hanf-sentence. A formula in *Hanf normal form* (over σ) is a Boolean combination[3] of sphere-formulas and Hanf-sentences (over σ). The *locality radius* of a formula ψ in Hanf normal form is the maximum of the locality radii of the Hanf-sentences and the sphere-formulas that occur in ψ. The formula is *d-bounded* if all types τ that occur in sphere-formulas or Hanf-sentences of ψ are d-bounded, i.e., T is of degree $\leqslant d$, where $\tau = (T, \overline{t})$. Our query evaluation algorithms for FO+MOD rely on the following result by Heimberg, Kuske, and Schweikardt [12].

Theorem 3.1 ([12]). *There is an algorithm that receives as input a degree bound $d \in \mathbb{N}$ and an FO+MOD$[\sigma]$-formula φ and constructs a d-equivalent formula ψ in Hanf normal form (over σ) with the same free variables as φ. For any $d \geqslant 2$, the formula ψ is d-bounded and has locality radius $\leqslant 4^{qr(\varphi)}$, and the algorithm's runtime is $2^{d^{2^{O(\|\varphi\|+\|\sigma\|)}}}$.*

The first step of all our query evaluation algorithms is to use Theorem 3.1 to transform a given query $\varphi(\overline{x})$ into a d-equivalent query $\psi(\overline{x})$ in Hanf normal form. The following lemma summarises standard facts that we will apply at several places throughout the article to evaluate the sphere-formulas that occur in ψ.

Lemma 3.1. *Let $d \geqslant 2$ and let D be a σ-db of degree $\leqslant d$. Let $r \geqslant 0$, $k \geqslant 1$, and $\overline{a} = (a_1, \ldots, a_k) \in \text{adom}(D)^k$.*

(a) $|N_r^D(\overline{a})| \leqslant k\sum_{i=0}^{r} d^i \leqslant kd^{r+1}$.
(b) Given D and $\overline{a}$, the r-neighbourhood $\mathcal{N}_r^D(\overline{a})$ can be computed in time $(kd^{r+1})^{O(\|\sigma\|)}$.
(c) $\mathcal{N}_r^D(a_1, a_2)$ is connected if and only if $dist^D(a_1, a_2) \leqslant 2r{+}1$.
(d) If $\mathcal{N}_r^D(\overline{a})$ is connected, then $N_r^D(\overline{a}) \subseteq N_{r+(k-1)(2r+1)}^D(a_i)$, for all $i \in [k]$.
(e) Let D' be a σ-db of degree $\leqslant d$ and let $\overline{b} = (b_1, \ldots, b_k) \in adom(D')^k$. It can be tested in time $(kd^{r+1})^{O(\|\sigma\|+kd^{r+1})} \leqslant 2^{O(\|\sigma\|k^2d^{2r+2})}$ whether $(\mathcal{N}_r^D(\overline{a}), \overline{a}) \cong (\mathcal{N}_r^{D'}(\overline{b}), \overline{b})$.

Proof. Parts (a)–(d) are straightforward. Concerning Part (e), a brute-force approach is to loop through all mappings from $N_r^D(\overline{a})$ to $N_r^{D'}(\overline{b})$ that map a_i to b_i for every $i \in [k]$ and check whether this mapping is an isomorphism. Each such check can be accomplished in time $n^{O(\|\sigma\|)}$ for $n := kd^{r+1}$, and the number of mappings that have to be checked is $\leqslant n^n$. Thus, the isomorphism test is accomplished in time $n^{O(n+\|\sigma\|)} = (kd^{r+1})^{O(\|\sigma\|+kd^{r+1})}$. □

The time bound stated in part (e) of Lemma 3.1 is obtained by a brute-force approach. When using Luks' polynomial time isomorphism test for bounded degree graphs [17], the time bound of Lemma 3.1(e) can be improved to $(kd^{r+1})^{poly(d\|\sigma\|)}$. However, the asymptotic overall runtime of

[3]Throughout this article, whenever we speak of *Boolean combinations* we mean *finite* Boolean combinations.

our algorithms for evaluating FO+MOD-queries will not improve when using Luks's algorithm instead of the brute-force isomorphism test of Lemma 3.1(e).

4 ANSWERING BOOLEAN FO+MOD QUERIES UNDER UPDATES

In Reference [8], Frick and Grohe showed that in the static setting (i.e., without database updates), Boolean FO-queries φ can be answered on databases D of degree $\leqslant d$ in time $f(\varphi, d) \cdot \|D\|$. Our first main theorem extends their result to FO+MOD-queries and the dynamic setting.

Theorem 4.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a Boolean FO+MOD$[\sigma]$-query φ, and a σ-db D_0 of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_{\text{update}} = f(\varphi, d)$ and allows to return the query result $\varphi(D)$ with answer time $t_{\text{answer}} = O(1)$.*

If φ is a d-bounded Hanf-sentence of locality radius r, then $f(\varphi, d)$ improves to $f(\varphi, d) = 2^{O(\|\sigma\| d^{2r+2})}$, and the initialisation time is $t_{\text{init}} = O(\|\varphi\|)$.

The proof will be an easy consequence of Theorem 3.1 and the following lemma.

Lemma 4.1. *There is a dynamic algorithm that receives a schema σ, a number $s \in \mathbb{N}_{\geqslant 1}$, a list $(r_j)_{j \in [s]}$ of non-negative integers, a list $(\rho_j)_{j \in [s]}$ where each ρ_j is an r_j-type with 1 centre (over σ), a degree bound $d \geqslant 2$, and a σ-db D of degree $\leqslant d$. The algorithm computes within $t_{\text{init}} = O(s)$ initialisation time a data structure that can be updated in time $t_{\text{update}} = \sum_{j=1}^{s} (d^{r_j+1})^{O(\|\sigma\| + d^{r_j+1})}$. On input of a number $j \in [s]$ the algorithm returns within time $O(1)$ the number $|\{a \in adom(D) : (\mathcal{N}_{r_j}^{D}(a), a) \cong \rho_j\}|$.*

In particular, the update time t_{update} is at most $s \cdot 2^{O(\|\sigma\| \cdot d^{2r+2})}$, for $r := \max_{j \in [s]} r_j$.

Proof. For each $j \in [s]$ our data structure will store the number $\mathsf{A}[j]$ of all elements $a \in \text{adom}(D)$ whose r_j-type is isomorphic to ρ_j, i.e., $(\mathcal{N}_{r_j}^{D}(a), a) \cong \rho_j$. The initialisation for the empty database $D_\emptyset$ lets $\mathsf{A}[j] = 0$ for all $j \in [s]$.

To update our data structure on a command update $R(a_1, \ldots, a_k)$, for $k = \text{ar}(R)$ and update $\in \{\text{insert}, \text{delete}\}$, we proceed as follows. The idea is to remove from the data structure the information on all the database elements whose r_j-neighbourhood (for some $j \in [s]$) is affected by the update and then to recompute the information concerning all these elements on the updated database.

Let D_{old} be the database before the update is received and let D_{new} be the database after the update has been performed. We consider each $j \in [s]$. All elements whose r_j-neighbourhood might have changed belong to the set $U_j := N_{r_j}^{D'}(\overline{a})$, where $D' := D_{new}$ if the update command is insert $R(\overline{a})$, and $D' := D_{old}$ if the update command is delete $R(\overline{a})$.

To remove the old information from $\mathsf{A}[j]$, we compute for each $a \in U_j$ the neighbourhood $T_a := \mathcal{N}_{r_j}^{D_{old}}(a)$, check whether $(T_a, a) \cong \rho_j$, and, if so, decrement the value $\mathsf{A}[j]$.

To recompute the new information for $\mathsf{A}[j]$, we compute for all $a \in U_j$ the neighbourhood $T'_a := \mathcal{N}_{r_j}^{D_{new}}(a)$, check whether $(T'_a, a) \cong \rho_j$, and, if so, increment the value $\mathsf{A}[j]$.

Using Lemma 3.1, we obtain for each $j \in [s]$ that $|U_j| \leqslant k d^{r_j+1}$. For each $a \in U_j$, the neighbourhoods T_a and T'_a can be computed in time $(d^{r_j+1})^{O(\|\sigma\|)}$, and testing for isomorphism with ρ_j can be done in time $(d^{r_j+1})^{O(\|\sigma\| + d^{r_j+1})}$. Thus, the update of $\mathsf{A}[j]$ is done in time $k \cdot (d^{r_j+1})^{O(\|\sigma\| + d^{r_j+1})}$. Recall that $k = \text{ar}(R) \leqslant \|\sigma\|$. Hence, the entire update time is $t_{\text{update}} = \sum_{j=1}^{s} (d^{r_j+1})^{O(\|\sigma\| + d^{r_j+1})}$. Finally, note that

$$(d^{r+1})^{O(\|\sigma\| + d^{r+1})} \leqslant 2^{d^{r+1} \cdot O(\|\sigma\| + d^{r+1})} \leqslant 2^{O(\|\sigma\| \cdot d^{2r+2})}.$$

This completes the proof of Lemma 4.1. □

Proof of Theorem 4.1. W.l.o.g. we assume that all the symbols of σ occur in φ (otherwise, we remove from σ all symbols that do not occur in φ). In the preprocessing routine, we first use Theorem 3.1 to transform φ into a d-equivalent sentence ψ in Hanf normal form; this takes time $f(\varphi, d)$. The sentence ψ is a Boolean combination of d-bounded Hanf-sentences (over σ) of locality radius at most $r := 4^{\mathrm{qr}(\varphi)}$. Let $\rho_1, \ldots, \rho_s$ be the list of all types that occur in ψ. Thus, every Hanf-sentence in ψ is of the form $\exists^{\geqslant k} x\ \mathsf{sph}_{\rho_j}(x)$ or $\exists^{i \bmod m} x\ \mathsf{sph}_{\rho_j}(x)$ for some $j \in [s]$ and $k, i, m \in \mathbb{N}$ with $k \geqslant 1$, $m \geqslant 2$, and $i < m$. For each $j \in [s]$ let r_j be the radius of $\mathsf{sph}_{\rho_j}(x)$. Thus, ρ_j is an r_j-type with 1 centre (over σ).

We use the dynamic data structure provided by Lemma 4.1, and in addition, we also store a Boolean value Ans, where $\mathsf{Ans} = \varphi(D)$ is the answer of the Boolean query φ on the current database D. This way, the query can be answered in time $O(1)$ by simply outputting Ans.

The initialisation for the empty database $D_\emptyset$ computes Ans as follows. Every Hanf-sentence of the form $\exists^{\geqslant k} x\ \mathsf{sph}_{\rho_j}(x)$ in ψ is replaced by the Boolean constant false. Every Hanf-sentence of the form $\exists^{i \bmod m} x\ \mathsf{sph}_{\rho_j}(x)$ is replaced by true if $i = 0$ and by false otherwise. The resulting formula, a Boolean combination of the Boolean constants true and false, then is evaluated, and we let Ans be the obtained result. The entire initialisation takes time at most $t_{\mathrm{init}} = \|\psi\| = f(\varphi, d) = 2^{d^{2^{O(\|\varphi\|)}}}$.

To update our data structure on a command update $R(a_1, \ldots, a_k)$, we first perform the update routine of the data structure provided by Lemma 4.1. Afterwards, we recompute the query answer Ans as follows. Every Hanf-sentence of the form $\exists^{\geqslant k} x\ \mathsf{sph}_{\rho_j}(x)$ in ψ is replaced by the Boolean constant true if $|\{a \in \mathrm{adom}(D) : (\mathcal{N}^D_{r_j}(a), a) \cong \rho_j\}| \geqslant k$, and by the Boolean constant false otherwise. Every Hanf-sentence of the form $\exists^{i \bmod m} x\ \mathsf{sph}_{\rho_j}(x)$ is replaced by true if $|\{a \in \mathrm{adom}(D) : (\mathcal{N}^D_{r_j}(a), a) \cong \rho_j\}| \equiv i \bmod m$, and by false otherwise. The resulting formula, a Boolean combination of the Boolean constants true and false, then is evaluated, and we let Ans be the obtained result. Thus, recomputing Ans takes time $poly(\|\psi\|)$.

Noting that $r_j \leqslant 4^{\mathrm{qr}(\varphi)} \leqslant 2^{O(\|\varphi\|))}$ and $s \leqslant \|\psi\|$, we obtain that the entire update time is

$$t_{\mathrm{update}} \quad \leqslant \quad \sum_{j=1}^{s} (d^{r_j+1})^{O(\|\sigma\|+d^{r_j+1})} + poly(\|\psi\|) \quad \leqslant \quad 2^{d^{2^{O(\|\varphi\|)}}} \quad = \quad f(\varphi, d).$$

This completes the proof of Theorem 4.1. □

In Reference [8], Frick and Grohe obtained a matching lower bound for answering Boolean FO-queries of schema $\sigma = \{E\}$ on databases of degree at most $d := 3$ in the static setting. They used the (reasonable) complexity theoretic assumption $\mathsf{FPT} \neq \mathsf{AW}[*]$ and showed that if this assumption is correct, then there is no algorithm that answers Boolean FO-queries φ on σ-dbs D of degree $\leqslant 3$ in time $2^{2^{2^{o(\|\varphi\|)}}} \cdot poly(\|D\|)$ in the static setting (see Theorem 2 in Reference [8]). As a consequence, the same lower bound holds in the dynamic setting and shows that in Theorem 4.1, the threefold exponential dependency on the query size $\|\varphi\|$ cannot be substantially lowered (unless $\mathsf{FPT} = \mathsf{AW}[*]$):

Corollary 4.2. *Let $\sigma := \{E\}$ and let $d := 3$. If $\mathsf{FPT} \neq \mathsf{AW}[*]$, then there is no dynamic algorithm that receives a Boolean $\mathsf{FO}[\sigma]$-query φ and a σ-db D_0 and computes within $t_{\mathrm{preprocess}} \leqslant f(\varphi) \cdot poly(\|D_0\|)$ preprocessing time a data structure that can be updated in time $t_{\mathrm{update}} \leqslant f(\varphi)$ and allows to return the query result $\varphi(D)$ with answer time $t_{\mathrm{answer}} \leqslant f(\varphi)$, for a function f with $f(\varphi) = 2^{2^{2^{o(\|\varphi\|)}}}$.*

5 TECHNICAL LEMMAS ON TYPES AND SPHERES USEFUL FOR HANDLING NON-BOOLEAN QUERIES

For our algorithms for evaluating non-Boolean queries it will be convenient to work with a fixed list of representatives of d-bounded r-types, provided by the following lemma.

Lemma 5.1. *There is an algorithm that on input of a schema σ, a degree bound $d \geqslant 2$, a radius $r \geqslant 0$, and a number $k \geqslant 1$ computes a list $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$ (for a suitable $\ell \geqslant 1$) of d-bounded r-types with k centres (over σ), such that for every d-bounded r-type τ with k centres (over σ) there is exactly one $i \in [\ell]$ such that $\tau \cong \tau_i$. The algorithm's runtime is $2^{(kd^{r+1})^{O(\|\sigma\|)}}$. Furthermore, on input of a d-bounded r-type τ with k centres (over σ), the particular $i \in [\ell]$ with $\tau \cong \tau_i$ can be computed in time $2^{(kd^{r+1})^{O(\|\sigma\|)}}$.*

Taking into account the statements of Lemma 3.1 (in particular, the time bound provided by Lemma 3.1(e)), the proof of Lemma 5.1 is straightforward. Throughout the remainder of this article, $\mathcal{L}_r^{\sigma,d}(k)$ will always denote the list provided by Lemma 5.1. The following lemma will be useful for evaluating Boolean combinations of sphere-formulas.

Lemma 5.2. *Let σ be a schema, let $r \geqslant 0$, $k \geqslant 1$, $d \geqslant 2$, and let $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$.*

Let $\overline{x} = (x_1, \ldots, x_k)$ be a list of k pairwise distinct variables. For every Boolean combination $\psi(\overline{x})$ of d-bounded sphere-formulas of radius at most r (over σ), there is an $I \subseteq [\ell]$ such that $\psi(\overline{x}) \equiv_d \bigvee_{i \in I} \mathrm{sph}_{\tau_i}(\overline{x})$.

Furthermore, given $\psi(\overline{x})$, the set I can be computed in time poly$(\|\psi\|) \cdot 2^{(kd^{r+1})^{O(\|\sigma\|)}}$.

Proof. As a first step, we consider each sphere-formula ζ that occurs in ψ and replace it by a d-equivalent disjunction of sphere-formulas $\mathrm{sph}_{\tau_j}(\overline{x})$ with τ_j in $\mathcal{L}_r^{\sigma,d}(k)$: If ζ has arity $k' \leqslant k$ and radius $r' \leqslant r$ and is of the form $\mathrm{sph}_\rho(\overline{x}')$ with $\overline{x}' = (x_{\nu_1}, \ldots, x_{\nu_{k'}})$ for $1 \leqslant \nu_1 < \cdots < \nu_{k'} \leqslant k$ and $\rho = (S, \overline{s})$ with $\overline{s} = (s_1, \ldots, s_{k'})$, then we replace ζ by the formula $\zeta' := \bigvee_{j \in J} \mathrm{sph}_{\tau_j}(\overline{x})$, where J consists of all those $j \in [\ell]$, where for $(T, \overline{t}) = \tau_j$ with $\overline{t} = (t_1, \ldots, t_k)$ and for $\overline{t}' := (t_{\nu_1}, \ldots, t_{\nu_{k'}})$ we have $(S, \overline{s}) \cong (T[N_{r'}^T(\overline{t}')], \overline{t}')$. It is straightforward to see that ζ' and ζ are d -equivalent.

Let ψ_1 be the formula obtained from ψ by replacing each ζ by ζ'. By the Lemmas 5.1 and 3.1, ψ_1 can be constructed in time $O(\|\psi\| \cdot 2^{(kd^{r+1})^{O(\|\sigma\|)}})$. Note that ψ_1 is a Boolean combination of formulas $\mathrm{sph}_{\tau_j}(\overline{x})$ for $j \in [\ell]$.

In the second step, we repeatedly use de Morgan's law to push all $\neg$-symbols in ψ_1 directly in front of sphere-formulas. Afterwards, we replace every subformula of the form $\neg\,\mathrm{sph}_{\tau_j}(\overline{x})$ by the d-equivalent formula $\bigvee_{i \in [\ell] \setminus \{j\}} \mathrm{sph}_{\tau_i}(\overline{x})$. Let ψ_2 be the formula obtained from ψ_1 by these transformations. Constructing ψ_2 from ψ_1 takes time at most $O(\|\psi_1\|) \cdot 2^{(kd^{r+1})^{O(\|\sigma\|)}} = O(\|\psi\| \cdot 2^{(kd^{r+1})^{O(\|\sigma\|)}})$.

In the third step, we eliminate all the $\wedge$-symbols in ψ_2. By the definition of the sphere-formulas $\tau_1, \ldots, \tau_\ell$, we have

$$\mathrm{sph}_{\tau_i}(\overline{x}) \wedge \mathrm{sph}_{\tau_{i'}}(\overline{x}) \quad \equiv_d \quad \begin{cases} \mathrm{sph}_{\tau_i}(\overline{x}), & \text{if } i = i' \\ \bot, & \text{if } i \neq i' \end{cases}, \tag{2}$$

where $\bot$ is an unsatisfiable formula. Thus, by the distributive law, we obtain for all $m \geqslant 1$ and all $I_1, \ldots, I_m \subseteq [\ell]$ that

$$\bigwedge_{j \in [m]} \left(\bigvee_{i \in I_j} \mathrm{sph}_{\tau_i}(\overline{x}) \right) \quad \equiv_d \quad \bigvee_{i_1 \in I_1} \cdots \bigvee_{i_m \in I_m} \left(\mathrm{sph}_{\tau_{i_1}}(\overline{x}) \wedge \cdots \wedge \mathrm{sph}_{\tau_{i_m}}(\overline{x}) \right) \quad \equiv_d \quad \bigvee_{i \in I} \mathrm{sph}_{\tau_i}(\overline{x})$$

for $I := I_1 \cap \cdots \cap I_m$. We repeatedly use this equivalence during a bottom-up traversal of the syntax-tree of ψ_2 to eliminate all the $\wedge$-symbols in ψ_2. The resulting formula ψ_3 is obtained in time polynomial in the size of ψ_2. Furthermore, ψ_3 is of the desired form $\bigvee_{i \in I} \mathsf{sph}_{\tau_i}(\overline{x})$ for an $I \subseteq [\ell]$. The overall time for constructing ψ_3 and I is $poly(\|\psi\|) \cdot 2^{(kd^{r+1})^{O(\|\sigma\|)}}$. This completes the proof of Lemma 5.2. □

For evaluating a Boolean combination $\psi(\overline{x})$ of sphere-formulas *and Hanf-sentences* on a given σ-db D, an obvious approach is to first consider every Hanf-sentence χ that occurs in ψ, to check if $D \models \chi$ and to replace every occurrence of χ in ψ with true (respectively, false) if $D \models \chi$ (respectively, $D \not\models \chi$). The resulting formula $\psi'(\overline{x})$ is then transformed into a disjunction $\psi''(\overline{x}) := \bigvee_{i \in I} \mathsf{sph}_{\tau_i}(\overline{x})$ by Lemma 5.2, and the query result $\psi(D) = \psi''(D)$ is obtained as the union of the query results $\mathsf{sph}_{\tau_i}(D)$ for all $i \in I$.

While this works well in the static setting (i.e., without database updates), in the dynamic setting we have to take care of the fact that database updates might change the status of a Hanf-sentence χ in ψ, i.e., an update operation might turn a database D with $D \models \chi$ into a database D' with $D' \not\models \chi$ (and vice versa). Consequently, the formula $\psi''(\overline{x})$ that is equivalent to $\psi(\overline{x})$ on D might be inequivalent to $\psi(\overline{x})$ on D'.

To handle the dynamic setting correctly, at the end of each update step we will use the following lemma, which is an extension of Lemma 5.2 and is proved in a similar way.

Lemma 5.3. *Let σ be a schema. Let $s \geqslant 0$ and let $\chi_1, \ldots, \chi_s$ be arbitrary formulas of schema σ. Let $r \geqslant 0$, $k \geqslant 1$, $d \geqslant 2$, and let $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$. Let $\overline{x} = (x_1, \ldots, x_k)$ be a list of k pairwise distinct variables. For every Boolean combination $\psi(\overline{x})$ of the formulas $\chi_1, \ldots, \chi_s$ and of d-bounded sphere-formulas of radius at most r (over σ), and for every $J \subseteq [s]$ there is a set $I \subseteq [\ell]$ such that*

$$\psi_J(\overline{x}) \quad \equiv_d \quad \bigvee_{i \in I} \mathsf{sph}_{\tau_i}(\overline{x}),$$

where ψ_J is the formula obtained from ψ by replacing every occurrence of a formula χ_j with true *if $j \in J$ and with* false *if $j \notin J$ (for every $j \in [s]$).*

Given ψ and J, the set I can be computed in time $poly(\|\psi\|) \cdot 2^{(kd^{r+1})^{O(\|\sigma\|)}}$.

To evaluate a single sphere-formula $\mathsf{sph}_\tau(\overline{x})$ for a given r-type τ with k centres (over σ), it will be useful to decompose τ into its connected components as follows. Let $\tau = (T, \overline{t})$ with $\overline{t} = (t_1, \ldots, t_k)$. Consider the Gaifman graph G^T of T and let $C_1, \ldots, C_c$ be the vertex sets of the c connected components of G^T. For each connected component C_j of G^T, let $\overline{t}_j$ be the subsequence of $\overline{t}$ consisting of all elements of $\overline{t}$ that belong to C_j, and let k_j be the length of $\overline{t}_j$. Since $(T, \overline{t})$ is an r-type with k centres, we have $T = \mathcal{N}_r^T(\overline{t})$, and thus $c \leqslant k$ and $k_j \geqslant 1$ for all $j \in [c]$. To avoid ambiguity, we make sure that the list $C_1, \ldots, C_c$ is sorted in such a way that for all $j < j'$ we have $i < i'$ for the smallest i with $t_i \in C_j$ and the smallest i' with $t_{i'} \in C_{j'}$.

For each C_j consider the r-type with k_j centres $\rho_j = (T[C_j], \overline{t}_j)$. Let ν_j be the unique integer such that ρ_j is isomorphic to the ν_jth element in the list $\mathcal{L}_r^{\sigma,d}(k_j)$, and let τ_{j,ν_j} be the ν_jth element in this list.

It is straightforward to see that the formula $\mathsf{sph}_\tau(\overline{x})$ is d-equivalent to the formula

$$\text{conn-sph}_\tau(\overline{x}) \quad := \quad \bigwedge_{j \in [c]} \mathsf{sph}_{\tau_{j,\nu_j}}(\overline{x}_j) \ \wedge \ \bigwedge_{j \neq j'} \neg \ \mathsf{dist}^{k_j, k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'}), \tag{3}$$

where $\overline{x}_j$ is the subsequence of $\overline{x}$ obtained from $\overline{x}$ in the same way as $\overline{t}_j$ is obtained from $\overline{t}$, and $\mathsf{dist}^{k_j, k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'})$ is a formula of schema σ that expresses that for some variable y in $\overline{x}_j$ and some variable y' in $\overline{x}_{j'}$ the distance between y and y' is $\leqslant 2r{+}1$. I.e., for $\overline{a} = (a_1, \ldots, a_{k_j})$ and $\overline{b} = (b_1, \ldots, b_{k_{j'}})$ we have $(\overline{a}, \overline{b}) \in \mathrm{dist}^{k_j,k_{j'}}_{\leqslant 2r+1}(D) \iff \mathit{dist}^D(\overline{a}; \overline{b}) \leqslant 2r{+}1$, where

$$\mathit{dist}^D(\overline{a}; \overline{b}) \leqslant 2r{+}1 \text{ means that } \mathit{dist}^D(a_i, b_{i'}) \leqslant 2r{+}1 \text{ for some } i \in [k_j] \text{ and } i' \in [k_{j'}]. \tag{4}$$

Using the Lemmas 3.1 and 5.1, the following lemma is straightforward.

Lemma 5.4. *There is an algorithm that on input of a schema σ, numbers $r \geqslant 0$, $k \geqslant 1$, and $d \geqslant 2$, and an r-type τ with k centres (over σ) computes the formula $\text{conn-sph}_\tau(\overline{x})$, along with the corresponding parameters c and k_j, ν_j, $\overline{x}_j$, τ_{j,ν_j} for all $j \in [c]$.*

The algorithm's runtime is $2^{(kd^{r+1})^{O(\|\sigma\|)}}$.

We define the *signature of τ* w.r.t. r to be the tuple $\mathrm{sgn}_r(\tau)$ built from the parameters c and $(k_j, \nu_j, \{\mu \in [k] : x_\mu \text{ belongs to } \overline{x}_j\})_{j\in[c]}$ obtained from the above lemma. The signature $\mathrm{sgn}^D_r(\overline{a})$ of a tuple $\overline{a}$ in a database D w.r.t. radius r is defined as $\mathrm{sgn}_r(\rho)$ for $\rho := (\mathcal{N}^D_r(\overline{a}), \overline{a})$. Note that $\overline{a} \in \mathrm{sph}_\tau(D) \iff \mathrm{sgn}^D_r(\overline{a}) = \mathrm{sgn}_r(\tau)$.

6 TESTING NON-BOOLEAN FO+MOD QUERIES UNDER UPDATES

This section is devoted to the proof of the following theorem.

Theorem 6.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary $\mathsf{FO{+}MOD}[\sigma]$-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$ and computes within $t_{\mathrm{preprocess}} = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_{\mathrm{update}} = f(\varphi, d)$ and allows to test for any input tuple $\overline{a} \in \mathbf{dom}^k$ whether $\overline{a} \in \varphi(D)$ within testing time $t_{\mathrm{test}} = O(k^2)$.*

For the proof, we use the lemmas provided in Section 5 and the following lemma.

Lemma 6.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, numbers $r \geqslant 0$ and $k \geqslant 1$, an r-type τ with k centres (over σ), and a σ-db D_0 of degree $\leqslant d$ and computes within $t_{\mathrm{preprocess}} = 2^{(kd^{r+1})^{O(\|\sigma\|)}} \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_{\mathrm{update}} = 2^{(kd^{r+1})^{O(\|\sigma\|)}}$ and allows to test for any input tuple $\overline{a} \in \mathbf{dom}^k$ whether $\overline{a} \in \mathrm{sph}_\tau(D)$ within testing time $t_{\mathrm{test}} = O(k^2)$.*

Proof. The preprocessing routine starts by using Lemma 5.4 to compute the formula $\text{conn-sph}_\tau(\overline{x})$, along with the according parameters c and k_j, ν_j, $\overline{x}_j$, τ_{j,ν_j} for each $j \in [c]$. This is done in time $2^{(kd^{r+1})^{O(\|\sigma\|)}}$. We let $\mathrm{sgn}_r(\tau)$ be the signature of τ (defined directly after Lemma 5.4). Recall that $\text{conn-sph}_\tau(\overline{x}) \equiv_d \mathrm{sph}_\tau(\overline{x})$, and recall from Equation (3) the precise definition of the formula $\text{conn-sph}_\tau(\overline{x})$. Our data structure will store the following information on the database D:

- the set Γ of all tuples $\overline{b} \in \mathrm{adom}(D)^{k'}$, where $k' \leqslant k$ and $\mathcal{N}^D_r(\overline{b})$ *is connected*, and
- for every $j \in [c]$ and every tuple $\overline{b} \in \Gamma$ of arity k_j, the unique number $\nu_{\overline{b}}$ such that $\rho_{\overline{b}} := (\mathcal{N}^D_r(\overline{b}), \overline{b})$ is isomorphic to the $\nu_{\overline{b}}$th element in the list $\mathcal{L}^{\sigma,d}_r(k_j)$.

We want to store this information in such a way that for any given tuple $\overline{b} \in \mathbf{dom}^{k'}$ it can be checked in time $O(k)$ whether $\overline{b} \in \Gamma$. To ensure this, we use a k'-ary array $\Gamma_{k'}$[4] that is initialised to 0 and where during update operations the entry $\Gamma_{k'}[\overline{b}]$ is set to 1 for all $\overline{b} \in \Gamma$ of arity k'. In a similar way we can ensure that for any given $j \in [c]$ and any $\overline{b} \in \Gamma$ of arity k_j, the number $\nu_{\overline{b}}$ can be looked up in time $O(k)$.

[4]This array requires nonlinear but polynomial space

The **test** routine on input of a tuple $\overline{a} = (a_1, \ldots, a_k)$ proceeds as follows.

First, we partition $\overline{a}$ into $\overline{a}_1, \ldots, \overline{a}_{c'}$ (for $c' \leqslant k$) such that $C_j := N_r^D(\overline{a}_j)$ for $j \in [c']$ are the connected components of $\mathcal{N}_r^D(\overline{a})$. As in the definition of the formula conn-sph$_\tau(\overline{x})$, we make sure that this list is sorted in such a way that for all $j < j'$ we have $i < i'$ for the smallest i with $a_i \in C_j$ and the smallest i' with $a_{i'} \in C_{j'}$. All of this can be done in time $O(k^2)$ by first constructing the graph H with vertex set $[k]$ and where there is an edge between vertices i and j iff the tuple (a_i, a_j) belongs to Γ (i.e., $\mathcal{N}_r^D(a_i, a_j)$ is connected) and then computing the connected components of H.

Afterwards, for each $j \in [c']$, we use time $O(k)$ to look up the number $\nu_{\overline{a}_j}$. We then let $\text{sgn}_r^D(\overline{a})$ be the tuple built from c' and $(|\overline{a}_j|, \nu_{\overline{a}_j}, \{\mu \in [k] : a_\mu \text{ belongs to } \overline{a}_j\})_{j \in [c']}$. It is straightforward to see that $\overline{a} \in \text{conn-sph}_\tau(D)$ iff $\text{sgn}_r^D(\overline{a}) = \text{sgn}_r(\tau)$. Therefore, the **test** routine checks whether $\text{sgn}_r^D(\overline{a}) = \text{sgn}_r(\tau)$ and outputs "yes" if this is the case and "no" otherwise. The entire time used by the **test** routine is $t_{\text{test}} = O(k^2)$.

To finish the proof of Lemma 6.1, we have to give further details on the **preprocess** routine and the **update** routine. The **preprocess** routine initialises Γ as the empty set $\emptyset$ and then performs $|D_0|$ update operations to insert all the tuples of D_0 into the data structure. The **update** routine proceeds as follows.

Let D_{old} be the database before the update is received and let D_{new} be the database after the update has been performed. Let the update command be of the form update $R(a_1, \ldots, a_{\text{ar}(R)})$. We let $r' := r + (\text{ar}(R){-}1)(2r{+}1)$. All elements whose r'-neighbourhood might have changed belong to the set $U := N_{r'}^{D'}(\overline{a})$, where $D' := D_{new}$ if the update command is insert $R(\overline{a})$, and $D' := D_{old}$ if the update command is delete $R(\overline{a})$.

According to Lemma 3.1(d), all tuples $\overline{b}$ that have to be inserted into or deleted from Γ are built from elements in U. To update the information stored in our data structure, we loop through all tuples of arity $\leqslant k$ that are built from elements in U.

Using Lemma 3.1(a), we obtain that $|U| \leqslant \text{ar}(R) \cdot d^{r'+1}$. The number of candidate tuples $\overline{b}$ built from elements in U is at most $(\text{ar}(R) \cdot d^{r'+1})^{k+1}$. Using the Lemmas 3.1 and 5.1, it is not difficult to see that the entire update time is at most $t_{\text{update}} = 2^{(kd^{r+1})^{O(\|\sigma\|)}}$. The initialisation time t_{init} is of the same form, and hence the preprocessing time is as claimed in the lemma. This completes the proof of Lemma 6.1. □

Using Lemma 6.1 and Lemma 5.3, we can show the following.

LEMMA 6.2. *Let σ be a schema and let $d \geqslant 2$ be a degree bound. Let $s \geqslant 0$ and let $\chi_1, \ldots, \chi_s$ be arbitrary sentences of schema σ, and assume we have available for each $j \in [s]$ a dynamic algorithm with initialisation time t'_{init} and update time t'_{update} that allows to check within answer time t'_{answer} whether or not $D \models \chi_j$ for d-bounded σ-dbs D.*

Then, there is a dynamic algorithm for d-bounded σ-dbs that receives as input numbers $r \geqslant 0$ and $k \geqslant 1$, a tuple $\overline{x} = (x_1, \ldots, x_k)$ of pairwise distinct variables, and a Boolean combination $\psi(\overline{x})$ of the sentences $\chi_1, \ldots, \chi_s$ and of d-bounded sphere-formulas of radius at most r (over σ). Within initialisation time

$$t_{\text{init}} = s(t'_{\text{init}} + t'_{\text{answer}}) + poly(\|\psi\|) 2^{(kd^{r+1})^{O(\|\sigma\|)}} \tag{5}$$

the algorithm builds a data structure that can be updated within time

$$t_{\text{update}} = s(t'_{\text{update}} + t'_{\text{answer}}) + poly(\|\psi\|) 2^{(kd^{r+1})O(\|\sigma\|)} \tag{6}$$

and allows us to test for any input tuple $\overline{a} \in adom(D)^k$ whether $\overline{a} \in \psi(D)$ within testing time

$$t_{\text{test}} = O(k^2). \tag{7}$$

Proof. We use Lemma 5.1 to compute the list $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$. For each $i \in [\ell]$, we use the dynamic algorithm provided by Lemma 6.1 for $\tau := \tau_i$. Furthermore, for each $j \in [s]$, we use the dynamic algorithm provided by the lemma's assumption for checking whether or not $D \models \chi_j$. In addition to the components used by these dynamic algorithms, our data structure also stores

- the set $J := \{j \in [s] \,:\, D \models \chi_j\}$,
- the particular set $I \subseteq [\ell]$ provided by Lemma 5.3 for $\psi(\overline{x})$ and J, and
- the set $K = \{\text{sgn}_r(\tau_i) \,:\, i \in I\}$, where for each type τ, $\text{sgn}_r(\tau)$ is the signature of τ defined directly after Lemma 5.4.

The **test** routine on input of a tuple $\overline{a} = (a_1, \ldots, a_k)$ proceeds in the same way as in the proof of Lemma 6.1 to compute in time $O(k^2)$ the signature $\text{sgn}_r^D(\overline{a})$ of the tuple $\overline{a}$. For every $i \in [\ell]$, we have $\overline{a} \in \text{sph}_{\tau_i}(D) \iff \text{sgn}_r^D(\overline{a}) = \text{sgn}_r(\tau_i)$. Thus, $\overline{a} \in \varphi(D) \iff \text{sgn}_r^D(\overline{a}) \in K$. Therefore, the **test** routine checks whether $\text{sgn}_r^D(\overline{a}) \in K$ and outputs "yes" if this is the case and "no" otherwise. To ensure that this test can be done in time $O(k^2)$, we use an array construction for storing K (similar to the one for storing Γ in the proof of Lemma 6.1).

The **update** routine runs the update routines for all the used dynamic data structures. Afterwards, it recomputes J by calling the **answer** routine for χ_j for all $j \in [s]$. Then, it uses Lemma 5.3 to recompute I. The set K is then recomputed by applying Lemma 5.4 for $\tau := \tau_i$ for all $i \in I$. It is straightforward to verify that the initialisation time t_{init}, the update time t_{update}, and the testing time t_{test} are as claimed by the lemma. □

Theorem 6.1 is now obtained by combining Theorem 3.1, Lemma 6.2, and Theorem 4.1.

Proof of Theorem 6.1. For $k = 0$, the theorem immediately follows from Theorem 4.1. Consider the case where $k \geqslant 1$. As in the proof of Theorem 4.1, we assume w.l.o.g. that all the symbols of σ occur in φ. We start the preprocessing routine by using Theorem 3.1 to transform $\varphi(\overline{x})$ into a d-equivalent query $\psi(\overline{x})$ in Hanf normal form; this takes time $2^{d^{2^{O(\|\varphi\|)}}}$. The formula ψ is a Boolean combination of d-bounded Hanf-sentences and sphere-formulas (over σ) of locality radius at most $r := 4^{\text{qr}(\varphi)}$, and each sphere-formula is of arity at most k. Let $\chi_1, \ldots, \chi_s$ be the list of all Hanf-sentences that occur in ψ.

From Theorem 4.1 we have available for each $j \in [s]$ a dynamic algorithm with initialisation time $t'_{\text{init}} = O(\max_{j\in[s]} \|\chi_j\|)$ and update time $t'_{\text{update}} = 2^{O(\|\sigma\| d^{2r+2})}$ that allows us to check within answer time $t'_{\text{answer}} = O(1)$ whether $D \models \chi_j$ for d-bounded σ-dbs D. The proof of Theorem 6.1 therefore immediately follows from Lemma 6.2. □

7 REPRESENTING DATABASES BY COLOURED GRAPHS

To obtain dynamic algorithms for counting and enumerating query results, it will be convenient to work with a representation of databases by coloured graphs that is similar to the representation used in Reference [7]. The main advantage of this representation is that it unveils the combinatorial core of evaluating non-Boolean queries. In Theorem 7.1, we provide a general reduction from evaluating FO+MOD[σ]-queries to finding coloured independent sets in coloured graphs, which allows us to focus on this combinatorial graph problem when presenting our algorithms for counting (Section 8) and enumeration (Section 9).

For defining this representation, let us consider a fixed d-bounded r-type τ with k centres (over a schema σ). Use Lemma 5.4 to compute the formula $\text{conn-sph}_\tau(\overline{x})$ (for $\overline{x} = (x_1, \ldots, x_k)$) and the according parameters c and $k_j, \nu_j, \overline{x}_j, \tau_{j,\nu_j}$, and let $\text{sgn}_r(\tau)$ be the signature of τ. To keep the notation simple, we assume w.l.o.g. that $\overline{x}_1 = x_1, \ldots, x_{k_1}$, $\overline{x}_2 = x_{k_1+1}, \ldots, x_{k_1+k_2}$, and so on.

Recall that $\mathsf{sph}_\tau(\overline{x})$ is d-equivalent to the formula

$$\text{conn-sph}_\tau(\overline{x}) \quad := \quad \bigwedge_{j\in[c]} \mathsf{sph}_{\tau_{j,\nu_j}}(\overline{x}_j) \ \wedge \ \bigwedge_{j\neq j'} \neg\, \mathsf{dist}^{k_j,k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'}).$$

To count or enumerate the results of the formula $\mathsf{sph}_\tau(\overline{x})$, we represent the database D by a *c-coloured graph $\mathcal{G}_D$*. Here, a *c-coloured graph $\mathcal{G}$* is a database of the particular schema

$$\sigma_c \quad := \quad \{E, C_1, \ldots, C_c\},$$

where E is a binary relation symbol and $C_1, \ldots, C_c$ are unary relation symbols. We define $\mathcal{G}_D$ in such a way that the task of counting or enumerating the results of the query $\mathsf{sph}_\tau(\overline{x})$ on the database D can be reduced to counting or enumerating the results of the query

$$\varphi_c(z_1, \ldots, z_c) \quad := \quad \bigwedge_{j\in[c]} C_j(z_j) \ \wedge \ \bigwedge_{j\neq j'} \neg\, E(z_j, z_{j'}) \tag{8}$$

on the c-coloured graph $\mathcal{G}_D$. The vertices of $\mathcal{G}_D$ correspond to k_j-tuples over $\mathrm{adom}(D)$ (for some $k_j \in \{k_1, \ldots, k_c\} \subseteq [k]$) whose r-neighbourhood is connected; a vertex has colour C_j if its associated tuple $\overline{a}$ is in $\mathsf{sph}_{\tau_{j,\nu_j}}(D)$; and an edge between two vertices indicates that $dist^D(\overline{a}; \overline{b}) \leqslant 2r+1$ for their associated tuples $\overline{a}$ and $\overline{b}$. The following theorem allows us to translate a dynamic algorithm for counting or enumerating the results of the query $\varphi_c(z_1, \ldots, z_c)$ on c-coloured graphs into a dynamic algorithm for counting or enumerating the result of an FO+MOD-query $\varphi(\overline{x})$ on D.

Theorem 7.1. *Suppose that for any $d', c \in \mathbb{N}$ the counting problem (the enumeration problem) for $\varphi_c(\overline{z})$ on σ_c-dbs of degree at most d' can be solved by a dynamic algorithm with initialisation time $t_{\mathsf{init}}(c, d')$, update time $t_{\mathsf{update}}(c, d')$, and counting time $t_{\mathsf{count}}(c, d')$ (delay $t_{\mathsf{delay}}(c, d')$). Then for every schema σ and every $d \geqslant 2$ the following holds. The counting problem (the enumeration problem) for k-ary FO+MOD$[\sigma]$-queries $\varphi(\overline{x})$ on σ-dbs of degree at most d can be solved with counting time $O(1)$ (delay $O(\widehat{t}_{\mathsf{delay}} + k)$), initialisation time $\widehat{t}_{\mathsf{init}} \cdot f(\varphi, d)$, and update time*

- *$(\widehat{t}_{\mathsf{update}} + \widehat{t}_{\mathsf{count}}) \cdot f(\varphi, d)$ for the counting problem, and*
- *$\widehat{t}_{\mathsf{update}} \cdot f(\varphi, d)$ for the enumeration problem,*

where $\widehat{t_x} = \max_{c=1}^{k} t_x(c, d^{2^{O(\|\varphi\|)}})$ for $t_x \in \{t_{\mathsf{init}}, t_{\mathsf{update}}, t_{\mathsf{count}}, t_{\mathsf{delay}}\}$.

The proof will be obtained as an easy consequence of Theorem 3.1, Theorem 4.1, and the following lemma.

Lemma 7.1. *Suppose that the counting problem (the enumeration problem) for $\varphi_c(\overline{z})$ on σ_c-dbs of degree at most d' can be solved by a dynamic algorithm with initialisation time $t_{\mathsf{init}}(c, d')$, update time $t_{\mathsf{update}}(c, d')$, and counting time $t_{\mathsf{count}}(c, d')$ (delay $t_{\mathsf{delay}}(c, d')$).*

Then for every schema σ and every $d \geqslant 2$ the following holds. Let $r \geqslant 0$, $k \geqslant 1$, and fix $d' := d^{2k^2(2r+1)}$ and $\widetilde{t_x} := \max_{c=1}^{k} t_x(c, d')$ for $t_x \in \{t_{\mathsf{init}}, t_{\mathsf{update}}, t_{\mathsf{count}}, t_{\mathsf{delay}}\}$.

(1) Let τ be a d-bounded r-type with k centres. The counting problem (the enumeration problem) for $\mathsf{sph}_\tau(\overline{x})$ on σ-dbs of degree at most d can be solved by a dynamic algorithm with counting time $\widetilde{t}_{\mathsf{count}}$ (delay $O(\widetilde{t}_{\mathsf{delay}} + k)$), initialisation time $\widetilde{t}_{\mathsf{init}}$, and update time at most $\widetilde{t}_{\mathsf{update}} d^{O(k^2 r + k\|\sigma\|)} + 2^{O(\|\sigma\| k^2 d^{2r+2})}$.

(2) Let $s \geqslant 0$ and let $\chi_1, \ldots, \chi_s$ be arbitrary sentences of schema σ, and assume we have available for each $j \in [s]$ a dynamic algorithm with initialisation time t'_{init} and update time t'_{update} that

allows to check within answer time t'_{answer} whether or not $D \models \chi_j$ for d-bounded σ-dbs D.

Let $\overline{x} = (x_1, \ldots, x_k)$ be a tuple of pairwise distinct variables, and let $\psi(\overline{x})$ be a Boolean combination of the sentences $\chi_1, \ldots, \chi_s$ and of d-bounded sphere-formulas of radius at most r (over σ).

Then, the counting problem (the enumeration problem) for $\psi(\overline{x})$ on σ-dbs D of degree at most d can be solved by a dynamic algorithm with counting time $O(1)$ (delay $O(\widetilde{t}_{\text{delay}} + k)$), initialisation time $s(t'_{\text{init}} + t'_{\text{answer}}) + 2^{(kd^{r+1})^{O(\|\sigma\|)}}(poly(\|\psi\|) + \widetilde{t}_{\text{init}})$, and update time at most

- *$s(t'_{\text{update}} + t'_{\text{answer}}) + 2^{(k^2d^{2r+2})^{O(\|\sigma\|)}}(poly(\|\psi\|) + \widetilde{t}_{\text{count}} + \widetilde{t}_{\text{update}}d^{O(k^2r+k\|\sigma\|)})$ for the counting problem, and*
- *$s(t'_{\text{update}} + t'_{\text{answer}}) + 2^{(k^2d^{2r+2})^{O(\|\sigma\|)}}(poly(\|\psi\|) + t_{\text{update}}d^{O(k^2r+k\|\sigma\|)})$ for the enumeration problem.*

Proof. We prove part (1) by a reduction from conn-sph$_\tau(\overline{x})$ to φ_c. We use the notation introduced at the beginning of Section 7, and we let $\tau_j := \tau_{j,\nu_j}$ for every $j \in [c]$. For a σ-db D of degree at most d we let $\mathcal{G}_D$ be the σ_c-db with

$$\begin{aligned} C_j^{\mathcal{G}_D} &:= \{\, v_{\overline{a}} \,:\, \overline{a} \in \text{adom}(D)^{k_j} \text{ with } \left(\mathcal{N}_r^D(\overline{a}), \overline{a}\right) \cong \tau_j \,\}, \quad \text{for all } j \in [c], \quad \text{and} \\ E^{\mathcal{G}_D} &:= \{\, (v_{\overline{a}}, v_{\overline{b}}) \in V^2 \,:\, dist^D(\overline{a};\overline{b}) \leqslant 2r{+}1 \,\}, \end{aligned}$$

where $V := \bigcup_{j\in[c]} C_j^{\mathcal{G}_D}$. We will shortly write E and C_j instead of $E^{\mathcal{G}_D}$ and $C_j^{\mathcal{G}_D}$.

Using Lemma 3.1 and the fact that τ_j is connected we obtain that $(v_{\overline{a}}, v_{\overline{b}}) \in E$ iff $\mathcal{N}_r^D(\overline{a}, \overline{b})$ is connected. If $\mathcal{N}_r^D(\overline{a}, \overline{b})$ is connected, then by Lemma 3.1(d) $\overline{b} \in (N^D_{r+(|\overline{a}|+|\overline{b}|-1)(2r+1)}(a_1))^{|\overline{b}|}$. It follows that the degree of $\mathcal{G}_D$ is bounded by $d^{2k^2(2r+1)}$. Furthermore, by the definition of $\mathcal{G}_D$ and φ_c, we get that $(\overline{a}_1, \ldots, \overline{a}_c) \in \text{sph}_\tau(D) \iff (v_{\overline{a}_1}, \ldots, v_{\overline{a}_c}) \in \varphi_c(\mathcal{G}_D)$, for all tuples $\overline{a}_1, \ldots, \overline{a}_c$ where $\overline{a}_j$ has arity k_j for each $j \in [c]$. As a consequence, $|\,\text{sph}_\tau(D)| = |\varphi_c(\mathcal{G}_D)|$, and we can therefore use the **count** routine for φ_c on $\mathcal{G}_D$ to count the number of tuples in sph$_\tau(D)$. Furthermore, by annotating every vertex $v_{\overline{a}}$ with its tuple $\overline{a}$, we can translate every tuple $(v_{\overline{a}_1}, \ldots, v_{\overline{a}_c}) \in \varphi_c(\mathcal{G}_D)$ to $(\overline{a}_1, \ldots, \overline{a}_c)$ in time $O(k)$. Therefore, given an **enumerate** routine for $\varphi_c(\mathcal{G}_D)$ with delay t_{delay} we can produce an enumeration of sph$_\tau(D)$ with delay $O(t_{\text{delay}} + k)$.

It remains to show how to construct and maintain $\mathcal{G}_D$ when the database D is updated. As initialisation for the empty database $D_\emptyset$, we just perform the **init** routine of the dynamic algorithm for $\varphi_c(\overline{z})$ on σ_c-dbs of degree at most d'. The **update** routine of the dynamic algorithm for sph$_\tau(\overline{x})$ on σ-dbs of degree at most d is provided by the following claim.

Claim 7.2. *If D_{new} is obtained from D_{old} by one update step, then $\mathcal{G}_{D_{new}}$ can be obtained from $\mathcal{G}_{D_{old}}$ by $d^{O(k^2r+k\|\sigma\|)}$ update steps and additional computing time $2^{O(\|\sigma\|k^2d^{2r+2})}$.* □

Proof. Let the update command be of the form update $R(a_1, \ldots, a_{\text{ar}(R)})$ with $\overline{a} = (a_1, \ldots, a_{\text{ar}(R)})$. Let $D' \in \{D_{old}, D_{new}\}$ be the database whose relation R contains the tuple $\overline{a}$ (either before deletion or after insertion). Let $r' := r + (k{-}1)(2r{+}1)$ and note that all elements in the active domain whose r'-neighbourhood in the database might have changed belong to the set $U := N_{r'}^{D'}(\overline{a})$.

For every $j \in [c]$ and every tuple $\overline{b}$ of arity at most k of elements in U, we check whether the r-type $(\mathcal{N}_r^{D_{new}}(\overline{b}), \overline{b})$ of $\overline{b}$ is isomorphic to τ_j. Depending on the outcome of this test, we include or exclude $v_{\overline{b}}$ from the relation C_j. Note that it indeed suffices to consider the tuples $\overline{b}$ built from elements in U: The r-type of some tuple $\overline{b}$ is changed by the update command only if $N_r^{D'}(\overline{b})$ contains some element from $\overline{a}$. Furthermore, we only have to consider tuples $\overline{b}$ whose r-neighbourhood

$\mathcal{N}_r^{D'}(\overline{b})$ is connected. Using Lemma 3.1(d), we therefore obtain that each component of $\overline{b}$ belongs to $N_{r'}^{D'}(\overline{a}) = U$.

Afterwards, we update the coloured graph's edge relation E. There is an edge $(v_{\overline{b}}, v_{\overline{b}'}) \in E^{D_{new}}$, if and only if, there are $j, j' \in [c]$ such that $v_{\overline{b}} \in C_j$, $v_{\overline{b}'} \in C_{j'}$ and $dist^{D_{new}}(\overline{b}; \overline{b}') \leqslant 2r{+}1$. Note that if $dist^{D_{new}}(\overline{b}; \overline{b}') \leqslant 2r{+}1$, then there is some component b_i in $\overline{b}$ and some component $b'_{i'}$ in $\overline{b}'$ such that $dist^{D_{new}}(b_i, b'_{i'}) \leqslant 2r{+}1$; and in case that $v_{\overline{b}'} \in C_{j'}$, we know that $\mathcal{N}_r^{D_{new}}(\overline{b}')$ is connected, and hence every component of the tuple $\overline{b}'$ belongs to $N^{D_{new}}_{r+(k-1)(2r+1)}(b'_{i'}) \subseteq N^{D_{new}}_{r+k(2r+1)}(b_i) \subseteq N^{D_{new}}_{r+k(2r+1)}(\overline{b})$. Moreover, for correctly updating the edge relation E, it suffices to consider only those pairs of tuples $\overline{b}$ and $\overline{b}'$ where for at least one of the two tuples, at least one component belongs to $N_r^{D'}(\overline{a})$, and hence all components belong to $N_{r'}^{D'}(\overline{a})$, since these are the tuples where the r-neighbourhood or the condition $dist^{D}(\overline{b}; \overline{b}') \leqslant 2r{+}1$ might be affected by the database update. Overall, the algorithm proceeds as follows: We compute for all tuples $\overline{b}$ of arity at most k in $N_{r'}^{D'}(\overline{a})$, all tuples $\overline{b}'$ of arity at most k in $N^{D'}_{r+k(2r+1)}(\overline{b})$ (later, we will call these tuples *candidate tuples*) and check whether (1) there is a $j \in [c]$ such that $v_{\overline{b}} \in C_j$, (2) there is a $j' \in [c]$ such that $v_{\overline{b}'} \in C_{j'}$, and (3) $dist^{D_{new}}(\overline{b}; \overline{b}') \leqslant 2r{+}1$. If all three checks return the result "yes," then we insert the tuple $(v_{\overline{b}}, v_{\overline{b}'})$ into E; otherwise, we remove it from E.

It remains to analyse the runtime of the described update procedure. By Lemma 3.1, $|U| \leqslant \text{ar}(R)d^{r'+1} \leqslant \|\sigma\| d^{r+(k-1)(2r+1)+1} \leqslant d^{O(kr+\lg\|\sigma\|)} \leqslant d^{O(kr+\|\sigma\|)}$. Furthermore, U can be computed in time $(\text{ar}(R)d^{r'+1})^{O(\|\sigma\|)} \leqslant d^{O(kr\|\sigma\|+\|\sigma\|^2)}$. The number of tuples $\overline{b}$ that we have to consider is at most $\sum_{i=1}^{k} |U|^i \leqslant |U|^{k+1} \leqslant d^{O(k^2r+k\|\sigma\|)}$.

For each such $\overline{b}$ we use Lemma 3.1(e) to check in time $2^{O(\|\sigma\|k^2d^{2r+2})}$ whether the r-type of $\overline{b}$ in D_{new} is isomorphic to τ_j, for some $j \in [c]$. In summary, for updating the sets $C_1, \ldots, C_c$ we use at most $c|U|^{k+1} \leqslant d^{O(k^2r+k\|\sigma\|)}$ calls of the **update** routine of the dynamic algorithm on coloured graphs, and in addition to that we use computation time at most $2^{O(\|\sigma\|k^2d^{2r+2})}$.

To update the edge relation, we compute for each of the $d^{O(k^2r+k\|\sigma\|)}$ tuples $\overline{b}$ a list of all its candidate tuples $\overline{b}'$; using Lemma 3.1 this can be done in time $(\text{ar}(R)d^{r+k(2r+1)+1})^{O(\|\sigma\|)} \leqslant d^{O(kr\|\sigma\|+\|\sigma\|^2)}$. By Lemma 3.1, the number of candidate tuples is $\leqslant d^{O(k^2r+k\|\sigma\|)}$. By Lemma 3.1(e), it takes time $2^{O(\|\sigma\|k^2d^{2r+2})}$ to check if $(\mathcal{N}_r^{D_{new}}(\overline{b}), \overline{b})$ is isomorphic to τ_j and $(\mathcal{N}_r^{D_{new}}(\overline{b}'), \overline{b}')$ is isomorphic to $\tau_{j'}$, for some $j, j' \in [c]$. We can use and maintain an additional array that allows us to check, for any a_i and b_j, in constant time whether $dist^{D}(a_i, b_j) \leqslant 2r{+}1$. Overall, we obtain that also the edge relation E can be updated by at most $d^{O(k^2r+k\|\sigma\|)}$ calls of the **update** routine of the dynamic algorithm on coloured graphs and additional computation time at most $2^{O(\|\sigma\|k^2d^{2r+2})}$.

This completes the proof of Claim 7.2. $\square$

Finally, the **preprocess** routine of the dynamic algorithm for $\text{sph}_\tau(\overline{x})$ proceeds in the obvious way by first calling the **init** routine for $D_\emptyset$ and then performing $|D_0|$ update steps to insert all the tuples of D_0 into the data structure. This completes the proof of part (1) of Lemma 7.1.

We now turn to the proof of part (2) of Lemma 7.1. We use Lemma 5.1 to compute the list $\mathcal{L}_r^{\sigma,d}(k) = \tau_1, \ldots, \tau_\ell$ within time $2^{(kd^{r+1})^{O(\|\sigma\|)}}$. For each $i \in [\ell]$, we use the dynamic algorithm for $\text{sph}_{\tau_i}(\overline{x})$ provided from the lemma's part (1). Furthermore, for each $j \in [s]$, we use the dynamic algorithm for answering whether or not $D \models \chi_j$, provided by the assumption of part (2) of Lemma 7.1. In addition to the components used by these dynamic algorithms, our data structure also stores

- the set $J := \{j \in [s] \,:\, D \models \chi_j\}$ and
- the particular set $I \subseteq [\ell]$ provided by Lemma 5.3 for $\psi(\overline{x})$ and J.

For the case where we want to solve the counting problem, our data structure also stores

- the cardinality $n = |\varphi(D)|$ of the query result.

The **count** routine simply outputs the value n in time $O(1)$.

The **enumerate** routine runs the **enumerate** routine on $\mathrm{sph}_{\tau_i}(D)$ for every $i \in I$. Note that this enumerates, without repetition, all tuples in $\varphi(D)$, because by Lemma 5.3, $\varphi(D)$ is the union of the sets $\mathrm{sph}_{\tau_i}(D)$ for all $i \in I$, and this is a union of pairwise disjoint sets.

The **update** routine runs the update routines for all used dynamic data structures. Afterwards, it recomputes J by calling the **answer** routine for χ_j for all $j \in [s]$. Then, it uses Lemma 5.3 to recompute I. For the case where we want to solve the counting problem, we afterwards recompute the number n by letting $n = \sum_{i \in I} n_i$, where n_i is the result of the **count** routine for τ_i.

By using the statement of part (1) and the assumptions of part (2) of the lemma, it is straightforward to verify that the initialisation time, the update time, and the counting time (the delay) are as claimed by the lemma. □

Theorem 7.1 is now obtained by combining Theorem 3.1, part (2) of Lemma 7.1, and Theorem 4.1.

Proof of Theorem 7.1. For $k = 0$, the result follows immediately from Theorem 4.1. Consider the case where $k \geqslant 1$. W.l.o.g. we assume that all the symbols of σ occur in φ (otherwise, we remove from σ all symbols that do not occur in φ). We start the preprocessing routine by using Theorem 3.1 to transform $\varphi(\overline{x})$ into a d-equivalent query $\psi(\overline{x})$ in Hanf normal form; this takes time $2^{d^{2^{O(\|\varphi\|)}}}$. The formula ψ is a Boolean combination of d-bounded Hanf-sentences and sphere-formulas (over σ) of locality radius at most $r := 4^{\mathrm{qr}(\varphi)}$, and each sphere-formula is of arity at most k. Note that for $d' := d^{2k^2(2r+1)}$ as used in the lemma's part (1), it holds that $d' = d^{2^{O(\|\varphi\|)}}$. Let $\chi_1, \ldots, \chi_s$ be the list of all Hanf-sentences that occur in ψ, and note that $s \leqslant 2^{d^{2^{O(\|\varphi\|)}}}$.

From Theorem 4.1 we have available for each $j \in [s]$ a dynamic algorithm with initialisation time $t'_{\text{init}} = O(\max_{j \in [s]} \|\chi_j\|)$ and update time $t'_{\text{update}} = 2^{O(\|\sigma\| d^{2r+2})}$ that allows to check within answer time $t'_{\text{answer}} = O(1)$ whether $D \models \chi_j$ for d-bounded σ-dbs D.

Applying part (2) of Lemma 7.1, we obtain a dynamic algorithm that solves the counting problem (the enumeration problem) for $\varphi(\overline{x})$ on σ-dbs of degree at most d with counting time $O(1)$ (delay $O(\widetilde{t}_{\text{delay}} + k) = O(\widehat{t}_{\text{delay}} + k)$), initialisation time

$$s(t'_{\text{init}} + t'_{\text{answer}}) + 2^{(kd^{r+1})^{O(\|\sigma\|)}} (poly(\|\psi\|) + \widetilde{t}_{\text{init}}) \quad \leqslant \quad \widehat{t}_{\text{init}} \cdot 2^{d^{2^{O(\|\varphi\|)}}}$$

and update time at most

$$\begin{aligned} & s(t'_{\text{update}} + t'_{\text{answer}}) + 2^{(k^2 d^{2r+2})^{O(\|\sigma\|)}} (poly(\|\psi\|) + \widetilde{t}_{\text{count}} + \widetilde{t}_{\text{update}} d^{O(k^2 r + k\|\sigma\|)}) \\ \leqslant & \left(\widehat{t}_{\text{update}} + \widehat{t}_{\text{count}}\right) \cdot 2^{d^{2^{O(\|\varphi\|)}}} \end{aligned}$$

when dealing with the counting problem, and update time at most

$$s(t'_{\text{update}} + t'_{\text{answer}}) + 2^{(k^2 d^{2r+2})^{O(\|\sigma\|)}} (poly(\|\psi\|) + \widetilde{t}_{\text{update}} d^{O(k^2 r + k\|\sigma\|)}) \quad \leqslant \quad \widehat{t}_{\text{update}} \cdot 2^{d^{2^{O(\|\varphi\|)}}}$$

when dealing with the enumeration problem. □

8 COUNTING RESULTS OF FO+MOD QUERIES UNDER UPDATES

This section is devoted to the proof of the following theorem.

Theorem 8.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary* FO+MOD$[\sigma]$*-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$ and computes* *within* $t_{\text{preprocess}} = f(\varphi, d) \cdot \|D_0\|$ *preprocessing time a data structure that can be updated in time* $t_{\text{update}} = f(\varphi, d)$ *and allows to return the cardinality* $|\varphi(D)|$ *of the query result within time* $O(1)$.

The theorem follows immediately from Theorem 7.1 and the following dynamic counting algorithm for the query $\varphi_c(\overline{z})$.

LEMMA 8.1. *There is a dynamic algorithm that receives a number* $c \geqslant 1$, *a degree bound* $d \geqslant 2$, *and a* σ_c*-db* $\mathcal{G}_0$ *of degree* $\leqslant d$ *and computes* $|\varphi_c(\mathcal{G})|$ *with* $d^{O(c^2)}$ *initialisation time,* $O(1)$ *counting time, and* $d^{O(c^2)}$ *update time.*

PROOF. Recall from Equation (8) that $\varphi_c(z_1, \ldots, z_c) = \bigwedge_{i\in[c]} C_i(z_i) \wedge \bigwedge_{j\neq j'} \neg E(z_j, z_{j'})$. For all $j, j' \in [c]$ with $j \neq j'$ consider the formula $\theta_{j,j'}(z_1, \ldots, z_c) := E(z_j, z_{j'}) \wedge \bigwedge_{i\in[c]} C_i(z_i)$. Furthermore, let $\alpha(z_1, \ldots, z_c) := \bigwedge_{i\in[c]} C_i(z_i)$. Clearly, for every σ_c-db $\mathcal{G}$ we have

$$\alpha(\mathcal{G}) = C_1^{\mathcal{G}} \times \cdots \times C_c^{\mathcal{G}},$$

$$\varphi_c(\mathcal{G}) = \alpha(\mathcal{G}) \setminus \left(\bigcup_{j\neq j'} \theta_{j,j'}(\mathcal{G})\right), \quad \text{and, hence,} \quad |\varphi_c(\mathcal{G})| = |\alpha(\mathcal{G})| - \left|\bigcup_{j\neq j'} \theta_{j,j'}(\mathcal{G})\right|.$$

By the *inclusion-exclusion principle*, we obtain for $J := \{(j, j') : j, j' \in [c],\ j \neq j'\}$ that

$$\left|\bigcup_{j\neq j'} \theta_{j,j'}(\mathcal{G})\right| = \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|-1} \left|\bigcap_{(j,j')\in K} \theta_{j,j'}(\mathcal{G})\right| = \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|-1} |\varphi_K(\mathcal{G})|$$

for the formula $\varphi_K(z_1, \ldots, z_c) := \bigwedge_{i\in[c]} C_i(z_i) \wedge \bigwedge_{(j,j')\in K} E(z_j, z_{j'})$.

Our data structure stores the following values:

- $|C_i^{\mathcal{G}}|$, for each $i \in [c]$, and $n_1 := |\alpha(\mathcal{G})| = \prod_{i\in[c]} |C_i^{\mathcal{G}}|$,
- $|\varphi_K(\mathcal{G})|$, for each $K \subseteq J$ with $K \neq \emptyset$, and
- $n_2 := \sum_{\emptyset\neq K\subseteq J} (-1)^{|K|-1} |\varphi_K(\mathcal{G})|$ and $n_3 := n_1 - n_2$.

Note that $n_3 = |\varphi_c(\mathcal{G})|$ is the desired size of the query result. Therefore, the **count** routine can answer in time $O(1)$ by just outputting the number n_3.

It remains to show how these values can be initialised and updated during updates of $\mathcal{G}$. The initialisation for the empty graph initialises all the values to 0. In the **update** routine, the values for $|C_i^{\mathcal{G}}|$ and n_1 can be updated in a straightforward way (using time $O(c)$). For each $K \subseteq J$, the update of $|\varphi_K(\mathcal{G})|$ is provided within time $d^{O(c^2)}$ by the following Claim 8.2.

CLAIM 8.2. *For every* $K \subseteq J$, *the cardinality* $|\varphi_K(\mathcal{G})|$ *of a* σ_c*-db* $\mathcal{G}$ *of degree at most* d *can be updated within time* $d^{O(c^2)}$ *after* $d^{O(c^2)} \cdot |\mathcal{G}_0|$ *preprocessing time.* □

PROOF. Consider the directed graph $H := (V, K)$ with vertex set $V := [c]$ and edge set K. Decompose the Gaifman graph of H into its connected components. Let $V_1, \ldots, V_s$ be the connected components (for a suitable $s \leqslant c$). For each $i \in [s]$ let $H_i := H[V_i]$ be the induced subgraph of H on V_i. We write K_i to denote the set of edges of H_i. For every $i \in [s]$ let $\ell_i = |V_i|$, and let $t(i, 1) < t(i, 2) < \cdots < t(i, \ell_i)$ be the ordered list of the vertices in V_i. Consider the query

$$\varphi_{K_i}(z_{t(i,1)}, \ldots, z_{t(i,\ell_i)}) := \bigwedge_{j\in V_i} C_j(z_j) \wedge \bigwedge_{(j,j')\in K_i} E(z_j, z_{j'}). \tag{9}$$

Note that φ_K is the conjunction of the formulas φ_{K_i} for all $i \in [s]$. Since the variables of the formulas φ_{K_i} for $i \in [s]$ are pairwise disjoint, we have $\varphi_K(\mathcal{G}) = \varphi_{K_1}(\mathcal{G}) \times \cdots \times \varphi_{K_s}(\mathcal{G})$ (modulo permutations of the tuples), and thus $|\varphi_K(\mathcal{G})| = \prod_{i\in[s]} |\varphi_{K_i}(\mathcal{G})|$.

For each $i \in [s]$, the value $|\varphi_{K_i}(\mathcal{G})|$ can be computed as follows. For every $v \in \text{adom}(\mathcal{G})$ we consider the set $S_i^v := \{(w_{t(i,1)}, \ldots, w_{t(i,\ell_i)}) \in \varphi_{K_i}(\mathcal{G}) \ : \ w_{t(i,1)} = v\}$. Since the Gaifman graph of H_i is connected and has ℓ_i nodes, it follows that each component of every tuple in S_i^v is contained in the $(\ell_i - 1)$-neighbourhood of v in $\mathcal{G}$, and this neighbourhood contains at most d^{ℓ_i} elements. Therefore, $|S_i^v| \leqslant d^{(\ell_i)^2}$, and using breadth-first search starting from v, the set S_i^v can be computed in time $d^{O(c^2)}$. Note that $\varphi_{K_i}(\mathcal{G})$ is the disjoint union of the sets S_i^v for all $v \in \text{adom}(\mathcal{G})$. Therefore, $|\varphi_{K_i}(\mathcal{G})| = \sum_{v \in \text{adom}(\mathcal{G})} |S_i^v|$.

In our data structure, we store for every $i \in [s]$ and every $v \in \text{adom}(\mathcal{G})$ the number $\mu_{i,v} = |S_i^v|$. Moreover, for every $i \in [s]$ we store the sum $\mu_i = \sum_{v \in \text{adom}(\mathcal{G})} \mu_{i,v} = |\varphi_{K_i}(\mathcal{G})|$.

The initialisation for the empty σ_c-db $\mathcal{G}_0$ sets all these values to 0. Whenever the colour of a vertex of $\mathcal{G}$ is updated or an edge is inserted or deleted, we update all affected numbers accordingly. Note that a number $\mu_{i,v}$ changes only if v is in the $(c-1)$-neighbourhood around the updated edge or vertex in the graph $\mathcal{G}$. Hence, for at most $2d^c$ vertices v, the numbers $\mu_{i,v}$ are affected by an update, and each of them can be updated in time $d^{O(c^2)}$. Moreover, for each $i \in [s]$, the sum μ_i can be updated in time $O(d^c)$ by subtracting the old value of $\mu_{i,v}$ and adding the new value of $\mu_{i,v}$ for each of the at most $2d^c$ relevant vertices v. Finally, it takes time $O(c)$ to compute the updated value $|\varphi_K(\mathcal{G})| = \prod_{i \in [s]} \mu_i$. The overall time used to produce the update is $d^{O(c^2)}$. □

Once we have available the updated numbers $|\varphi_K(\mathcal{G})|$ for all $K \subseteq J$, the value n_2 can be computed in time $O(|2^J|) \leqslant 2^{O(c^2)}$. And n_3 is then obtained in time $O(1)$. Altogether, performing the **update** routine takes time at most $d^{O(c^2)}$. The **preprocess** routine initialises all values for the empty graph and then uses $|\mathcal{G}_0|$ update steps to insert all the tuples of $\mathcal{G}_0$ into the data structure. This completes the proof of Lemma 8.1. □

9 ENUMERATING RESULTS OF FO+MOD QUERIES UNDER UPDATES

In this section, we prove—and afterwards improve—the following theorem.

THEOREM 9.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary* FO+MOD$[\sigma]$*-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_{\text{update}} = f(\varphi, d)$ and allows to enumerate $\varphi(D)$ with $d^{2^{O(\|\varphi\|)}}$ delay.*

The theorem follows immediately from Theorem 7.1 and the following dynamic enumeration algorithm for the query $\varphi_c(\overline{z})$.

LEMMA 9.1. *There is a dynamic algorithm that receives a number $c \geqslant 1$, a degree bound $d \geqslant 2$, and a σ_c-db $\mathcal{G}_0$ of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = d^{poly(c)} \cdot |\mathcal{G}_0|$ preprocessing time a data structure that can be updated in time $d^{poly(c)}$ and allows us to enumerate the query result $\varphi_c(\mathcal{G})$ with $O(c^3 d)$ delay.*

PROOF. For a σ_c-db $\mathcal{G}$ and a vertex $v \in \text{adom}(\mathcal{G})$, we let $N^{\mathcal{G}}(v)$ be the set of all neighbours of v in $\mathcal{G}$, i.e., $N^{\mathcal{G}}(v)$ is the set of all $w \in \text{adom}(\mathcal{G})$ such that (v, w) or (w, v) belongs to $E^{\mathcal{G}}$.

The underlying idea of the enumeration procedure is the following greedy strategy. We cycle through all vertices $u_1 \in C_1^{\mathcal{G}}$, $u_2 \in C_2^{\mathcal{G}} \setminus N^{\mathcal{G}}(u_1)$, $u_3 \in C_3^{\mathcal{G}} \setminus (N^{\mathcal{G}}(u_1) \cup N^{\mathcal{G}}(u_2)), \ldots, u_c \in C_c^{\mathcal{G}} \setminus \bigcup_{i \leqslant c-1} N^{\mathcal{G}}(u_i)$ and output $(u_1, \ldots, u_c)$. This strategy does not yet lead to a constant delay enumeration, as there might be vertex tuples $(u_1, \ldots, u_i)$ (for $i < c$) that do extend to an output tuple $(u_1, \ldots, u_c)$, but where many possible extensions are checked before this output tuple is encountered. We now show how to overcome this problem and describe an enumeration procedure with $O(c^3 d)$ delay and update time $d^{poly(c)}$.

Note that for every $J \subseteq [c]$ we have $|\bigcup_{j \in J} N^{\mathcal{G}}(u_j)| \leqslant cd$. Hence, if a set $C_i^{\mathcal{G}}$ contains more than cd elements, then we know that *every* considered tuple has an extension $u_i \in C_i^{\mathcal{G}}$ that is not a neighbour of any vertex in the tuple. Let $I := \{i \in [c] \ : \ |C_i^{\mathcal{G}}| \leqslant cd\}$ be the set of *small* colour classes in $\mathcal{G}$ and to simplify the presentation we assume without loss of generality that $I = \{1, \ldots, s\}$. In our data structure, we store the current index set I and the set

$$\mathcal{S} \quad := \quad \left\{ (u_1, \ldots, u_s) \ \in \ C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}} \quad : \quad (u_j, u_{j'}) \notin E^{\mathcal{G}}, \ \ \text{for all } j \neq j' \right\} \tag{10}$$

of tuples on the small colours. Note that a tuple $(u_1, \ldots, u_s) \in C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}}$ extends to an output tuple $(u_1, \ldots, u_c) \in \varphi_c(\mathcal{G})$ if and only if it is contained in $\mathcal{S}$. We store the current sizes of all colours and this enables us to keep the set I of small colours updated. Moreover, as $|\mathcal{S}| \leqslant (cd)^c$, we can update the set $\mathcal{S}$ in time $d^{poly(c)}$ after every update by a brute-force approach. The enumeration procedure is given in Algorithm 1.

ALGORITHM 1: Enumeration procedure with delay $O(c^3 d)$

```
1: for all $(u_1, \ldots, u_s) \in \mathcal{S}$ do Enum$(u_1, \ldots, u_s)$.
2: Output the end-of-enumeration message EOE .
3:
4: function Enum$(u_1, \ldots, u_i)$
5:     if $i = c$ then output the tuple $(u_1, \ldots, u_c)$.
6:     else
7:         for all $u_{i+1} \in C_{i+1}^{\mathcal{G}}$ do
8:             if $u_{i+1} \notin \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ then Enum$(u_1, \ldots, u_i, u_{i+1})$.
```

It is straightforward to see that this procedure enumerates $\varphi_c(\mathcal{G})$. Let us analyse the delay. Since for all $i > s$ we have $|C_i^{\mathcal{G}}| > cd$, it follows that every call of Enum$(u_1, \ldots, u_i)$ leads to at least one recursive call of Enum$(u_1, \ldots, u_i, u_{i+1})$. Furthermore, there are at most cd iterations of the loop in line 7 that do *not* lead to a recursive call. As every test in line 8 can be done in time $O(c)$, it follows that the time spans until the first recursive call, between the calls, and after the last call are bounded by $O(c^2 d)$. As the recursion depth is c, the overall delay between two output tuples is bounded by $O(c^3 d)$. □

By using similar techniques as in Reference [7], we obtain the following improved version of Lemma 9.1, where the delay is independent of the degree bound d.

Lemma 9.2. *There is a dynamic algorithm that receives a number $c \geqslant 1$, a degree bound $d \geqslant 2$, and a σ_c-db $\mathcal{G}_0$ of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = d^{poly(c)} \cdot |\mathcal{G}_0|$ preprocessing time a data structure that can be updated in time $d^{poly(c)}$ and allows us to enumerate the query result $\varphi_c(\mathcal{G})$ with $O(c^2)$ delay.*

Before proving Lemma 9.2, let us first point out that Lemma 9.2 in combination with Theorem 7.1 directly improves the delay in Theorem 9.1 from $d^{2^{O(\|\varphi\|)}}$ to $O(k^2)$, immediately leading to the following theorem.

Theorem 9.2. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary FO+MOD$[\sigma]$-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be updated in time $t_{\text{update}} = f(\varphi, d)$ and allows to enumerate $\varphi(D)$ with $O(k^2)$ delay.*

The rest of the section is devoted to the proof of Lemma 9.2.

Proof of Lemma 9. Consider Algorithm 1, which enumerates $\varphi_c(\mathcal{G})$ with $O(c^3 d)$ delay. To enumerate the tuples with only $O(c^2)$ delay, we replace the loop in lines 7–8 by a precomputed "skip" function that allows to iterate through all elements in $C_{i+1}^{\mathcal{G}} \setminus \bigcup_{j=1}^{i} N^{\mathcal{G}}(u_j)$ with $O(c)$ delay.

For every $i \in [c]$ we store all elements of $C_i^{\mathcal{G}}$ in a doubly linked list and let void be an auxiliary element that appears at the end of the list. We let first_i be the first element in the list and $\mathsf{succ}_i(u)$ the successor of $u \in C_i^{\mathcal{G}}$. We denote by $\leqslant^i$ the linear order induced by this list. We let $\tilde{E}^{\mathcal{G}}$ be the symmetric closure of $E^{\mathcal{G}}$, i.e., $\tilde{E}^{\mathcal{G}} = E^{\mathcal{G}} \cup \{(v,u) : (u,v) \in E^{\mathcal{G}}\}$. For every $i \in [c]$, we define the function

$$\mathsf{skip}_i(y, V) \quad := \quad \min\left\{ z \in C_i^{\mathcal{G}} \cup \{\text{void}\} \;:\; y \leqslant^i z \;\text{ and }\; \text{for all } v \in V,\; (v,z) \notin \tilde{E}^{\mathcal{G}} \right\},$$

which assigns to every $V \subseteq \mathrm{adom}(\mathcal{G})$ with $|V| \leqslant c{-}1$ and every $y \in C_i^{\mathcal{G}}$ the next node in $C_i^{\mathcal{G}}$ that is not adjacent to any vertex in V.

Using these functions, our improved enumeration algorithm is given in Algorithm 2. Below, we show that we can access the values $\mathsf{skip}_i(y,V)$ in time $O(c)$. By the same analysis as given in the proof of Lemma 9.1 it then follows that Algorithm 2 enumerates $\varphi_c(\mathcal{G})$ with $O(c^2)$ delay.

ALGORITHM 2: Enumeration procedure with delay $O(c^2)$

```
1:  for all $(u_1, \ldots, u_s) \in \mathcal{S}$ do
2:      Enum$(u_1, \ldots, u_s)$.
3:  Output the end-of-enumeration message EOE.
4:
5:  function Enum$(u_1, \ldots, u_i)$
6:      if $i = c$ then
7:          output the tuple $(u_1, \ldots, u_c)$.
8:      else
9:          $y \leftarrow \mathsf{skip}_{i+1}(\mathsf{first}_{i+1}, \{u_1, \ldots, u_i\})$
10:         while $y \neq$ void do
11:             Enum$(u_1, \ldots, u_i, y)$.
12:             $y \leftarrow \mathsf{skip}_{i+1}(\mathsf{succ}_{i+1}(y), \{u_1, \ldots, u_i\})$.
```

What remains to show is that we can access the values $\mathsf{skip}_i(y,V)$ for all i, y, V in time $O(c)$ and maintain them with $d^{poly(c)}$ update time. At first sight, this is not clear at all, because the domain of skip_i has size $\Omega(|\mathrm{adom}(\mathcal{G})|^c)$. In what follows, we show that for every y, the number of distinct values that $\mathsf{skip}_i(y,V)$ can take is bounded by $d^{poly(c)}$ and that we can store them in a look-up table with update time $d^{poly(c)}$.

To illustrate the main idea, let us start with a simple example. We want to enumerate φ_4 on a coloured graph $\mathcal{H}$ with four vertex colours, blue, red, yellow, and green (in this order), and analyse the call of Enum(b, r, y), which is supposed to enumerate all green nodes g_i that are not adjacent to any of the nodes b, r, and y. The relevant part of $\mathcal{H}$ is depicted in Figure 1.

The enumeration procedure starts by considering the first element g_1 in the list of green vertices, but the first element in the actual output is $g_5 = \mathsf{skip}_4(g_1, \{b, r, y\})$. Therefore, we have to skip the irrelevant vertices $g_1, \ldots, g_4$.

To do this, we want to know the neighbours of the vertices that we skip (b and r in our example) when looking at g_1. For this purpose, we define inductively new sorts of edges $E_4^1 \subseteq E_4^2 \subseteq \cdots$ that connect green vertices $g \in \{g_1, \ldots, g_6\}$ with $\tilde{E}$-neighbours of skipped vertices. In our example, we

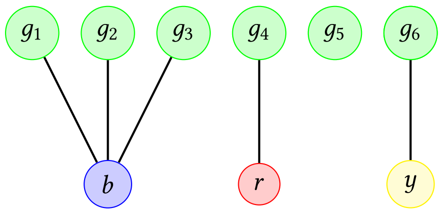

Fig. 1. Illustration of the relevant part of graph $\mathcal{H}$.

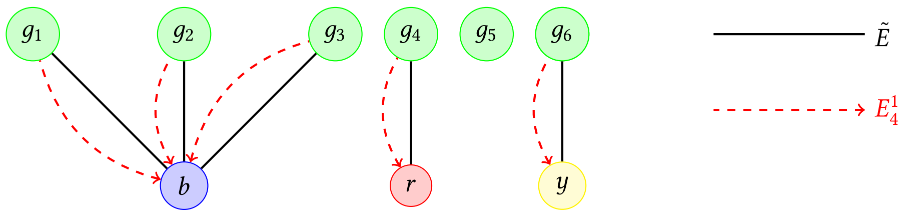

Fig. 2. $\tilde{E}$-edges and E_4^1-edges in our example.

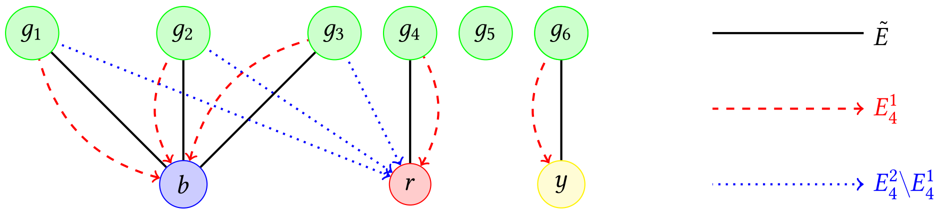

Fig. 3. $\tilde{E}$-edges, E_4^1-edges and E_4^2-edges in our example.

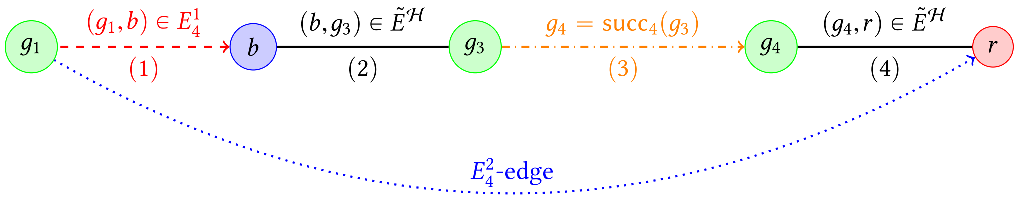
g_1
$(g_1, b) \in E_4^1$
(1)
b
$(b, g_3) \in \tilde{E}^{\mathcal{H}}$
(2)
g_3
$g_4 = \text{succ}_4(g_3)$
(3)
g_4
$(g_4, r) \in \tilde{E}^{\mathcal{H}}$
(4)
r
E_4^2-edge

Fig. 4. Introducing an E_4^2-edge between g_1 and r.

first have to skip g_1, because it is $\tilde{E}$-connected to b, and to be able to handle this, we let E_4^1 be the set of tuples $(g_i, v) \in \tilde{E}^{\mathcal{H}}$ (see Figure 2).

After realising that even more vertices (g_2 and g_3) are excluded by b, the next try would be g_4. However, this vertex is excluded by its $\tilde{E}$-neighbour r, so we have to take r into account when computing the skip value for g_1 and indicate this by the E_4^2-edge (g_1, r) (see Figure 3). This immediately leads to an inductive definition: E_4^2 contains all pairs of vertices that are already in E_4^1 or connected by a path as shown in Figure 4.

The idea outlined above can be formalised as follows. For $i, j \in [c]$, we define inductively the auxiliary edge sets E_i^j:

$$\begin{aligned} E_i^1 &:= \{\, (y,u) \,:\, y \in C_i^{\mathcal{G}} \text{ and } (y,u) \in \tilde{E}^{\mathcal{G}} \,\} \quad \text{and} \\ E_i^{j+1} &:= E_i^j \cup \Big\{ (y,u) \,:\, \text{there are } v, z \text{ with } (y,v) \in E_i^j,\ (v,z) \in \tilde{E}^{\mathcal{G}},\ (\mathrm{succ}_i(z), u) \in \tilde{E}^{\mathcal{G}} \Big\}. \end{aligned}$$

Now we define for every $y \in C_i^{\mathcal{G}}$ the set

$$S_i^y \;:=\; \{\, u \,:\, (y,u) \in E_i^c \,\}\,.$$

Note that $|S_i^y| = O(d^{2c})$. The following claim states that the elements of S_i^y are the only ones we need to take into account when computing $\mathsf{skip}_i(y, V)$.

Claim 9.3. *For all $i \leqslant c$, $y \in C_i^{\mathcal{G}} \cup \{\mathrm{void}\}$, and $V \subseteq adom(\mathcal{G})$ with $|V| \leqslant c{-}1$, it holds that*

$$\mathsf{skip}_i(y, V) \;=\; \mathsf{skip}_i(y, V \cap S_i^y). \tag{11}$$

Proof. The proof is identical to the proof of Claim 1 in Reference [7]. For the reader's convenience, we include a proof here. If $c = 1$ or $y = \mathrm{void}$, then the claim is trivial. Hence assume that $c \geqslant 2$, $y \neq \mathrm{void}$, and let $z := \mathsf{skip}_i(y, V \cap S_i^y)$. By definition, we have $y \leqslant^i z \leqslant^i \mathsf{skip}_i(y, V)$, and therefore we have to show $z \geqslant^i \mathsf{skip}_i(y, V)$, which holds if and only if $(u, z) \notin \tilde{E}^{\mathcal{G}}$ for all $u \in V \setminus S_i^y$. If $z = y$, then the claim clearly holds as all $\tilde{E}^{\mathcal{G}}$-neighbours of y are contained in S_i^y. Hence we have $z >^i y$ and let $z' \geqslant^i y$ be the predecessor of z, i.e., $z = \mathrm{succ}_i(z')$. Now assume for contradiction that there is an $u \in V \setminus S_i^y$ such that $(*)$ $(u, z) \in \tilde{E}^{\mathcal{G}}$. Note that since $z' <^i z = \mathsf{skip}_i(y, V \cap S_i^y)$, there is a $v \in V \cap S_i^y$ such that $(**)$ $(v, z') \in \tilde{E}^{\mathcal{G}}$. In the following, we show that $(***)$ $(y, v) \in E_i^{c-1}$. Note that this finishes the proof of the claim, as by the definition of E_i^c, the statements $(*)$, $(**)$, and $(***)$ imply that $u \in S_i^y$, contradicting the assumption that $u \in V \setminus S_i^y$.

To show that $(y, v) \in E_i^{c-1}$, let

$$V_j \;:=\; \{\, v' \in V \,:\, (y, v') \in E_i^j \,\} \tag{12}$$

for all $j \in [c]$. Note that $V_c = V \cap S_i^y$. Furthermore, if there is a $j < c$ with $V_j = V_{j+1}$, then we have

$$V_j \;=\; V_{j+1} \;=\; \cdots \;=\; V_c \;=\; V \cap S_i^y. \tag{13}$$

Since $|V| \leqslant c{-}1$ and $u \in V \setminus S_i^y$, we have $|V \cap S_i^y| \leqslant c - 2$. In particular, it holds that $V_{c-1} = V \cap S_i^y$. Since $v \in V \cap S_i^y$, it holds that $v \in V_{c-1}$ and thus $(y, v) \in E_i^{c-1}$. □

In our dynamic algorithm, we maintain an array that allows random access to the values $\mathsf{skip}_i(y, S')$ for all $y \in C_i^{\mathcal{G}}$ and all $S' \subseteq S_i^y$ of size at most $c{-}1$. By Claim 9.3, we can then compute $\mathsf{skip}_i(y, V)$ by first computing $S' = V \cap S_i^y$ and then looking up $\mathsf{skip}_i(y, S')$. This can be done in time $O(c)$. The next claim states that we can efficiently maintain the sets S_i^y.

Claim 9.4. *There is a data structure that*

(1) stores the elements from the sets S_i^y and all subsets $S' \subseteq S_i^y$ of cardinality at most $c{-}1$,
(2) allows to test membership in these sets in time $O(1)$, and
(3) can be updated in time $d^{poly(c)}$ after every update of the form insert $C_i(v)$, delete $C_i(v)$, insert $E(u, v)$, *and* delete $E(u, v)$.

Proof. Note that $u \in S_i^y \iff (y, u) \in E_i^c$. We store the edge sets E_i^j for all $i, j \in [c]$ in adjacency lists and additionally maintain arrays to allow constant-time access to all list entries. This allows us to store a list of elements from S_i^y and access the elements in S_i^y in constant time. Moreover, as the size of S_i^y is bounded by $O(d^{2c})$, the number of subsets $S' \subseteq S_i^y$ of cardinality at most $c-1$ is bounded by $O(d^{2c^2})$. Consequently, we can provide constant-time access to all these subsets S'.

On every insertion or deletion of an edge in $E^{\mathcal{G}}$, as well as every insertion or deletion of a vertex in $C_i^{\mathcal{G}}$, at most $O(d)$ pairs in the relation E_i^1 change, and the relation can be updated in time $O(d)$. Afterwards, we update the edge sets E_i^j according to their inductive definition. To do this efficiently, we use a breadth-first search starting from u and v, for every tuple (u, v) that has changed in relation E_i^1, up to depth $3c$ to identify the relevant nodes that are affected by the change. By using the adjacency lists, this can be done in time $d^{poly(c)}$ as the degree of the edge sets is bounded by $d^{poly(c)}$. We leave the details to the reader. □

In our data structure we store the values $\text{skip}_i(y, S')$ for every $i \in [c]$, $y \in C_i^{\mathcal{G}}$ and for all sets $S' \subseteq S_i^y$ of cardinality at most $c-1$. On every insertion or deletion of an edge, we update the sets S_i^y and their subsets S' of cardinality at most $c-1$ and update affected values of $\text{skip}_i(y, S')$. According to Claim 9.4 this can be done in time $d^{poly(c)}$.

We do the same on updates of the form insert $C_i(v)$ and delete $C_i(v)$ but have to do some additional work, as v might occur in the image of skip-functions. On insert $C_i(v)$, we insert v at the beginning of the list C_i. This ensures that existing skip values will not be affected. Afterwards, we compute the set S_i^v and the values $\text{skip}_i(v, S')$ for all $S' \subseteq S_i^v$ of cardinality at most $c-1$. Again, this can be done in time $d^{poly(c)}$.

If we receive the update delete $C_i(v)$, then we have to recompute all skip values $\text{skip}_i(y, S')$ that point to v. Note that (since $\mathcal{G}$ has degree $\leqslant d$) this is only the case for nodes $y \leqslant^i v$ whose distance from v w.r.t. succ_i is at most $(c-1)d$. Hence, it suffices to recompute $\text{skip}_i(y, S')$ for at most $(c-1)d$ vertices y and all $S' \subseteq S_i^y$ of cardinality at most $c-1$. This can be done in time $d^{poly(c)}$. By Claim 9.3, all this suffices to access the value for $\text{skip}_i(y, V)$ in time $O(c)$. This concludes the proof of Lemma 9.2. □

10 REFINING THE ENUMERATION ROUTINE

In the previous section, we have presented a dynamic algorithm that allows to enumerate with delay $O(k^2)$ the result $\varphi(D)$ of a k-ary FO+MOD-query $\varphi(x_1, \ldots, x_k)$ on a database D of degree at most d (see Theorem 9.2). In this section, we generalise this to provide the following functionality. On input of a tuple $\overline{a} = (a_1, \ldots, a_\ell) \in \mathbf{dom}^\ell$ for some $\ell \in [k]$, we would like to be able to enumerate all tuples $\overline{b} = (b_1, \ldots, b_k)$ in $\varphi(D)$ whose first ℓ components coincide with $\overline{a}$. In fact, since we already know that the first ℓ components of $\overline{b}$ coincide with $\overline{a}$, we only output the remaining components $(b_{\ell+1}, \ldots, b_k)$ of $\overline{b}$.

On input of the tuple $\overline{a}$, our algorithm spends some *preparation time* $t_{\text{preparation}}$, after which it outputs all the desired result tuples with delay t_{delay}.

For formulating this section's main result, the following notation will be convenient. Given $k \geqslant 1$ and $\ell \in [k]$, we let $m := k-\ell$. For a tuple $\overline{x} = (x_1, \ldots, x_k)$ of k pairwise distinct variables, we let $\overline{z} = (z_1, \ldots, z_\ell) := (x_1, \ldots, x_\ell)$ and $\overline{y} = (y_1, \ldots, y_m) := (x_{\ell+1}, \ldots, x_k)$.

For a k-ary FO+MOD[σ]-query $\varphi(\overline{x}) = \varphi(\overline{z}, \overline{y})$ and a tuple $\overline{a} = (a_1, \ldots, a_\ell) \in \mathbf{dom}^\ell$, we let $\varphi(\overline{a}, \overline{y})$ be the m-ary query that, when evaluated on a σ-db D, returns the result set

$$\{\, (b_1, \ldots, b_m) \ : \ (a_1, \ldots, a_\ell, b_1, \ldots, b_m) \in \varphi(D) \,\}\,.$$

This section is devoted to the proof of the following theorem.

Theorem 10.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary FO+MOD[σ]-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}_{\geqslant 1}$), a number $\ell \in [k]$ and a σ-db D_0 of degree $\leqslant d$, and computes within $t_{\text{preprocess}} = f(\varphi, d) \cdot \|D_0\|$ preprocessing time a data structure that can be* *updated in time $t_{\text{update}} = f(\varphi, d)$ and provides the following functionality: On input of an arbitrary tuple $\overline{a} = (a_1, \ldots, a_\ell) \in \mathbf{dom}^\ell$, after $t_{\text{preparation}} = f(\varphi, d)$ preparation time it enumerates with delay $O(k^2)$ all result tuples of $\varphi(\overline{a}, \overline{y})$ on D.*

For proving Theorem 10.1, we will use the following variant of Lemma 9.2, which deals with the queries

$$\varphi_J(z_{j_1}, \ldots, z_{j_c}) \quad := \quad \bigwedge_{j \in J} C_j(z_j) \ \wedge \bigwedge_{j,j' \in J,\ j \neq j'} \neg E(z_j, z_j) \tag{14}$$

for all sets $J = \{j_1, \ldots, j_c\} \subseteq [2c]$ of cardinality $|J| = c$.

Lemma 10.1. *There is a dynamic algorithm that receives a number $c \geqslant 1$, a degree bound $d \geqslant 2$, and a σ_{2c}-db $\mathcal{G}_0$ of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = d^{poly(c)} \cdot |\mathcal{G}_0|$ preprocessing time a data structure that can be updated in time $d^{poly(c)}$ and provides the following functionality: On input of an arbitrary set $J \subseteq [2c]$ of size c, after $d^{poly(c)}$ preparation time it enumerates $\varphi_J(\mathcal{G})$ with delay $O(c^2)$.*

Proof. We use the dynamic algorithm provided by the proof of Lemma 9.2 for input $2c$ instead of c. When receiving an update command, we apply this algorithm's **update** routine.

Recall that the dynamic data structure constructed in the proof of Lemma 9.2 (for $2c$ instead of c) stores the set $I \subseteq [2c]$ of *small* colour classes in $\mathcal{G}$, i.e., $i \in I$ iff $|C_i^{\mathcal{G}}| \leqslant 2cd$.

On input of a set J we proceed as follows. Compute the set $I' := I \cap J$ of all small colour classes that are relevant for the query φ_J. For simplicity, let us assume that $I' = \{1, \ldots, s\}$ for some $s \leqslant c$, and $J \setminus I' = \{s{+}1, \ldots, c\}$ (otherwise, we rename the colours accordingly). Within preparation time $d^{poly(c)}$ we can compute the set

$$\mathcal{S} \quad := \quad \left\{ (u_1, \ldots, u_s) \in C_1^{\mathcal{G}} \times \cdots \times C_s^{\mathcal{G}} \ : \ (u_j, u_{j'}) \notin E^{\mathcal{G}}, \text{ for all } j, j' \in [s] \text{ with } j \neq j' \right\}.$$

To enumerate the query result $\varphi_J(\mathcal{G})$, we then use Algorithm 2 for this set $\mathcal{S}$ (without any changes; in particular, in lines 6 and 7 we do not replace c by $2c$ but keep the value c).

Revisiting the proof of Lemma 9.2, it is straightforward to verify that the resulting dynamic data structure provides the desired functionality within the claimed preparation time and delay. □

Based on the above lemma, we can prove the following variant of Lemma 7.1.

Lemma 10.2. *For $d', c \in \mathbb{N}$ let $t_{\text{init}}(c, d') := (d')^{poly(c)}$, $t_{\text{update}}(c, d') := (d')^{poly(c)}$, and $t_{\text{delay}}(c, d') :$ $= O(c^2)$.*
For every schema σ and every $d \geqslant 2$ the following holds. Let $r \geqslant 0$, $k \geqslant 1$, and fix $d' := d^{2k^2(2r+1)}$ and $\widetilde{t_x} := \max_{c=1}^{k} t_x(c, d')$ for $t_x \in \{t_{\text{init}}, t_{\text{update}}, t_{\text{count}}, t_{\text{delay}}\}$.

(1) *Let τ be a d-bounded r-type with k centres and let $\ell \in [k]$. There is a dynamic algorithm that within initialisation time $\widetilde{t}_{\text{init}}$ builds a data structure that can be updated in time at most $\widetilde{t}_{\text{update}} d^{O(k^2 r + k\|\sigma\|)} + 2^{O(\|\sigma\| k^2 d^{2r+2})}$ and provides the following functionality for σ-dbs of degree at most d: On input of an arbitrary tuple $\overline{a} \in \mathbf{dom}^\ell$, after $\widetilde{t}_{\text{update}} 2^{O(\|\sigma\| k^2 d^{2r+2})}$ preparation time it enumerates with delay $O(\widetilde{t}_{\text{delay}} + k)$ all result tuples of $\text{sph}_\tau(\overline{a}, \overline{y})$ on D.*
(2) *Let $s \geqslant 0$ and let $\chi_1, \ldots, \chi_s$ be arbitrary sentences of schema σ, and assume we have available for each $j \in [s]$ a dynamic algorithm with initialisation time t'_{init} and update time t'_{update} that allows us to check within answer time t'_{answer} whether or not $D \models \chi_j$ for d-bounded σ-dbs D.*
   Let $\overline{x} = (x_1, \ldots, x_k)$ be a tuple of pairwise distinct variables, and let $\psi(\overline{x})$ be a Boolean combination of the sentences $\chi_1, \ldots, \chi_s$ and of d-bounded sphere-formulas of radius at most r (over σ).

There is a dynamic algorithm that within initialisation time $s(t'_{\text{init}} + t'_{\text{answer}}) + 2^{(kd^{r+1})^{O(\|\sigma\|)}}(poly(\|\psi\|) + \widetilde{t}_{\text{init}})$ *builds a data structure that can be updated in time* $s(t'_{\text{update}} + t'_{\text{answer}}) + 2^{(k^2d^{2r+2})^{O(\|\sigma\|)}}(poly(\|\psi\|) + \widetilde{t}_{\text{update}}d^{O(k^2r+k\|\sigma\|)})$ *and provides the following functionality: On input of a tuple* $\overline{a} \in \mathbf{dom}^{\ell}$*, after* $2^{(k^2d^{2r+2})^{O(\|\sigma\|)}} \cdot \widetilde{t}_{\text{update}}$ *preparation time it enumerates with delay* $O(\widetilde{t}_{\text{delay}} + k)$ *all result tuples of* $\psi(\overline{a}, \overline{y})$ *on* D*.*

Proof. For the proof of part (1) we proceed in a similar way as in the proof of Lemma 7.1(1), and utilise the dynamic algorithm provided by Lemma 10.1.

On input of a tuple $\overline{a} = (a_1, \ldots, a_\ell) \in \mathbf{dom}^\ell$, we want to enumerate all tuples $\overline{b} \in \mathbf{dom}^m$ such that $D \models \text{sph}_\tau(\overline{a}, \overline{b})$. We use the same notation as in the proof of Lemma 7.1(1). In particular, recall that we consider the formula

$$\text{conn-sph}_\tau(\overline{x}) \quad := \quad \bigwedge_{j\in[c]} \text{sph}_{\tau_{j,\nu_j}}(\overline{x}_j) \ \wedge \ \bigwedge_{j\neq j'} \neg\ \text{dist}^{k_j,k_{j'}}_{\leqslant 2r+1}(\overline{x}_j, \overline{x}_{j'})$$

and the coloured graph $\mathcal{G}_D$ of degree at most d' with

$$C_j^{\mathcal{G}_D} \ := \ \{ v_{\overline{b}} \ : \ \overline{b} \in \text{adom}(D)^{k_j} \text{ with } \big(\mathcal{N}_r^D(\overline{b}), \overline{b}\big) \cong \tau_j \}, \quad \text{for all } j \in [c], \quad \text{and}$$
$$E^{\mathcal{G}_D} \ := \ \{ (v_{\overline{b}}, v_{\overline{b}'}) \in V^2 \ : \ dist^D(\overline{b}; \overline{b}') \leqslant 2r{+}1 \},$$

where $V := \bigcup_{j\in[c]} C_j^{\mathcal{G}_D}$, and recall that $\tau_j := \tau_{j,\nu_j}$ is a connected r-type for each $j \in [c]$.

Recall that the input tuple $\overline{a}$ will be assigned to the variables $(x_1, \ldots, x_\ell) = (z_1, \ldots, z_\ell) = \overline{z}$. W.l.o.g. we assume that there is a number $\kappa \in \{1, \ldots, c\}$ such that the following is true. For each $j \in [\kappa]$, the tuple $\overline{x}_j$ contains at least one of the variables in $\overline{z}$, and for each $j \in [k] \setminus [\kappa]$, the tuple $\overline{x}_j$ contains none of the variables in $\overline{z}$, i.e., it only consists of variables in $\overline{y} = (y_1, \ldots, y_m) = (x_{\ell+1}, \ldots, x_k)$.

For an input tuple $\overline{a}$, we extend $\mathcal{G}_D$ by c further colours $C_{c+1}, \ldots, C_{2c}$ to obtain the following σ_{2c}-db $\mathcal{G}_{D,\overline{a}}$. The edge relation E and the colours $C_1, \ldots, C_c$ of $\mathcal{G}_{D,\overline{a}}$ are the same as those of $\mathcal{G}_D$. For each $j \in \{1, \ldots, \kappa\}$, we let

$$C_{c+j}^{\mathcal{G}_{D,\overline{a}}} \quad := \quad \left\{ v_{\overline{b}} \in C_j^{\mathcal{G}_D} \ : \ \begin{array}{l} \text{for every position } \nu \text{ in } \overline{x}_j \text{ which consists of a variable } z_i \\ \text{in } \overline{z}, \text{ the entry of } \overline{b} \text{ at position } \nu \text{ is } a_i \end{array} \right\},$$

and for each $j \in \{\kappa{+}1, \ldots, c\}$, we let $C_{c+j}^{\mathcal{G}_{D,\overline{a}}} := \emptyset$.

To enumerate the result tuples of $\text{sph}_\tau(\overline{a}, \overline{y})$, we use the **enumerate** routine provided by Lemma 10.1 on input

$$J \quad := \quad \{c{+}1, \ldots, c{+}\kappa\} \ \cup \ \{\kappa{+}1, \ldots, c\}$$

to enumerate the set $\varphi_J(\mathcal{G}_{D,\overline{a}})$. Note that according to the definition of the sets $C_{c+j}^{\mathcal{G}_{D,\overline{a}}}$, the set $\varphi_J(\mathcal{G}_{D,\overline{a}})$ contains exactly those tuples in $(v_{\overline{b}_1}, \ldots, v_{\overline{b}_c}) \in \varphi_J(\mathcal{G}_D)$, where for every position ν in some $\overline{x}_j$ that consists of a variable z_i in $\overline{z}$, the entry of $\overline{b}_j$ at position ν is a_i. Therefore, after suitably re-ordering the components of the tuples $(\overline{b}_1, \ldots, \overline{b}_c)$ and dropping the components that correspond to $\overline{a}$, we obtain the desired result tuples for $\text{sph}_\tau(\overline{a}, \overline{y})$ on D.

It remains to show how to construct and maintain $\mathcal{G}_{D,\overline{a}}$ when the database is updated or when a new tuple $\overline{a}$ is given as input.

The **update** routine proceeds in exactly the same way as in the proof of Lemma 7.1(1) and uses the **update** routine provided by Lemma 10.1 but does not bother to update the colours $C_{c+1}, \ldots, C_{2c}$.

On input of a tuple $\overline{a}$, we use the preparation time to first delete all elements from the colours $C_{c+1}, \ldots, C_{2c}$, and then insert all the elements that belong to the sets $C_{c+1}^{\mathcal{G}_{D,\overline{a}}}, \ldots, C_{c+\kappa}^{\mathcal{G}_{D,\overline{a}}}$. The details can be carried out as follows.

Note that according to the definition of $C_{c+j}^{\mathcal{G}_{D,\overline{a}}}$, the following is true for each $j \in [\kappa]$: For every element $v_{\overline{b}}$ in $C_{c+j}^{\mathcal{G}_{D,\overline{a}}}$, some component of the tuple $\overline{b}$ is equal to a component a_i of the tuple $\overline{a}$. Since $(\mathcal{N}_r^D(\overline{b}), \overline{b}) \cong \tau_j$ and τ_j is a connected r-type, every component of the tuple $\overline{b}$ belongs to $\mathcal{N}_{r'}^D(a_i)$ for $r' := r + (k-1)(2r+1)$. In particular, from Lemma 3.1, we obtain that $|C_{c+j}^{\mathcal{G}_{D,\overline{a}}}| \leqslant |\mathcal{N}_{r'}^D(a_i)|^k \leqslant d^{k^2(2r+1)}$. Thus, the first step of deleting all elements that are still present in C_{c+j} (as an artifact of a previous enumeration request) can be accomplished in time $d^{O(k^2 r)}\widetilde{t}_{\mathrm{update}}$.

Afterwards, we proceed as follows to insert into C_{c+j} all elements that do belong to $C_{c+j}^{\mathcal{G}_{D,\overline{a}}}$. We compute the set $U := \mathcal{N}_{r'}^D(a_i)$ and test for every tuple $\overline{b} \in U^{|\overline{x}_j|}$ whether the following two conditions hold:

(1) The r'-type $(\mathcal{N}_{r'}^D(\overline{b}), \overline{b})$ is isomorphic to τ_j, and
(2) for every position ν in $\overline{x}_j$ that consists of a variable z_μ in $\overline{z}$, the entry of $\overline{b}$ at position ν is a_μ.

If both conditions are met, then we insert the vertex $v_{\overline{b}}$ into C_{c+j} by using the **update** routine provided by Lemma 10.1. Note that this constructs the correct set $C_{c+j}^{\mathcal{G}_{D,\overline{a}}}$.

To analyse the time needed for this construction, note that by Lemma 3.1, $|U| \leqslant d^{r'+1} \leqslant d^{k(2r+1)} \leqslant d^{O(kr)}$. Furthermore, U can be computed in time $(d^{r'+1})^{O(\|\sigma\|)} \leqslant d^{O(kr\|\sigma\|)}$. The number of tuples $\overline{b}$ that we consider is at most $|U|^k \leqslant d^{O(k^2 r)}$. To check if the first condition for $\overline{b}$ is met, then we use Lemma 3.1(e) to check in time $2^{O(\|\sigma\| k^2 d^{2r+2})}$ whether the r-type of $\overline{b}$ is isomorphic to τ_j. The second condition can be checked in time $O(k)$. In summary, we compute the sets $C_{c+j}^{\mathcal{G}_{D,\overline{a}}}$ for all $j \in [\kappa]$ in time $d^{O(k^2 r\|\sigma\|)} \cdot 2^{O(\|\sigma\| k^2 d^{2r+2})} \cdot \widetilde{t}_{\mathrm{update}} \leqslant 2^{O(\|\sigma\| k^2 d^{2r+2})} \cdot \widetilde{t}_{\mathrm{update}}$.

This completes the proof of part (1).

Let us now turn to the proof of part (2). The proof can be taken verbatim from the proof of Lemma 7.1(2), with the following changes: Instead of using the dynamic algorithms for $\mathrm{sph}_{\tau_i}(\overline{x})$ provided from Lemma 7.1(1), we now use the dynamic algorithms for $\mathrm{sph}_{\tau_i}(\overline{z}, \overline{y})$ provided by Lemma 10.2(1). On input of a tuple $\overline{a} \in \mathbf{dom}^\ell$, we run the preparation phase of the dynamic algorithms for $\mathrm{sph}_{\tau_i}(\overline{a}, \overline{y})$ for all $i \in I$, and afterwards, we loop through all $i \in I$ and enumerate the result tuples of $\mathrm{sph}_{\tau_i}(\overline{a}, \overline{y})$ on D. This completes the proof of Lemma 10.2. □

By using Lemma 10.2(2) instead of Lemma 7.1(2), we obtain the following analogue to Theorem 7.1.

Lemma 10.3. *For $d', c \in \mathbb{N}$ let $t_{\mathrm{init}}(c, d') = (d')^{poly(c)}$, $t_{\mathrm{update}}(c, d') = (d')^{poly(c)}$, and $t_{\mathrm{delay}}(c, d') = O(c^2)$. There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary* FO+MOD$[\sigma]$*-query $\varphi(\overline{x})$ (for some $k \in \mathbb{N}_{\geqslant 1}$), and a number $\ell \in [k]$. Within initialisation time $\widehat{t}_{\mathrm{init}} \cdot f(\varphi, d)$, this algorithm builds a data structure that can be updated in time $\widehat{t}_{\mathrm{update}} \cdot f(\varphi, d)$ and provides the following functionality for σ-dbs of degree at most d: On input of an arbitrary tuple $\overline{a} \in$ **dom**$^\ell$, after $\widehat{t}_{\mathrm{update}} \cdot f(\varphi, d)$ preparation time it enumerates with delay $O(\widehat{t}_{\mathrm{delay}} + k)$ all result tuples of $\varphi(\overline{a}, \overline{y})$ on D , where $\widehat{t_x} = \max_{c=1}^k t_x(c, d^{2^{O(\|\varphi\|)}})$ for $t_x \in \{t_{\mathrm{init}}, t_{\mathrm{update}}, t_{\mathrm{delay}}\}$.*

Proof. The proof can be taken verbatim from the proof of Theorem 7.1, with the following change: Instead of applying part (2) of Lemma 7.1, we now use the dynamic algorithm provided by part (2) of Lemma 10.2. □

Proof of Theorem 10.1. The result is an immediate consequence of Lemma 10.3. □

11 ENUMERATING THE DIFFERENCE

In this section, we introduce a new update routine called **update_and_report_diff** that, immediately after performing the database update, reports the difference between the new query result and the old query result, i.e., it first enumerates all tuples in $\varphi(D_{new}) \setminus \varphi(D_{old})$ (terminated by the end-of-enumeration message EOE) and then all tuples in $\varphi(D_{old}) \setminus \varphi(D_{new})$ (again, terminated by the message EOE), where D_{old} and D_{new} denote the database before and after performing the given update command. This section's main result reads as follows.

Theorem 11.1. *There is a dynamic algorithm that receives a schema σ, a degree bound $d \geqslant 2$, a k-ary FO+MOD$[\sigma]$-query $\varphi(\overline{y})$ (for some $k \in \mathbb{N}$), and a σ-db D_0 of degree $\leqslant d$ and computes within $t_{\text{preprocess}} = f(\varphi, d)\cdot\|D_0\|$ preprocessing time a data structure providing an* **update_and_report_diff** *routine that, on input of a command update $R(\overline{a})$ for update $\in$ {insert, delete}, $R \in \sigma$, and $\overline{a} \in \textbf{dom}^{\text{ar}(R)}$, updates the data structure within time $t_{\text{update}} = f(\varphi, d)$ and then enumerates the difference between the new and the old query result with delay $O(k^2)$.*

Proof. We use Theorem 10.1 for the following queries. For each update $\in$ {insert, delete} and each $R \in \sigma$ we let $\ell := \text{ar}(R)$ and fix a tuple $\overline{z} = (z_1, \ldots, z_\ell)$ of pairwise distinct variables that do not occur in $\varphi(\overline{x})$. We construct $(\ell{+}k)$-ary FO+MOD$[\sigma]$-queries

$$\varphi^{+}_{\text{update}\,R}(\overline{z}, \overline{x}) \qquad \text{and} \qquad \varphi^{-}_{\text{update}\,R}(\overline{z}, \overline{x})$$

such that the following is true for all σ-dbs D and all $\overline{a} \in \textbf{dom}^{\ell}$: If D_{new} is the σ-db obtained from $D_{old} \neq D_{new}$ by performing the update operation update $R(\overline{a})$, then

- the set of result tuples of $\varphi^{+}_{\text{update}\,R}(\overline{a}, \overline{x})$ on D_{new} is exactly the set $\varphi(D_{new}) \setminus \varphi(D_{old})$, and
- the set of result tuples of $\varphi^{-}_{\text{update}\,R}(\overline{a}, \overline{x})$ on D_{new} is exactly the set $\varphi(D_{old}) \setminus \varphi(D_{new})$.

Before constructing these queries let us explain how they can be used to finish the proof of Theorem 11.1. Let Ψ be the set consisting of the queries $\varphi^{+}_{\text{update}\,R}$ and $\varphi^{-}_{\text{update}\,R}$ for update $\in$ {insert, delete} and $R \in \sigma$. We use in parallel for each ψ in Ψ the dynamic algorithm provided by Theorem 10.1. On input of an update operation update $R(\overline{a})$, the **update_and_report_diff** routine proceeds as follows. Let $D = D_{old}$ be the database before executing the update command.

In the case where the given update command does not change the database (i.e., the operation intends to delete a tuple that does not belong to the database relation R^D, or it intends to insert a tuple that already belongs to R^D, or it intends to insert a tuple that would result in a database that exceeds the given degree bound d), then all the data structures remain unchanged and the routine just outputs "EOE EOE."

Otherwise, we proceed as follows. First, consider each query ψ in Ψ and perform the **update** routine on input "update $R(\overline{a})$" of the dynamic algorithm provided by Theorem 10.1 for the query ψ. Let D_{new} be the updated database. We then use the functionality provided by Theorem 10.1 to perform in parallel the preparation phase for the queries $\varphi^{+}_{\text{update}\,R}(\overline{z}, \overline{x})$ and $\varphi^{-}_{\text{update}\,R}(\overline{z}, \overline{x})$ on input of the tuple $\overline{a}$. Afterwards, we first enumerate the result tuples of $\varphi^{+}_{\text{update}\,R}(\overline{a}, \overline{x})$ on D_{new} and then enumerate the result tuples of $\varphi^{-}_{\text{update}\,R}(\overline{a}, \overline{x})$ on D_{new}.

All that remains to be done to finish the proof of Theorem 11.1 is to construct the queries $\varphi^+_{\text{update}\,R}(\overline{z},\overline{x})$ and $\varphi^-_{\text{update}\,R}(\overline{z},\overline{x})$. The idea is straightforward: We let

$$\varphi^+_{\text{update}\,R}(\overline{z},\overline{x}) := \varphi(\overline{x}) \ \wedge\ \neg\, \varphi^{old}_{\text{update}\,R}(\overline{z},\overline{x}) \qquad \text{and}$$
$$\varphi^-_{\text{update}\,R}(\overline{z},\overline{x}) := \varphi^{old}_{\text{update}\,R}(\overline{z},\overline{x}) \ \wedge\ \neg\, \varphi(\overline{x})\,,$$

where $\varphi^{old}_{\text{update}\,R}(\overline{z},\overline{x})$ is a formula for which the following is true: If $D_{new} \neq D_{old}$ is obtained from D_{old} by performing the update command update $R(\overline{a})$, then evaluating $\varphi^{old}_{\text{update}\,R}(\overline{a},\overline{x})$ on D_{new} simulates the evaluation of $\varphi(\overline{x})$ on D_{old}. A somewhat annoying technical detail in the construction of these formulas is that the semantics of quantifiers takes into account the database's active domain, and the database's active domain might be changed by the update command.

Let us first consider the (easier) case that update = insert. We let

$$\alpha(\overline{z},y) := \exists y_1 \cdots \exists y_\ell \left(\bigvee_{i=1}^{\ell} y{=}y_i \ \wedge\ R(y_1,\ldots,y_\ell) \ \wedge\ \neg \bigwedge_{i=1}^{\ell} y_i{=}z_i \right)$$
$$\vee \bigvee_{S\in\sigma\setminus\{R\}} \exists y_1 \cdots \exists y_{\text{ar}(S)} \left(\bigvee_{i=1}^{\text{ar}(S)} y{=}y_i \ \wedge\ S(y_1,\ldots,y_{\text{ar}(S)}) \right).$$

If $D_{new} \neq D_{old}$ is obtained from D_{old} by performing the update command insert $R(\overline{a})$, then the result set of $\alpha(\overline{a},y)$ on D_{new} is exactly the active domain of D_{old}. Therefore, we can choose $\varphi^{old}_{\text{insert}\,R}(\overline{z},\overline{x})$ to be the query obtained from the input query $\varphi(\overline{x})$ by replacing every atomic subformula of the form $R(u_1,\ldots,u_\ell)$ with the formula $(R(u_1,\ldots,u_\ell) \ \wedge\ \neg \bigwedge_{i=1}^{\ell} u_i{=}z_i)$ and by relativising every quantification to a variable y to those y that satisfy $\alpha(\overline{z},y)$, i.e., we replace every subformula of the form $\exists y\, \vartheta$ (or $\exists^{i \bmod m} y\, \vartheta$) with the formula $\exists y\, (\alpha(\overline{z},y) \wedge \vartheta)$ (or $\exists^{i \bmod m} y\, (\alpha(\overline{z},y) \wedge \vartheta)$). It is straightforward to verify that the resulting formula $\varphi^{old}_{\text{insert}\,R}(\overline{z},\overline{x})$ expresses the desired property.

Let us now turn to the case where update = delete. The problem here is that $\text{adom}(D_{old})$ contains all the elements in $\overline{a} = (a_1,\ldots,a_\ell)$, while some (or, all) of these elements might be missing in $\text{adom}(D_{new})$, and due to the active domain semantics of FO+MOD, there is no explicit means of enabling quantifiers to range over elements that do not belong to the active domain. To overcome this, let $J \subseteq [\ell]$ be a set of indices such that $|J| = |\{a_1,\ldots,a_\ell\}|$ and $\{a_1,\ldots,a_\ell\} = \{a_j : j \in J\}$. By induction on the construction of formulas we define for every FO+MOD[σ]-query $\vartheta(\overline{y})$ that does not contain any of the variables in $\overline{z} = (z_1,\ldots,z_\ell)$ an FO+MOD[σ]-query $\hat{\vartheta}_J(\overline{z},\overline{y})$ such that the set of result tuples of $\hat{\vartheta}_J(\overline{a},\overline{y})$ on D_{new} is exactly the set $\vartheta(D_{old})$. To achieve this, we proceed as follows:

- if ϑ is of the form $R(y_1,\ldots,y_\ell)$, then $\hat{\vartheta}_J := (R(y_1,\ldots,y_\ell) \ \vee\ \bigwedge_{i=1}^{\ell} y_i{=}z_i)$
- if ϑ is of the form $y_1{=}y_2$ or of the form $S(y_1,\ldots,y_{\text{ar}(S)})$ with $S \in \sigma \setminus \{R\}$, then $\hat{\vartheta}_J := \vartheta$
- if ϑ is of the form $\neg\theta$, then $\hat{\vartheta}_J := \neg\hat{\theta}_J$
- if ϑ is of the form $(\theta' \vee \theta'')$, then $\hat{\vartheta}_J := (\widehat{\theta'}_J \vee \widehat{\theta''}_J)$
- if ϑ is of the form $\exists y\, \theta$, then $\hat{\vartheta}_J := (\vartheta \ \vee\ \bigvee_{j\in J} \hat{\theta}_J \frac{z_j}{y})$, where $\hat{\theta}_J \frac{z_j}{y}$ is the formula obtained from $\hat{\theta}_J$ by replacing every free occurrence of the variable y by the variable z_j
- if ϑ is of the form $\exists^{i \bmod m} y\, \theta$, then $\hat{\vartheta}_J := \bigvee_{(i_1,i_2)\in I} \xi_{(i_1,i_2)}$, where I is the set of all $(i_1,i_2) \in \{0,\ldots,m{-}1\}^2$ with $i_1 + i_2 \equiv i \bmod m$, and

$$\xi_{(i_1,i_2)} \quad := \quad \exists^{i_1 \bmod m} y \left(\hat{\theta}_J \wedge \bigwedge_{j\in J} \neg\, y{=}z_j \right) \ \wedge \bigvee_{\substack{J'\subseteq J\\ |J'|=i_2}} \left(\bigwedge_{j\in J'} \hat{\theta}_J \tfrac{z_j}{y} \ \wedge \bigwedge_{j\in J\setminus J'} \neg\hat{\theta}_J \tfrac{z_j}{y} \right).$$

It is straightforward to verify that the query $\hat{\vartheta}_J(\overline{z},\overline{y})$ indeed has the desired meaning. Thus, we can choose

$$\varphi^{old}_{\text{delete } R}(\overline{z},\overline{x}) \quad := \quad \bigvee_{J\subseteq[\ell]} \left(\hat{\varphi}_J \quad \wedge \bigwedge_{\substack{i,j\in J\\ i\neq j}} \neg\, z_i{=}z_j \quad \wedge \bigwedge_{i\in[\ell]\setminus J}\ \bigvee_{j\in J} z_i{=}z_j \right).$$

This completes the proof of Theorem 11.1. □

12 CONCLUSION

Our main results show that in the dynamic setting (i.e., allowing database updates), the results of k-ary FO+MOD-queries on bounded degree databases can be tested and counted in constant time and enumerated with constant delay, after linear time preprocessing and with constant update time. Here, "constant time" refers to data complexity and is of size $poly(k)$ concerning the delay and the time for testing and counting. The time for performing a database update is threefold exponential in the size of the query and the degree bound and is worst-case optimal.

The starting point of our algorithms is to decompose the given query into a query in Hanf normal form, using a recent result of Ref. [12]. This normal form is only available for the setting with a fixed maximum degree bound d, i.e., the setting considered in this article.

Recently, Kuske and Schweikardt [15] introduced a new kind of Hanf normal form for a variant of *first-order logic with counting* that contains and extends Libkin's logic FO(Cnt) [16] and Grohe's logic FO+C [9]. As an application it is shown in Reference [15] that the present article's techniques can be lifted from FO+MOD to first-order logic with counting terms and numerical predicates FOC($\mathbb{P}$).

An obvious future task is to investigate to which extent further query evaluation results that are known for the static setting can be lifted to the dynamic setting. More specifically: Are there efficient dynamic algorithms for evaluating (i.e., answering, testing, counting, or enumerating) results of first-order queries on other sparse classes of databases (e.g. planar, bounded treewidth, bounded expansion, nowhere dense) or databases of low degree, lifting the "static" results accumulated in [7, 10, 14, 23] to the dynamic setting?

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[2] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering conjunctive queries under updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, (PODS'17)*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, 303–318. DOI: http://dx.doi.org/10.1145/3034786.3034789

[3] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering FO+MOD queries under updates on bounded degree databases. In *Proceedings of the 20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy (LIPIcs)*, Michael Benedikt and Giorgio Orsi (Eds.), Vol. 68. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:18. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2017.8

[4] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2018. Answering UCQs under Updates and in the presence of integrity constraints. In *Proceedings of the 21st International Conference on Database Theory (ICDT'18)*. 8:1–8:19. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2018.8

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). MIT Press.

[6] Arnaud Durand and Etienne Grandjean. 2007. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.* 8, 4 (2007). DOI: http://dx.doi.org/10.1145/1276920.1276923

[7] Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. 2014. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*. 121–131. DOI: http://dx.doi.org/10.1145/2594538.2594539

[8] Markus Frick and Martin Grohe. 2004. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic* 130, 1–3 (2004), 3–31. DOI: http://dx.doi.org/10.1016/j.apal.2004.01.007

[9] Martin Grohe. 2017. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic, Vol. 47. Association for Symbolic Logic in conjunction with Cambridge University Press.

[10] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. 2017. Deciding first-order properties of nowhere dense graphs. *J. ACM* 64, 3 (2017), 17:1–17:32. DOI: http://dx.doi.org/10.1145/3051095 Conference version: in Proceedings of the 46th ACM Symposium on Theory of Computing (STOC'14), pp. 89–98, 2014.

[11] Martin Grohe and Nicole Schweikardt. 2018. First-order query evaluation with cardinality conditions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'18)*. DOI: http://dx.doi.org/10.1145/3196959.3196970

[12] Lucas Heimberg, Dietrich Kuske, and Nicole Schweikardt. 2016. Hanf normal form for first-order logic with unary counting quantifiers. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'16)*. 277–286. DOI: http://dx.doi.org/10.1145/2933575.2934571

[13] Wojciech Kazana and Luc Segoufin. 2011. First-order query evaluation on structures of bounded degree. *Logic. Methods Comput. Sci.* 7, 2 (2011). DOI: http://dx.doi.org/10.2168/LMCS-7(2:20)2011

[14] Wojciech Kazana and Luc Segoufin. 2013. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'13)*. 297–308. DOI: http://dx.doi.org/10.1145/2463664.2463667

[15] Dietrich Kuske and Nicole Schweikardt. 2017. First-order logic with counting: At least, *weak* Hanf normal forms always exist and can be computed! In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'17)*. Full version available at CoRR abs/1703.01122.

[16] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer. DOI: http://dx.doi.org/10.1007/978-3-662-07003-1

[17] Eugene M. Luks. 1982. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.* 25, 1 (1982), 42–65. DOI: http://dx.doi.org/10.1016/0022-0000(82)90009-5

[18] Bernard M. E. Moret and Henry D. Shapiro. 1991. *Algorithms from P to NP: Volume 1: Design & Efficiency*. Benjamin-Cummings.

[19] Sushant Patnaik and Neil Immerman. 1997. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.* 55, 2 (1997), 199–209. DOI: http://dx.doi.org/10.1006/jcss.1997.1520

[20] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. 2018. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'18)*. DOI: http://dx.doi.org/10.1145/3196959.3196971

[21] Thomas Schwentick and Thomas Zeume. 2016. Dynamic complexity: Recent updates. *SIGLOG News* 3, 2 (2016), 30–52. DOI: http://dx.doi.org/10.1145/2948896.2948899

[22] Detlef Seese. 1996. Linear time computable problems and first-order descriptions. *Math. Struct. Comput. Sci.* 6, 6 (1996), 505–526.

[23] Luc Segoufin and Alexandre Vigny. 2017. Constant delay enumeration for FO queries over databases with local bounded expansion. In *Proceedings of the 20th International Conference on Database Theory (ICDT'17)*. 20:1–20:16. DOI: http://dx.doi.org/10.4230/LIPIcs.ICDT.2017.20

Received July 2017; revised April 2018; accepted June 2018