



Twin-Width and Types

Jakub Gajarský  

University of Warsaw, Poland

Michał Pilipczuk  

University of Warsaw, Poland

Wojciech Przybyszewski  

University of Warsaw, Poland

Szymon Toruńczyk  

University of Warsaw, Poland

Abstract

We study problems connected to first-order logic in graphs of bounded twin-width. Inspired by the approach of Bonnet et al. [FOCS 2020], we introduce a robust methodology of *local types* and describe their behavior in contraction sequences – the decomposition notion underlying twin-width. We showcase the applicability of the methodology by proving the following two algorithmic results. In both statements, we fix a first-order formula $\varphi(x_1, \dots, x_k)$ and a constant d , and we assume that on input we are given a graph G together with a contraction sequence of width at most d .

- One can in time $\mathcal{O}(n)$ construct a data structure that can answer the following queries in time $\mathcal{O}(\log \log n)$: given w_1, \dots, w_k , decide whether $\varphi(w_1, \dots, w_k)$ holds in G .
- After $\mathcal{O}(n)$ -time preprocessing, one can enumerate all tuples w_1, \dots, w_k that satisfy $\varphi(x_1, \dots, x_k)$ in G with $\mathcal{O}(1)$ delay.

In the first case, the query time can be reduced to $\mathcal{O}(1/\varepsilon)$ at the expense of increasing the construction time to $\mathcal{O}(n^{1+\varepsilon})$, for any fixed $\varepsilon > 0$. Finally, we also apply our tools to prove the following statement, which shows optimal bounds on the VC density of set systems that are first-order definable in graphs of bounded twin-width.

- Let G be a graph of twin-width d , A be a subset of vertices of G , and $\varphi(x_1, \dots, x_k, y_1, \dots, y_l)$ be a first-order formula. Then the number of different subsets of A^k definable by φ using l -tuples of vertices from G as parameters, is bounded by $\mathcal{O}(|A|^l)$.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases twin-width, FO logic, model checking, query answering, enumeration

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.123

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: <https://arxiv.org/abs/2206.08248>

Funding This work is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 948057).

Acknowledgements The authors thank Rose McCarty and Felix Reidl for many initial discussions on the type approach to first-order logic on graphs of bounded twin-width.

1 Introduction

Twin-width is a graph parameter recently introduced by Bonnet et al. [7]. Its definition is based on the concept of a *contraction sequence*: a sequence of partitions of the vertex set of the graph that starts with the partition into singletons, where every subsequent partition is



© Jakub Gajarský, Michał Pilipczuk, Wojciech Przybyszewski, and Szymon Toruńczyk; licensed under Creative Commons License CC-BY 4.0

49th International Colloquium on Automata, Languages, and Programming (ICALP 2022).

Editors: Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff;

Article No. 123; pp. 123:1–123:21



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



obtained from the previous one by merging two parts ending with the partition with one part. The main idea lies in measuring the *width* of a contraction sequence: it is the smallest integer d such that at every step, every part of the current partition is *impure* – neither completely adjacent nor completely non-adjacent – towards at most d other parts of that partition. The *twin-width* of a graph G is the smallest possible width of a contraction sequence of G . Thus, one may think that a graph of bounded twin-width can be gradually “folded” into a single part so that at every point, every part has a non-trivial interaction with only a bounded number of other parts.

We remark that while twin-width was originally defined for graphs, the idea can be, and has been, generalized to any classes of binary relational structures, for instance ordered graphs [4] or permutations [8]. In this work we focus on the graph setting for simplicity. However, all our results lift to arbitrary structures over a fixed relational signature in which all relation symbols have arities at most two.

Since its recent introduction, multiple works have investigated combinatorial, algorithmic, and model-theoretic aspects of twin-width. In this work we are mostly interested in the two last ones. As proved by Bonnet et al. [7], provided a graph G is given together with a contraction sequence of width bounded by a constant, every property expressible in first-order logic can be verified in linear time on G ; in other words, the model-checking problem for first-order logic can be solved in linear fixed-parameter tractable time. Further, bounded twin-width is preserved under transductions: if a class of graphs \mathcal{C} has bounded twin-width, then any class that can be obtained from \mathcal{C} by a fixed (first-order) transduction also has bounded twin-width [7]. Finally, as proved by Bonnet et al. [4], classes of *ordered* graphs that have bounded twin-width exactly coincide with those that are *monadically NIP*, that is, do not transduce all graphs. All these results witness that twin-width is a model-theoretically important notion and a vital element of the emerging structural theory for graphs based around the notion of a (first-order) transduction. See [8, 13] for further discussion.

In this work we take a closer look at the model-checking algorithm for graphs of bounded twin-width, proposed in [7]. The basic technical notion used there is that of a *morphism tree*. While this is not explicit in [7], it is clear that morphism trees are combinatorial objects representing strategies in a form of an Ehrenfeucht-Fraïssé game, and basic operations on morphism trees correspond to manipulations on strategies. Mirroring the standard approach taken in finite model theory, one should be able to define a notion of a *type* suited for the setting of contraction sequences, as well as a corresponding model of an Ehrenfeucht-Fraïssé game that can be used to argue about properties of types such as compositionality. Providing robust foundations for such a type-based methodology for contraction sequences is the main goal of this work.

We remark that the type/game based perspective of the approach of [7], which we explained above, was recently briefly outlined in [5, Section 5].

Our contribution. We introduce the notion of a *local type* that is suited for describing first-order properties of tuples of vertices in vertex-partitioned graphs. Intuitively speaking, the rank- k local type of a tuple \mathbf{w} in a graph G with vertex partition \mathcal{P} is the set of all quantifier rank k formulas satisfied by \mathbf{w} , where we restrict quantification as follows. Whenever a new vertex, say z , is quantified, one has to specify the part $P \in \mathcal{P}$ which contains z , but at depth i of quantification one can quantify only over parts that are at distance at most 2^{k-i} from parts containing already quantified vertices (including vertices of \mathbf{w}). Here, we mean the

distance in the *impurity graph*: the graph on parts of \mathcal{P} where two parts of \mathcal{P} are adjacent if and only if they are neither completely adjacent nor completely non-adjacent. This definition mirrors, in logical terms, the morphism trees of Bonnet et al. [7]. In particular, it applies the same idea that the radius of quantification decreases exponentially with the depth.

We prove a set of fundamental lemmas for manipulation of local types upon consecutive steps in a contraction sequence. These reflect the mechanics of morphism trees of [7], but by basing the argumentation essentially on Ehrenfeucht-Fraïssé games, the obtained explanation is arguably simpler and more insightful. Also, contrary to [5, 7], the introduced toolbox applies to tuples of vertices, and not only to single parts in the contraction sequence. This is important in our applications, which we discuss next.

We use the toolbox of local types to give the following algorithmic results on first-order expressible problems in graphs of bounded twin-width. The first one concerns the problem of *query answering*, and the second concerns the problem of *query enumeration*. In both theorems we assume that the graph is *specified through* a contraction sequence; this is explained in Section 2. For a tuple of parameters \mathbf{p} , the notation $\mathcal{O}_{\mathbf{p}}(\cdot)$ hides factors depending on \mathbf{p} .

► **Theorem 1.** *Suppose we are given an n -vertex graph G specified through a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of width d , and a first-order formula $\varphi(\mathbf{x})$, where \mathbf{x} is a set of variables. Then one can construct in time $\mathcal{O}_{d,\varphi}(n)$ a data structure that can answer the following queries in time $\mathcal{O}_{d,\varphi}(\log \log n)$: given $\mathbf{w} \in V(G)^{\mathbf{x}}$, decide whether $G \models \varphi(\mathbf{w})$.*

► **Theorem 2.** *Suppose we are given an n -vertex graph G specified through a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of width d , and a first-order formula $\varphi(\mathbf{x})$, where \mathbf{x} is a set of variables. Then after preprocessing in time $\mathcal{O}_{d,\varphi}(n)$, one can enumerate all tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ such that $G \models \varphi(\mathbf{w})$ with $\mathcal{O}_{d,\varphi}(1)$ delay.*

Note that in Theorem 1 there is a factor of the form $\mathcal{O}_{d,\varphi}(\log \log n)$ appearing in the query time. This is a consequence of using a data structure for orthogonal range queries of Chan [9] that supports queries in time $\mathcal{O}(\log \log n)$. As explained in [19], there is also a simple data structure for orthogonal range queries that, for any fixed $\varepsilon > 0$, offers query time $\mathcal{O}(1/\varepsilon)$ at the expense of increasing the construction time and the space complexity to $\mathcal{O}(n^{1+\varepsilon})$. By replacing the usage of the data structure of Chan with this simple data structure, we may obtain the same tradeoff in Theorem 1: The query time is reduced to $\mathcal{O}(1/\varepsilon)$, while the construction time and the space complexity is increased to $\mathcal{O}(n^{1+\varepsilon})$; this holds for any fixed $\varepsilon > 0$.

Theorems 1 and 2 mirror classic results on evaluation and enumeration of monadic second-order queries on trees [2, 10, 14] (which imply analogous results for graphs of bounded treewidth and of bounded cliquewidth), and of first-order queries on classes of bounded expansion [11, 15] and nowhere dense classes [22]. Therefore, we believe that the applications discussed above witness the robustness of the developed methodology.

As another application, we prove optimal bounds on *VC density* of set systems definable in graphs of bounded twin-width. Suppose $\varphi(\mathbf{x}, \mathbf{y})$ is a first-order formula with free variables partitioned into \mathbf{x} and \mathbf{y} . For a graph G and a subset of vertices A , we define the *Stone space*

$$S^\varphi(A) := \{ \{ \mathbf{a} \in A^{\mathbf{x}} \mid G \models \varphi(\mathbf{a}, \mathbf{b}) \} : \mathbf{b} \in V(G)^{\mathbf{y}} \}.$$

In other words, every tuple $\mathbf{b} \in V(G)^{\mathbf{y}}$ gives rise to the subset $\varphi(A, \mathbf{b}) \subseteq A^{\mathbf{x}}$ consisting of those tuples \mathbf{a} that together with \mathbf{b} satisfy φ . Then the Stone space $S^\varphi(A)$ consists of all sets $\varphi(A, \mathbf{b})$ that can be defined in this way. See for example [18] for a discussion of this notion and its applications.

In general graphs, $S^\varphi(A)$ can be as large as the whole powerset of $A^{\mathbf{x}}$. However, under various structural assumptions, it will be typically much smaller. For instance, suppose that the twin-width of G is bounded by a constant d . Then by combining the results of Bonnet et al. [7] with that of Baldwin and Shelah [3], one can argue that the VC dimension of $S^\varphi(A)$, regarded as a set system over universe $A^{\mathbf{x}}$, is bounded by a constant depending only on d and φ . Consequently, by the Sauer-Shelah Lemma [21, 23], the cardinality of $S^\varphi(A)$ is bounded polynomially in $|A|$. However, the degree of this polynomial bound, which is known as the *VC density* (studied for example in [1]), still depends on d and φ , and in a quite non-explicit way. We prove that in fact, there is a much sharper upper bound: the VC density is bounded by just the number of variables in \mathbf{y} .

► **Theorem 3.** *Let G be a graph of twin-width at most d , A be a subset of vertices of G , and $\varphi(\mathbf{x}, \mathbf{y})$ be a first-order formula. Then*

$$|S^\varphi(A)| \leq \mathcal{O}_{d,\varphi}(|A|^{|\mathbf{y}|}).$$

It is easy to see (see e.g. [18]) that even in edgeless graphs one cannot hope for a bound better than $|A|^{|\mathbf{y}|}$, and therefore the bound of Theorem 3 is asymptotically optimum.

Theorem 3 mirrors analogous results for monadic second-order formulas on classes of bounded treewidth or cliquewidth [16], and for first-order formulas on classes of bounded expansion and nowhere dense classes [18]. We remark that the case $|\mathbf{x}| = |\mathbf{y}| = 1$ follows from the fact that classes of bounded twin-width are closed under first-order transductions, combined with known linear upper bounds on the *neighborhood complexity*¹ in classes of bounded twin-width [6, 20]. Tackling multiple free variables requires a better understanding of types for tuples of vertices, which is exactly where our methodology of local types comes into play.

Organization. After preliminaries in Section 2, we present the framework of local types in Section 3. Then we prove Theorem 1 in Sections 4. Theorem 2 is proved in Section 5. Theorem 3 is deferred to the full version, due to space constraints. Easy proofs of statements marked with ♠ are also deferred to the full version.

2 Preliminaries

Graphs. In this paper we work with finite, undirected graphs and we use standard graph notation. By $|G|$ we denote the number of vertices of a graph G .

A pair of disjoint vertex subsets $A, B \subseteq V(G)$ is *complete* if every vertex of A is adjacent to every vertex of B , and *anti-complete* if there is no edge with one endpoint in A and the other one in B . The pair A, B is *pure* if it is complete or anti-complete, and *impure* otherwise.

¹ In our notation, bounds on neighborhood complexity exactly correspond to the case when $\mathbf{x} = \{x\}$, $\mathbf{y} = \{y\}$, and $\varphi(x, y)$ just checks that x and y are adjacent.

A *trigraph* is a structure in which there is a vertex set and every pair of distinct vertices is bound by exactly one of the following three symmetric relations: adjacency, non-adjacency, and impurity. Thus, graphs are trigraphs without impurities. Given a partition \mathcal{P} of the vertex set of a graph G , we define the *quotient trigraph* G/\mathcal{P} as the trigraph on vertex set \mathcal{P} where distinct $A, B \in \mathcal{P}$ are adjacent if the pair A, B is complete in G , non-adjacent if the pair is anti-complete, and impure towards each other if A, B is impure. For a trigraph H , its *impurity graph* $\text{Imp}(H)$ is the graph on vertex set H where two vertices $u, v \in V(H)$ are considered adjacent if they are impure towards each other in H .

Contraction sequences. Let G be a graph on n vertices. A *contraction sequence* for G is a sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of partitions of the vertex set of G such that:

- \mathcal{P}_1 is the partition into singletons;
- \mathcal{P}_n is the partition with one part;
- for each $t \in [n]$, $t > 1$, \mathcal{P}_t is obtained from \mathcal{P}_{t-1} by taking some two parts $A, B \in \mathcal{P}_{t-1}$ and *contracting* them: replacing them with a single part $A \cup B \in \mathcal{P}_t$.

Indices $t \in [n]$ will be called *times*. The *width* of the contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ is the maximum degree in graphs $\text{Imp}(G/\mathcal{P}_t)$, at all times $t \in [n]$. The *twin-width* of G is the minimum possible width of a contraction sequence of G .

If G is supplied with a total order \leq on $V(G)$, then a subset of vertices A is *convex* if it forms an interval in \leq , that is, if $a, b \in A$ then also $c \in A$ whenever $a \leq c \leq b$. A partition is convex if all its parts are convex, and a contraction sequence is convex if all its partitions are convex.

Additional notation for partitions and contraction sequences. Fix a graph G with a partition \mathcal{P} of its vertices. We will use the following notation.

Denote $G_{\mathcal{P}} := G/\mathcal{P}$ and $G_{\mathcal{P}}^{\text{imp}} := \text{Imp}(G_{\mathcal{P}})$. By $\text{dist}_{\mathcal{P}}(\cdot, \cdot)$ we denote the distance function in $G_{\mathcal{P}}^{\text{imp}}$: for $A, B \in \mathcal{P}$, $\text{dist}_{\mathcal{P}}(A, B)$ is the minimum length of a path in $G_{\mathcal{P}}^{\text{imp}}$ connecting A and B , and $+\infty$ if there is no such path. We extend this notation to subsets, or tuples of elements of \mathcal{P} : $\text{dist}_{\mathcal{P}}(X, Y)$ denotes the minimum, over all A occurring in X and B occurring in Y , of $\text{dist}_{\mathcal{P}}(A, B)$.

For a set of parts $\mathcal{F} \subseteq \mathcal{P}$ and a radius parameter $r \in \mathbb{N}$, the *r-neighborhood* of \mathcal{F} , denoted $\text{Vicinity}_{\mathcal{P}}^r(\mathcal{F})$, is the trigraph induced in $G_{\mathcal{P}}$ by all parts at distance at most r from any part belonging to \mathcal{F} , that is

$$\text{Vicinity}_{\mathcal{P}}^r(\mathcal{F}) := G_{\mathcal{P}}[\{A \in \mathcal{P} \mid \text{dist}_{\mathcal{P}}(A, \mathcal{F}) \leq r\}].$$

We may use notation $\text{Vicinity}_{\mathcal{P}}^r(\cdot)$ for single parts or tuples of parts with the obvious meaning.

For brevity, whenever a graph G and its contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ are clear from the context, we fix the following notation. First, in all the notation defined above, concerning partitions, we write s in the subscript instead of \mathcal{P}_s . So for instance we write G_s to denote $G_{\mathcal{P}_s}$, and G_s^{imp} to denote $G_{\mathcal{P}_s}^{\text{imp}}$, and $\text{dist}_s(\cdot, \cdot)$ to denote $\text{dist}_{\mathcal{P}_s}(\cdot, \cdot)$, etc.

Fix a finite set of variables \mathbf{x} . For a pair of times $s, t \in [n]$, $s \leq t$, and a tuple of parts $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$, we define the tuple $\mathbf{u}(s \rightarrow t) \in \mathcal{P}_t^{\mathbf{x}}$ as follows: for each $y \in \mathbf{x}$, $\mathbf{u}(s \rightarrow t)(y)$ is the unique part of \mathcal{P}_t that contains $\mathbf{u}(y)$. For a tuple $\mathbf{u} \in V(G)^{\mathbf{x}}$ of vertices and $s \in [n]$, by $\mathbf{u}(s) \in \mathcal{P}_s^{\mathbf{x}}$ we denote the unique tuple whose y -component, for $y \in \mathbf{x}$, is the part of \mathcal{P}_s containing $\mathbf{u}(y)$.

For $s \in [n-1]$, by B_{s+1} we denote the unique part of \mathcal{P}_{s+1} that is the union of two parts in \mathcal{P}_s . For a parameter $r \in \mathbb{N}$, we define the r -relevant region in G_s as follows:

$$\text{Relevant}_s^r := G_s[\{C \in \mathcal{P}_s \mid C \subseteq B_{s+1}, \text{ or } C \in \mathcal{P}_{s+1} \text{ and } \text{dist}_{s+1}(C, B_{s+1}) \leq r\}].$$

In other words, Relevant_s^r is the trigraph induced in G_s by the two parts of \mathcal{P}_s that get contracted into B_{s+1} and all parts of \mathcal{P}_s that stay intact in \mathcal{P}_{s+1} and are at distance at most r from B_{s+1} in G_{s+1}^{imp} .

Note that we have $|\text{Relevant}_s^p| \leq \mathcal{O}_{d,p}(1)$ for all $s \in [n-1]$. The next lemma shows that the p -relevant regions can be computed efficiently.

► **Lemma 4** (♠). *Suppose a graph G on vertex set $[n]$ is provided through a convex contraction sequence \mathcal{P} of width d . Then for a given $p \in \mathbb{N}$, one can in time $\mathcal{O}_{d,p}(n)$ compute the trigraphs Relevant_s^p for all $s \in [n-1]$.*

Specifying a graph through its contraction sequence. In all algorithmic statements we will assume that a graph is given by specifying its contraction sequence together with some auxiliary information encoding the edge relation. We now make this precise.

Let $\mathcal{P}_1, \dots, \mathcal{P}_n$ be a contraction sequence of a graph G . We assume that every part participating in the partitions $\mathcal{P}_1, \dots, \mathcal{P}_n$ (that is, every element of the union $\mathcal{P}_1 \cup \dots \cup \mathcal{P}_n$, where each \mathcal{P}_i is viewed as a set of sets of vertices) is specified through a unique identifier taking a single machine word. For \mathcal{P}_1 , the identifiers of (singleton) parts coincide with identifiers of the corresponding vertices. Then sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ is represented by providing the following information for every time $s \in [n]$, $s > 1$:

- The identifiers of the two parts $A, A' \in \mathcal{P}_{s-1}$ that get contracted at time s , and the identifier of the obtained part $B = A \cup A' \in \mathcal{P}_s$.
- A list of identifiers of parts $C \in \mathcal{P}_s$ such that the pair B, C is impure in G (that is, the impurities incident to B in $\text{Imp}(G/\mathcal{P}_s)$).
- For each part C on the list above, the relation (completeness, anti-completeness, or impurity) between C and A and between C and B in G/\mathcal{P}_{s-1} .

It is easy to see that this representation uniquely defines the graph G . Since the representation takes $\mathcal{O}_d(1)$ machine words at any time s , we can thus represent an n -vertex graph of twin-width d using $\mathcal{O}_d(n)$ machine words.

We now show that, for a graph given through a contraction sequence, one can reindex the vertex set using integers from $[n]$ so that the contraction sequence becomes convex.

► **Lemma 5** (♠). *Suppose a graph G is given by specifying a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of width d . Then one can in time $\mathcal{O}_d(n)$ compute a bijection $\eta: V(G) \rightarrow [n]$ such that mapping G and $\mathcal{P}_1, \dots, \mathcal{P}_n$ through η yields an isomorphic graph G' on vertex set $[n]$ and its contraction sequence $\mathcal{P}'_1, \dots, \mathcal{P}'_n$ such that $\mathcal{P}'_1, \dots, \mathcal{P}'_n$ is convex in the natural order on integers in $[n]$.*

Note that if a graph is reindexed using Lemma 5, then every part participating in the resulting contraction sequence is an interval in $[n]$. Hence, as the identifier of a part we can simply use a pair of vertices – the left endpoint and the right endpoint – and such identifiers can be computed in time $\mathcal{O}_d(n)$ by scanning the contraction sequence. We will therefore assume that contraction sequences are convex with respect to a fixed ordering of the vertices, and the (convex) parts are identified by their endpoints.

First-order logic. We fix a countable set of variables, together with its enumeration. If Ω is a set and \mathbf{x} is a finite set of variables, then an \mathbf{x} -tuple with entries in Ω is a function from \mathbf{x} to Ω . Tuples are by convention denoted with boldface small letters, e.g. \mathbf{u} or \mathbf{v} . The set of all \mathbf{x} -tuples with entries in Ω is denoted by $\Omega^{\mathbf{x}}$. When $\mathbf{a} \in \Omega^{\mathbf{x}}$ is an \mathbf{x} -tuple and $b \in \Omega$, then by $\mathbf{a}b$ we denote the $(\mathbf{x} \cup \{y\})$ -tuple that extends \mathbf{a} and maps the first variable (according to the fixed enumeration of all variables) y not in \mathbf{x} , to b .

We consider standard first-order logic on graphs by modeling them as relational structures where the universe is the vertex set and there is a single binary predicate signifying adjacency. For a graph G , a formula $\varphi(\mathbf{x})$, where \mathbf{x} is the set of free variables of φ , and a tuple of vertices $\mathbf{w} \in V(G)^{\mathbf{x}}$, we write $G \models \varphi(\mathbf{w})$, or $G, \mathbf{w} \models \varphi(\mathbf{x})$, to denote that \mathbf{w} satisfies $\varphi(\mathbf{x})$ in G . We sometimes consider formulas with an explicitly partitioned set of free variables, e.g., $\varphi(\mathbf{x}, \mathbf{y})$. Sentences are formulas with no free variables.

Logical types. While the usual definition of a logical type of quantifier rank k of a tuple \mathbf{a} of vertices of G is the set of all formulas $\varphi(\mathbf{x})$ such that $G \models \varphi(\mathbf{a})$, we will rely on a definition which is more suitable for our purposes and is well known to be equivalent, by the result of Ehrenfeucht and Fraïssé (see for example [12]).

Let \mathbf{x} be a finite set of variables. An *atomic type with variables \mathbf{x}* is a maximal consistent set S of formulas of the form $x = y$, $x \neq y$, $E(x, y)$, $\neg E(x, y)$, where $x, y \in \mathbf{x}$. Here by *consistent* we mean that there is some graph G and a tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$ that satisfies all formulas occurring in the atomic type (this is decidable, as it is sufficient to consider graphs G with $|G| \leq |\mathbf{x}|$).

For $\mathbf{a} \in V(G)^{\mathbf{x}}$, the *atomic type of \mathbf{a} in G* is the atomic type with variables \mathbf{x} which consists of all formulas of the form $E(x, y)$ or $x = y$, where $x, y \in \mathbf{x}$, such that $G, \mathbf{a} \models x = y$ or $G, \mathbf{a} \models E(x, y)$.

► **Definition 6.** Let G be a graph \mathbf{x} a finite set of variables and $k \in \mathbb{N}$. For every $\mathbf{a} \in V(G)^{\mathbf{x}}$ we define its type of quantifier rank k , denoted $\text{tp}^k(\mathbf{a})$, as follows.

- If $k = 0$, then $\text{tp}^0(\mathbf{a})$ is the atomic type of \mathbf{a} in G .
- If $k > 0$, then $\text{tp}^k(\mathbf{a}) = \{\text{tp}^{k-1}(\mathbf{a}b) \mid b \in V(G)\}$.

For $k \geq 1$ we also set $\text{tp}^k(G) = \{\text{tp}^{k-1}(a) \mid a \in V(G)\}$.

This definition is usually intuitively explained in terms of Ehrenfeucht-Fraïssé games. Namely, two \mathbf{x} -tuples \mathbf{a} and \mathbf{b} of vertices of two graphs G and H , respectively, have equal types of quantifier rank k if and only if Duplicator wins the k -round game on the graphs G and H , where the initial pebbles in G and H are placed on the vertices occurring in \mathbf{a} and in \mathbf{b} , respectively. Indeed, suppose $\text{tp}^k(\mathbf{a}) = \text{tp}^k(\mathbf{b})$, where $k > 0$, and that Spoiler places a pebble on a vertex c of G . Then, since $\text{tp}^{k-1}(\mathbf{a}c) \in \text{tp}^k(\mathbf{a})$ by definition and $\text{tp}^k(\mathbf{a}) = \text{tp}^k(\mathbf{b})$, we have that there is some $d \in \text{tp}^k(\mathbf{b})$ such that $\text{tp}^{k-1}(\mathbf{b}d) \in \text{tp}^k(\mathbf{b})$. Then Duplicator responds by placing the pebble on the vertex d , and we have that $\text{tp}^{k-1}(\mathbf{a}c) = \text{tp}^{k-1}(\mathbf{b}d)$ so, by inductive assumption, Duplicator wins in the $k - 1$ round game from the current configuration, which shows that Duplicator has a winning strategy in the k round game starting from \mathbf{a} and \mathbf{b} . The implication in the other direction proceeds similarly.

As is well known, the set of types of \mathbf{x} -tuples of quantifier rank k that are realized by some tuple \mathbf{a} , in some graph, is non-computable, even though this set has size bounded in terms of \mathbf{x} and k . To overcome this problem, the usual solution is to define the set of abstract types (that may not be realized as actual types), which is computable from \mathbf{x} and k , has bounded size, and contains all types that may arise. This is done as follows.

Define $\text{Types}_{\mathbf{x}}^0$ as the set of all atomic types over \mathbf{x} and $\text{Types}_{\mathbf{x}}^k := \{M \mid M \subseteq \text{Types}_{\mathbf{x}y}^{k-1}\}$. Note that for any G and any $\mathbf{a} \in V(G)^{\mathbf{x}}$ it holds that $\text{tp}^k(\mathbf{a}) \in \text{Types}_{\mathbf{x}}^k$, but $\text{Types}_{\mathbf{x}}^k$ can also contain objects which are not realized by any tuple of vertices \mathbf{a} of any graph.

For a graph G we set $\text{Types}_{\mathbf{x}}^k(G) := \{\text{tp}^k(\mathbf{a}) \mid \mathbf{a} \in V(G)^{\mathbf{x}}\}$. Note that we have $\text{tp}^k(G) = \text{Types}_{\mathbf{x}}^{k-1}(G)$.

The following is well known and follows from the fact that our definition of types is equivalent to the usual definition of types using formulas.

► **Proposition 7.** *Let G be a graph, \mathbf{x} a set of variables and $k \in \mathbb{N}$.*

- $|\text{Types}_{\mathbf{x}}^k(G)| = \mathcal{O}_{k,\mathbf{x}}(1)$,
- For any $\mathbf{a} \in V(G)^{\mathbf{x}}$ and any first-order formula $\varphi(\mathbf{x})$ of quantifier rank at most k one can determine whether $G \models \varphi(\mathbf{a})$ from $\text{tp}^k(\mathbf{a})$ in time $\mathcal{O}_{k,\mathbf{x}}(1)$.
- For any first-order sentence φ of quantifier rank at most k one can determine whether $G \models \varphi$ from $\text{tp}^k(G)$ in time $\mathcal{O}_k(1)$.

3 Local types

In this section we define local types of quantifier rank k for partitioned graphs, or *local k -types* for short. They provide a framework for the results proved in the rest of the paper. The key lemmas are Lemma 14 and Lemma 16 and their corollaries Lemma 15 and Lemma 17.

3.1 Local types for partitioned graphs

Let G be a graph and \mathcal{P} be a partition of its vertex set, and let \mathbf{x} be a set of variables. For an \mathbf{x} -tuple $\mathbf{a} \in V(G)$ write $\mathbf{a}(\mathcal{P})$ for the \mathbf{x} -tuple $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}$ such that $\mathbf{u}(x)$ is the part containing $\mathbf{a}(x)$, for all $x \in \mathbf{x}$.

► **Definition 8.** *Let G be a graph, \mathcal{P} be a partition of its vertex set, \mathbf{x} a nonempty set of variables, and $k \in \mathbb{N}$. For any $\mathbf{a} \in V(G)^{\mathbf{x}}$ we define the local k -type of \mathbf{a} , denoted $\text{ltp}_{\mathcal{P}}^k(\mathbf{a})$, as follows:*

- $\text{ltp}_{\mathcal{P}}^0(\mathbf{a})$ is the atomic type of \mathbf{a} together with the \mathbf{x} -tuple $\mathbf{a}(\mathcal{P}) \in \mathcal{P}^{\mathbf{x}}$ of parts of \mathcal{P} corresponding to \mathbf{a} ,
- for $k > 0$, let $\text{ltp}_{\mathcal{P}}^k(\mathbf{a}) = \{\text{tp}_{\mathcal{P}}^{k-1}(\mathbf{a}\mathbf{b}) \mid \mathbf{b} \in w \text{ for some } w \in \mathcal{P} \text{ with } \text{dist}_{\mathcal{P}}(\mathbf{a}(\mathcal{P}), w) \leq 2^{k-1}\}$.

As with usual types of quantifier-rank k defined in the previous section, it is often convenient to think about equality of local types in terms of games. We now briefly describe the corresponding variant of Ehrenfeucht-Fraïssé games. This game will be played on a single graph G with a fixed partition \mathcal{P} of its vertex set (one can also imagine it being played on two copies of the same graph with the same partition). The starting position of the game is determined by two \mathbf{x} -tuples \mathbf{a} and \mathbf{b} of vertices of G (where \mathbf{x} is nonempty) such that for every $y \in \mathbf{x}$ we have that $\mathbf{a}(y)$ is in the same part of \mathcal{P} as $\mathbf{b}(y)$. The game is played for k rounds as the usual Ehrenfeucht-Fraïssé game with the following extra restrictions on the moves of the players: (1) In the i th round, Spoiler picks one of the tuples \mathbf{a} and \mathbf{b} , and he will then proceed to extending it. Suppose that he picks \mathbf{a} , the other case being symmetric. Spoiler then picks a vertex a in any part $P \in \mathcal{P}$ such that $\text{dist}_{\mathcal{P}}(P, Q) \leq 2^{k-1}$, where Q is some part containing a vertex of \mathbf{a} . He then appends a to \mathbf{a} to form the tuple $\mathbf{a}a$. (2) Duplicator replies by picking a vertex b in the same part P and extending the other tuple \mathbf{b} to $\mathbf{b}b$. The game then continues to the next round, with $\mathbf{a}a$ and $\mathbf{b}b$ forming the new position. Duplicator wins after k rounds if the two tuples have equal atomic types.

It is not difficult to see that Duplicator wins the k -round game described above, starting from the configuration \mathbf{a} and \mathbf{b} , if and only if $\text{ltp}_{\mathcal{P}}^k(\mathbf{a}) = \text{ltp}_{\mathcal{P}}^k(\mathbf{b})$. This is made formal in the following proposition, whose proof is an immediate consequence of Definition 8.

► **Proposition 9.** *Let G be a graph and \mathcal{P} be a partition of the vertices of G , and let \mathbf{x} a tuple of variables and $k \in \mathbb{N}$. Then the following holds for any $\mathbf{a}, \mathbf{b} \in V(G)^{\mathbf{x}}$:*

- $\text{ltp}_{\mathcal{P}}^0(\mathbf{a}) = \text{ltp}_{\mathcal{P}}^0(\mathbf{b})$ if and only if the atomic types of \mathbf{a} and \mathbf{b} are the same, and $\mathbf{a}\langle\mathcal{P}\rangle = \mathbf{b}\langle\mathcal{P}\rangle$;
- If $k > 0$ then $\text{ltp}_{\mathcal{P}}^k(\mathbf{a}) = \text{ltp}_{\mathcal{P}}^k(\mathbf{b})$ if and only if for any $P \in \mathcal{P}$ with $\text{dist}_{\mathcal{P}}(\mathbf{a}\langle P \rangle, P) \leq 2^{k-1}$ the following holds: for any $c \in P$ there exists $c' \in P$ such that $\text{ltp}_{\mathcal{P}}^{k-1}(\mathbf{a}c) = \text{ltp}_{\mathcal{P}}^{k-1}(\mathbf{b}c')$, and conversely, for any $c' \in P$ there exists $c \in P$ such that $\text{ltp}_{\mathcal{P}}^{k-1}(\mathbf{a}c) = \text{ltp}_{\mathcal{P}}^{k-1}(\mathbf{b}c')$.

We will also need to have an abstract set containing all types which could potentially occur for any $k \in \mathbb{N}$ and $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}$. Note that this includes also types which are not realized in G (or even in any graph).

► **Definition 10.** *Let \mathbf{x} be a nonempty set of variables and $k \in \mathbb{N}$. Fix a graph G together with a vertex-partition \mathcal{P} . For $\mathbf{u} \in \mathcal{P}^{\mathbf{x}}$ we define $\text{Types}_{\mathbf{u}, \mathcal{P}}^0 := \{(\alpha, \mathbf{u}) \mid \alpha \text{ is an atomic type with variables } \mathbf{x}\}$. For $k > 0$ let M be the set of all parts w of \mathcal{P} with $\text{dist}_{\mathcal{P}}(\mathbf{u}, w) \leq 2^{k-1}$ and let $M' := \bigcup_{w \in M} \text{Types}_{\mathbf{u}w, \mathcal{P}}^{k-1}$. We then define*

$$\text{Types}_{\mathbf{u}, \mathcal{P}}^k := \{S \mid S \subseteq M'\}.$$

Define also $\text{Types}_{\mathbf{u}, \mathcal{P}}^k(G) := \{\text{ltp}_{\mathcal{P}}^k(\mathbf{a}) \mid \mathbf{a} \in V(G)^{\mathbf{x}}, \mathbf{u} = \mathbf{a}\langle\mathcal{P}\rangle\}$.

Then $\text{Types}_{\mathbf{u}, \mathcal{P}}(G)$ is the set of all local k -types realized in \mathbf{u} , and is a subset of $\text{Types}_{\mathbf{u}, \mathcal{P}}^k$.

3.2 Properties of local types

In this section we establish the properties of local k -types used in the rest of the paper.

In the rest of this paper, we assume that we have fixed a graph G and a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of G . We write $\text{ltp}_s^k(\cdot)$ to denote $\text{ltp}_{\mathcal{P}_s}^k(\cdot)$, $\text{Types}_{\mathbf{u}, s}(\cdot)$ to denote $\text{Types}_{\mathbf{u}, \mathcal{P}_s}(\cdot)$, and $\text{dist}_s(\cdot, \cdot)$ to denote $\text{dist}_{\mathcal{P}_s}(\cdot, \cdot)$.

The following two lemmas establish some basic properties of local k -types. The proof of Lemma 11 follows immediately from the definition.

► **Lemma 11.** *The following holds at any time $s \in [n]$ and $k \geq 1$.*

- If $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{b})$, then $\text{ltp}_s^{k-1}(\mathbf{a}) = \text{ltp}_s^{k-1}(\mathbf{b})$.
- If $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{b})$, then $\mathbf{a}\langle s \rangle = \mathbf{b}\langle s \rangle$.

► **Lemma 12 (♠).** *Let $s \in [n]$ be a time and \mathbf{x} a tuple of variables, and $k \geq 0$. Then $|\text{Types}_{\mathbf{u}, s}^k| \leq \mathcal{O}_{d, k, \mathbf{x}}(1)$, for all $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$. Moreover, given vicinity $\text{Vicinity}_s^{2^k}(\mathbf{u})$, one can compute $\text{Types}_{\mathbf{u}, s}^k$ in time $\mathcal{O}_{d, k, \mathbf{x}}(1)$.*

The following lemma relates local types for partitioned graphs to usual first-order types, as defined in the preliminaries.

► **Lemma 13 (♠).** *Let \mathbf{x} be a tuple of variables and $\mathbf{a} \in V(G)^{\mathbf{x}}$. One can compute $\text{tp}^k(\mathbf{a})$ from $\text{ltp}_n^k(\mathbf{a})$ in time $\mathcal{O}_{k, d, \mathbf{x}}(1)$.*

The next lemma is a version of compositionality of local types and plays a key role in computing local types.

123:10 Twin-Width and Types

► **Lemma 14.** *Fix two disjoint sets of variables \mathbf{x} and \mathbf{y} . Let $\mathbf{a}, \mathbf{a}' \in V^{\mathbf{x}}$ and $\mathbf{b}, \mathbf{b}' \in V^{\mathbf{y}}$ be such that $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{a}')$ and $\text{ltp}_s^k(\mathbf{b}) = \text{ltp}_s^k(\mathbf{b}')$. Let $\mathbf{u} = \mathbf{a}\langle s \rangle$ and $\mathbf{v} = \mathbf{b}\langle s \rangle$ and assume that $\text{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$. Then $\text{ltp}_s^k(\mathbf{ab}) = \text{ltp}_s^k(\mathbf{a}'\mathbf{b}')$.*

Proof. We prove the statement by induction on k . For $k = 0$, to prove that $\text{ltp}_s^0(\mathbf{ab}) = \text{ltp}_s^0(\mathbf{a}'\mathbf{b}')$, we have to show that the atomic types of \mathbf{ab} and $\mathbf{a}'\mathbf{b}'$ are the same. Fix an atomic formula $\varphi(x, y)$, with $x, y \in \mathbf{x} \cup \mathbf{y}$. We show that $G, \mathbf{ab} \models \varphi(x, y)$ if and only if $G, \mathbf{a}'\mathbf{b}' \models \varphi(x, y)$. If x and y both belong to \mathbf{x} then the conclusion follows by assumption that $\text{ltp}_s^0(\mathbf{a}) = \text{ltp}_s^0(\mathbf{a}')$. The same holds if x and y both belong to \mathbf{y} .

So, by symmetry, it is enough to consider the case when $x \in \mathbf{x}$ and $y \in \mathbf{y}$. Since by our assumption $\text{dist}_s(\mathbf{u}, \mathbf{v}) > 2^0 = 1$, any part of \mathbf{u} is pure to any part \mathbf{v} , and so in particular the part $\mathbf{a}(x)\langle s \rangle$ is pure towards $\mathbf{b}(y)\langle s \rangle$. Because $\mathbf{a}(x), \mathbf{a}'(x) \in \mathbf{a}(x)\langle s \rangle$ and $\mathbf{b}(y), \mathbf{b}'(y) \in \mathbf{b}(y)\langle s \rangle$, this implies that $G, \mathbf{ab} \models \varphi(x, y)$ if and only if $G, \mathbf{a}'\mathbf{b}' \models \varphi(x, y)$, as required.

For $k > 0$, let c be a vertex in a part $w = c\langle s \rangle$ such that $\text{dist}_s(\mathbf{uv}, w) \leq 2^{k-1}$. Our task is to show that there exists $c' \in w$ such that $\text{ltp}_s^{k-1}(\mathbf{abc}) = \text{ltp}_s^{k-1}(\mathbf{a}'\mathbf{b}'c')$. Since $\text{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$, exactly one of $\text{dist}_s(\mathbf{u}, w) \leq 2^{k-1}$ and $\text{dist}_s(\mathbf{v}, w) \leq 2^{k-1}$ has to hold; without loss of generality assume that $\text{dist}_s(\mathbf{u}, w) \leq 2^{k-1}$ holds. Since $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{a}')$, there exists $c' \in w$ such that $\text{ltp}_s^{k-1}(\mathbf{ac}) = \text{ltp}_s^{k-1}(\mathbf{a}'c')$. Because $\text{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$ and $\text{dist}_s(\mathbf{u}, w) \leq 2^{k-1}$, we have $\text{dist}_s(\mathbf{uw}, \mathbf{v}) > 2^{k-1}$, so we can apply the induction hypothesis to $\mathbf{ac}, \mathbf{a}'c'$ and \mathbf{b}, \mathbf{b}' , which yields that $\text{ltp}_s^{k-1}(\mathbf{abc}) = \text{ltp}_s^{k-1}(\mathbf{a}'\mathbf{b}'c')$, as desired. ◀

The following lemma follows directly from Lemma 14, except for the part about efficient computation.

► **Lemma 15.** *Let $s \in [n]$ be a time and \mathbf{x} and \mathbf{y} are disjoint sets of variables. Suppose $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ and $\mathbf{v} \in \mathcal{P}_s^{\mathbf{y}}$ are tuples of parts such that $\text{dist}_s(\mathbf{u}, \mathbf{v}) > 2^k$. Then there is a function $f: \text{Types}_{\mathbf{u},s}^k \times \text{Types}_{\mathbf{v},s}^k \rightarrow \text{Types}_{\mathbf{uv},s}^k$ such that for every pair of tuples $\mathbf{a} \in V^{\mathbf{x}}$ and $\mathbf{b} \in V^{\mathbf{y}}$ satisfying $\mathbf{u} = \mathbf{a}\langle s \rangle$ and $\mathbf{v} = \mathbf{b}\langle s \rangle$, we have*

$$\text{ltp}_s^k(\mathbf{ab}) = f(\text{ltp}_s^k(\mathbf{a}), \text{ltp}_s^k(\mathbf{b})).$$

Moreover, given $k, \mathbf{u}, \mathbf{v}$, and the vicinity $\text{Vicinity}_s^{2^k}(\mathbf{uv})$, one can compute f in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$.

Regarding the computation of function f in the above lemma, by “computing f in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ ” we do not mean just evaluating f on any given input in desired time, but constructing the whole input-output table for f . The reason why this can be computed from $k, \mathbf{u}, \mathbf{v}$ and $\text{Vicinity}_s^{2^k}(\mathbf{uv})$ in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ is that the input and output sets have size bounded by $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ and the proof in Lemma 14 uses only information from $\text{Vicinity}_s^{2^k}(\mathbf{uv})$. A concrete approach to implementing this computation similar to that of [7] can be found in the full version.

The next lemma will allow us to determine how the k -type of a tuple \mathbf{a} develops over time.

► **Lemma 16.** *Let $s \in [n]$ be a time and let $\mathbf{a} \in V^{\mathbf{x}}, \mathbf{a}' \in V^{\mathbf{x}}$ be two tuples of vertices such that $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{a}')$. Then $\text{ltp}_{s+1}^k(\mathbf{a}) = \text{ltp}_{s+1}^k(\mathbf{a}')$.*

Proof. By induction on k . For $k = 0$ note that $\text{ltp}_s^0(\mathbf{a}) = \text{ltp}_s^0(\mathbf{a}')$ implies that atomic types of \mathbf{a} and \mathbf{a}' are the same and $\mathbf{a}\langle s \rangle = \mathbf{a}'\langle s \rangle$. It is easily seen that then also $\mathbf{a}\langle s+1 \rangle = \mathbf{a}'\langle s+1 \rangle$, as desired.

For $k > 0$, let $\mathbf{u} = \mathbf{a}\langle s+1 \rangle = \mathbf{a}'\langle s+1 \rangle$. We need to show that for any $w \in \mathcal{P}_{s+1}$ with $\text{dist}_{s+1}(\mathbf{u}, w) \leq 2^{k-1}$ and any $b \in w$ there is $b' \in w$ such that $\text{ltp}_{s+1}^{k-1}(\mathbf{a}b) = \text{ltp}_{s+1}^{k-1}(\mathbf{a}'b')$, and symmetrically, that for any $b' \in w$ there is $b \in w$ such that $\text{ltp}_{s+1}^{k-1}(\mathbf{a}b) = \text{ltp}_{s+1}^{k-1}(\mathbf{a}'b')$. We focus on the first option; the proof of the second one is analogous. Let $v = b\langle s \rangle$. We distinguish two possibilities:

- $\text{dist}_s(\mathbf{u}, v) \leq 2^{k-1}$: In this case, since $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{a}')$, there exists $b' \in v$ such that $\text{ltp}_s^{k-1}(\mathbf{a}b) = \text{ltp}_s^{k-1}(\mathbf{a}'b')$. Then by induction hypothesis it follows that $\text{ltp}_{s+1}^{k-1}(\mathbf{a}b) = \text{ltp}_{s+1}^{k-1}(\mathbf{a}'b')$, as desired.
- $\text{dist}_s(\mathbf{u}, v) > 2^{k-1}$: In this case we note that $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_s^k(\mathbf{a}')$ implies that $\text{ltp}_s^{k-1}(\mathbf{a}) = \text{ltp}_s^{k-1}(\mathbf{a}')$, and we set $b' := b$. We can now apply Lemma 14 to \mathbf{a} , \mathbf{a}' and b , b' to see that $\text{ltp}_s^{k-1}(\mathbf{a}b) = \text{ltp}_s^{k-1}(\mathbf{a}'b')$, and by induction hypothesis it follows that $\text{ltp}_{s+1}^{k-1}(\mathbf{a}b) = \text{ltp}_{s+1}^{k-1}(\mathbf{a}'b')$, as desired. ◀

Lemma 16 implies that there exists a function which maps $\text{ltp}_s^k(\mathbf{a})$ to $\text{ltp}_{s+1}^k(\mathbf{a})$, and by induction we get the following lemma.

► **Lemma 17.** *Let $s, t \in [n]$ be times with $s \leq t$. Suppose $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ and let $\mathbf{v} = \mathbf{u}\langle s \rightarrow t \rangle$. Then there exists a function $f: \text{Types}_{\mathbf{u},s}^k \rightarrow \text{Types}_{\mathbf{v},t}^k$ such that for every tuple $\mathbf{a} \in V^{\mathbf{x}}$ satisfying $\mathbf{u} = \mathbf{a}\langle s \rangle$, we have*

$$\text{ltp}_t^k(\mathbf{a}) = f(\text{ltp}_s^k(\mathbf{a})).$$

Moreover, if $t = s+1$, then given k , \mathbf{u} , \mathbf{v} and the relevant region $\text{Relevant}_s^{2^k(|\mathbf{x}|+1)}$, one can compute f in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$, provided that for every $y \in \mathbf{x}$ we have that $\mathbf{v}(y) \in \text{Relevant}_s^{2^k|\mathbf{x}|}$.

As in the case of Lemma 15, the whole input-output table of function f can be computed in time $\mathcal{O}_{d,k,\mathbf{x},\mathbf{y}}(1)$ from k , \mathbf{u} , \mathbf{v} and $\text{Relevant}_s^{2^k}$, since the proof of Lemma 16 uses only information from $\text{Relevant}_s^{2^k(|\mathbf{x}|+1)}$. Again, a concrete approach to implementing this computation similar to that of [7] can be found in the full version.

We will also use the fact that when going from time s to $s+1$ the local k -types of tuples in parts which are not in the trigraph $\text{Relevant}_s^{2^k}$ are not affected.

► **Lemma 18 (♠).** *Let \mathbf{x} be a finite set of variables, $s \in [n]$ a time, $k \in \mathbb{N}$ and let $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ be such for every $y \in \mathbf{x}$ it holds that $\mathbf{u}(y) \notin V(\text{Relevant}_s^{2^k})$. Then $\mathbf{u}\langle s \rightarrow s+1 \rangle = \mathbf{u}$, and for every \mathbf{a} with $\mathbf{u} = \mathbf{a}\langle s \rangle$ we have that $\text{ltp}_s^k(\mathbf{a}) = \text{ltp}_{s+1}^k(\mathbf{a})$. In particular, $\text{Types}_{\mathbf{u},s+1}^k(G) = \text{Types}_{\mathbf{u},s}^k(G)$.*

Model checking on graphs of bounded twin-width. With the machinery from the previous subsection we can now reprove the result of [7] that the first-order model checking problem on any class \mathcal{C} of graphs of twin-width at most d is solvable in time $\mathcal{O}_{d,\varphi}(n)$, provided that the contraction sequence of the input graph G is provided together with G .

► **Theorem 19.** *Let G be a graph on n vertices represented through its contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of width d . Then for any sentence φ of quantifier rank q one can decide whether $G \models \varphi$ in time $\mathcal{O}_{d,\varphi}(n)$.*

Proof. Let q be the quantifier rank of φ and set $k := q-1$ and $r := 2^k$. We will show how to compute the set $\text{Types}_{\mathbf{x}}^k(G)$ in desired time, and since $\text{tp}^q(G) = \text{Types}_{\mathbf{x}}^{q-1}(G)$, the result will follow by Proposition 7.

123:12 Twin-Width and Types

As a preprocessing step, the algorithm computes in time $\mathcal{O}_{d,k}(n)$ the trigraphs Relevant_s^r for all $s \in [n-1]$; this can be done due to Lemma 4. For the rest of the proof, let us for any time $s \in [n]$ denote by T_s the set of all sets of realized types at time s , i.e. $T_s := \{\text{Types}_{w,s}^k(G) \mid w \in \mathcal{P}_s\}$.

The algorithm first computes T_1 by computing $\text{Types}_{w,1}^k(G)$ for each $w \in \mathcal{P}_1$; since each such part w contains exactly one vertex, this can be done in time $\mathcal{O}_k(1)$ for any w , and so this takes time $\mathcal{O}_k(n)$ in total. From this point on the algorithm will proceed through times 2 to n and for every time s it will compute T_s from T_{s-1} . By Lemma 18, any part w of \mathcal{P}_{s-1} which is not in $\text{Relevant}_{s-1}^{2^k}$ is the same in \mathcal{P}_s as in \mathcal{P}_{s-1} and we have that $\text{Types}_{w,s-1}^k(G) = \text{Types}_{w,s}^k(G)$, which means that the computation only needs to be performed on parts from $\text{Relevant}_{s-1}^{2^k}$. We distinguish the following two possibilities:

- If v, w are the two parts of \mathcal{P}_{s-1} which get contracted into a part $u \in \mathcal{P}_s$, then the algorithm applies the function from Lemma 17 to all members of $\text{Types}_{v,s-1}^k(G)$ and $\text{Types}_{w,s-1}^k(G)$ and collects the results into $\text{Types}_{u,s}^k(G)$.
- If w is any other part in $\text{Relevant}_{s-1}^{2^k}$, then the algorithm applies the function from Lemma 17 to all members of $\text{Types}_{w,s-1}^k$ and collects the results into $\text{Types}_{w,s}^k(G)$.

In each of the above cases the computation can be done in time $\mathcal{O}_{d,k}(1)$, since each application of the function from Lemma 17 can be done in time $\mathcal{O}_{d,k,1}(1)$ and by Lemma 12 we have that $|\text{Types}_{w,s-1}^k(G)| \leq \mathcal{O}_{d,k}(1)$. Moreover, since $|\text{Relevant}_{s-1}^{2^k}| \leq \mathcal{O}_{d,k}(1)$, the computation of T_s from T_{s-1} can be done in time $\mathcal{O}_{d,k}(1)$. There are $n-1$ steps to obtain T_n and so the whole computation takes time $\mathcal{O}_{d,k}(n)$. Now T_n contains only $\text{Types}_{w,n}^k(G)$ where w is the only part of \mathcal{P}_n . By Lemma 13, from each local k -type in $\text{Types}_{w,n}^k(G)$ one can compute the corresponding k -type from $\text{Types}_x^k(G)$ in time $\mathcal{O}_k(1)$. This finishes the proof. ◀

4 Query answering

In this section we prove Theorem 1. For the remainder of this section let us fix a graph G and a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of G of width d , where $n = |V(G)|$. In all algorithmic statements that follow, we assume that G and \mathcal{P} are given on input.

Throughout this section our data structures work with the standard word RAM model.

4.1 Proximity oracle

For vertices $u, v \in V(G)$ and $r \in \mathbb{N}$, we define

$$\text{firstClose}_r(u, v) = \min\{t \mid \text{dist}_t(u\langle t \rangle, v\langle t \rangle) \leq r\}.$$

In other words, $\text{firstClose}_r(u, v)$ is the first time t such that the parts of \mathcal{P}_t containing u and v are at distance at most r in the impurity graph G_t^{imp} . Note that whenever $u \neq v$, we have $1 < \text{firstClose}_r(u, v) \leq n$. The main goal of this section is to construct an auxiliary data structure for answering queries about the values of $\text{firstClose}_r(\cdot, \cdot)$. This is described in the lemma below.

► **Lemma 20.** *For a given $r \in \mathbb{N}$, one can in time $\mathcal{O}_{d,r}(n)$ compute a data structure that can answer the following queries in time $\mathcal{O}_{d,r}(\log \log n)$: given $u, v \in V(G)$, output $\text{firstClose}_r(u, v)$.*

By Lemma 5, we may assume that the vertex set $V(G)$ is equal to $[n]$, and \mathcal{P} is a convex contraction sequence for the usual order on $[n]$. In particular, pairs of vertices can be identified with points in a plane, and intuitively, every pair of sets $A, B \subseteq V(G)$ corresponds

to a rectangle $A \times B \subseteq [n] \times [n]$. This correspondence will be important in the proof of Lemma 20, whose key technical component is the data structure for *orthogonal range queries* due to Chan [9], for manipulating rectangles in a plane. (We remark that the applicability of this data structure in the context of twin-width has already been observed in [19].) Let us recall the setting.

A *rectangle* is a set of pairs of integers of the form $\{(x, y) : a \leq x \leq a', b \leq y \leq b'\}$ for some integers a, a', b, b' . In all algorithmic statements that follow, every rectangle is represented by such a quadruple (a, a', b, b') . In the problem of orthogonal range queries, we are given a list of pairwise disjoint rectangles $\mathcal{R} = \{R_1, \dots, R_m\}$, all contained in $[n] \times [n]$, and the task is to set up a data structure that can efficiently answer the following queries: given $(x, y) \in [n] \times [n]$, output the index of the rectangle in \mathcal{R} that contains (x, y) , or output \perp if there is no such rectangle. Chan proposed the following data structure for this problem.

► **Theorem 21** ([9]). *Assuming $|\mathcal{R}| = \mathcal{O}(n)$, there is a data structure for the orthogonal range queries that takes $\mathcal{O}(n)$ space, can be initialized in time $\mathcal{O}(n)$, and can answer every query in time $\mathcal{O}(\log \log n)$.*

We remark that there is also a simple data structure for orthogonal range queries that for any fixed $\varepsilon > 0$, achieves query time $\mathcal{O}(1/\varepsilon)$ at the expense of space complexity and initialization time $\mathcal{O}(n^{1+\varepsilon})$. See the appendix of [19] for details. As we mentioned in Section 1, replacing the usage of the data structure of Chan with this simple data structure results in an analogous tradeoff in Theorem 1.

We reduce the statement of Lemma 20 to the result of Chan using the following lemma.

► **Lemma 22** (♠). *One can in time $\mathcal{O}_{d,r}(n)$ compute a list \mathcal{Q} of pairs of the form (R, t) , where $R \subseteq [n] \times [n]$ is a rectangle and $t \in [n]$, such that the following holds:*

- *the rectangles in pairs from \mathcal{Q} form a partition of $[n] \times [n]$, and*
- *for each $(u, v) \in [n] \times [n]$, if $(R, t) \in \mathcal{Q}$ is the unique pair satisfying $(u, v) \in R$, then we have $\text{firstClose}_r(u, v) = t$.*

Lemma 20 follows from Lemma 22 as follows. Let \mathcal{Q} be the list provided by Lemma 22; note that $|\mathcal{Q}| \leq \mathcal{O}_{d,r}(n)$, because this is an upper bound on the running time of the algorithm computing \mathcal{Q} . Let \mathcal{R} be the list of rectangles appearing in the pairs from \mathcal{Q} . Set up a data structure of Theorem 21 for \mathcal{R} and, additionally, for each $R \in \mathcal{R}$ remember the unique $t \in [n]$ such that $(R, t) \in \mathcal{Q}$. Then upon query $(u, v) \in [n] \times [n]$, it suffices to use the data structure of Theorem 21 to find the unique $R \in \mathcal{R}$ containing (u, v) and return the associated integer t .

4.2 The tree of r -close \mathbf{x} -tuples

In this section we are going to construct an auxiliary data structure for handling local types. Fix a number $k \in \mathbb{N}$; this is the quantifier rank of the types we would like to tackle. Denote $r := 2^k$. Also fix a finite set \mathbf{x} of variables, an n -vertex graph G , together with a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$.

For $s \in [n]$ and a tuple $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$, we call \mathbf{u} *r -close* at the time s if one cannot partition \mathbf{u} into two nonempty tuples $\mathbf{u}', \mathbf{u}''$ such that $\text{dist}_s(\mathbf{u}', \mathbf{u}'') > r$. Equivalently, if one considers an auxiliary graph on vertex set \mathbf{u} where two parts are connected iff they are at distance at most r in G_s^{imp} , then \mathbf{u} is r -close iff this auxiliary graph is connected. Note that if $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ is r -close at the time s , then for every t with $s \leq t \leq n$, the tuple $\mathbf{u}(s \rightarrow t)$ is also r -close at the time t .

123:14 Twin-Width and Types

For $s \in [n]$ with $s > 1$, by B_s denote the part of \mathcal{P}_s that is the union of two parts in \mathcal{P}_{s-1} . Let $T_{r,\mathbf{x}}$ be the set consisting of all pairs of the form (\mathbf{u}, s) such that $s \in [n]$, $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$ is r -close at the time s , and at least one of the following conditions is satisfied:

- $s = 1$; or
- $s > 1$ and $\text{dist}_s(B_s, \mathbf{u}) \leq r$; or
- $s < n$ and $\text{dist}_{s+1}(B_{s+1}, \mathbf{u}(s \rightarrow s+1)) \leq r$.

Note that as \mathbf{u} is assumed to be r -close, if the second condition holds then $\mathbf{u} \subseteq \text{Vicinity}_s^{r|\mathbf{x}|}(B_s)$, and if the third condition holds then $\mathbf{u}(s \rightarrow s+1) \subseteq \text{Vicinity}_{s+1}^{r|\mathbf{x}|}(B_{s+1})$. Since the trigraphs $\text{Vicinity}_s^{r|\mathbf{x}|}(B_s)$ and $\text{Vicinity}_{s+1}^{r|\mathbf{x}|}(B_{s+1})$ are of size $\mathcal{O}_{d,k,\mathbf{x}}(1)$, it follows that $T_{r,\mathbf{x}}$ contains $\mathcal{O}_{d,k,\mathbf{x}}(n)$ elements in total: n elements for $s = 1$ and $\mathcal{O}_{d,k,\mathbf{x}}(1)$ elements for each $1 < s \leq n$.

We consider an ancestor relation \preceq on $T_{r,\mathbf{x}}$ defined as follows:

$$(\mathbf{v}, t) \preceq (\mathbf{u}, s) \quad \text{if and only if} \quad s \leq t \text{ and } \mathbf{u}(s \rightarrow t) = \mathbf{v}.$$

It is easy to see $T_{r,\mathbf{x}}$ together with \preceq defines a rooted tree whose tree order is \preceq . The root is (\mathbf{r}, n) , where \mathbf{r} is the unique tuple of $\mathcal{P}_n^{\mathbf{x}}$, the one that maps all variables of \mathbf{x} to the unique part of \mathcal{P}_n . From now on we identify the set $T_{r,\mathbf{x}}$ with the tree it induces. Therefore, we call the elements of $T_{r,\mathbf{x}}$ *nodes* and the child-parent pairs in $T_{r,\mathbf{x}}$ the *edges* of $T_{r,\mathbf{x}}$.

► **Definition 23.** We call $T_{r,\mathbf{x}}$ the tree of r -close \mathbf{x} -tuples associated with G and the contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$.

Recall that $r = 2^k$, and $k \in \mathbb{N}$ is fixed. For every node $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$, let $\text{Types}_{\mathbf{u},s}^k$ be the set of all possible k -local types of tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ satisfying $\mathbf{u} = \mathbf{w}(s)$. By Lemma 12, there is a constant $M = \mathcal{O}_{d,k,\mathbf{x}}(1)$ such that $|\text{Types}_{\mathbf{u},s}^k| \leq M$ for every node (\mathbf{u}, s) , and $\text{Types}_{\mathbf{u},s}^k$ can be computed in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$ given access to G_s and \mathbf{u} .

Consider nodes $(\mathbf{u}, s), (\mathbf{v}, t) \in T_{r,\mathbf{x}}$ such that (\mathbf{v}, t) is the parent of (\mathbf{u}, s) . Let $e = ((\mathbf{u}, s), (\mathbf{v}, t))$ be the corresponding edge of $T_{r,\mathbf{x}}$. By Lemma 17, there exists a function $f_e: \text{Types}_{\mathbf{u},s}^k \rightarrow \text{Types}_{\mathbf{v},t}^k$ such that for every tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$ with $\mathbf{u} = \mathbf{w}(s)$, we have

$$\text{lt}_t^k(\mathbf{w}) = f_e(\text{lt}_s^k(\mathbf{w})). \quad (1)$$

We now verify that all the objects introduced above can be computed efficiently.

► **Lemma 24 (♠).** One can in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ compute the nodes and the edges of $T_{r,\mathbf{x}}$ (where $r = 2^k$) as well as, for every edge e of $T_{r,\mathbf{x}}$, the function f_e .

We can finally state and prove the main result of this section.

► **Lemma 25.** One can in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ construct a data structure that can answer the following queries in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$: given $\mathbf{w} \in V(G)^{\mathbf{x}}$, two nodes $(\mathbf{v}, t) \preceq (\mathbf{u}, s)$ of $T_{r,\mathbf{x}}$ such that $\mathbf{u} = \mathbf{w}(s)$ and $\mathbf{v} = \mathbf{w}(t)$, and the type $\text{lt}_s^k(\mathbf{w})$, output the type $\text{lt}_t^k(\mathbf{w})$.

Proof. Using Lemma 24 construct the tree $T_{r,\mathbf{x}}$ and functions f_e for the edges of $T_{r,\mathbf{x}}$. By Lemma 12, there is a constant $M = \mathcal{O}_{d,k,\mathbf{x}}(1)$ such that $|\text{Types}_{\mathbf{u},s}^k| \leq M$ for every node (\mathbf{u}, s) . Let $I := [M]$ be an indexing set of size M . Since for every node (\mathbf{u}, s) we have $|\text{Types}_{\mathbf{u},s}^k| \leq M$, we can set an arbitrary injection $\iota_{\mathbf{u},s}: \text{Types}_{\mathbf{u},s}^k \rightarrow I$. For an edge $e = ((\mathbf{u}, s), (\mathbf{v}, t))$, we set

$$g_e := \iota_{\mathbf{u},s}^{-1} \circ f_e \circ \iota_{\mathbf{v},t}.$$

Thus, g_e is a function from I to I that, intuitively, is just f_e reindexed using the index set I . Clearly, functions $\iota_{\mathbf{u},s}$ and g_e defined above can be computed in total time $\mathcal{O}_{d,k,\mathbf{x}}(n)$.

We will use the following result proved in [17].

► **Theorem 26** (Theorem 5.1 of [17]). *Let S be a semigroup and T be a rooted tree with edges labelled with elements of S . Then one can in time $|S|^{\mathcal{O}(1)} \cdot |T|$ construct a data structure that can answer the following queries in time $|S|^{\mathcal{O}(1)}$: given nodes $u, v \in T$ such that v is an ancestor of u , output the (top-down) product of elements of S associated with the edges on the path from v to u . The data structure uses $|S|^{\mathcal{O}(1)} \cdot |T|$ space.*

Let S be the semigroup of functions from I to I with the product defined as $f \cdot g = g; f$. Thus, the functions g_e form a labelling of edges of $T_{r,\mathbf{x}}$ with elements of S . Apply Theorem 26 to this S -labelled tree, and let \mathbb{S} be the obtained data structure. Now, to answer a query about nodes (\mathbf{u}, s) , (\mathbf{v}, t) , and type $\alpha = \text{ltp}_s^k(\mathbf{w})$ as in the lemma statement, it suffices to apply the following procedure:

- Compute $\tilde{\alpha} := \iota_{\mathbf{u},s}(\alpha)$.
- Query \mathbb{S} to compute the compositions of functions g_e along the path from (\mathbf{u}, s) to (\mathbf{v}, t) in $T_{r,\mathbf{x}}$. Call the resulting function h .
- Compute $\tilde{\beta} := h(\tilde{\alpha})$.
- Output $\beta := \iota_{\mathbf{v},t}^{-1}(\tilde{\beta})$.

The correctness of the procedure follows from a repeated use of (1), and it is clear that the running time is $\mathcal{O}_{d,k,\mathbf{x}}(1)$. ◀

4.3 Data structure

With all the tools prepared, we can prove Theorem 1.

Let k be the quantifier rank of φ . We set up two auxiliary data structures:

- The data structure of Lemma 20 for radius parameter $r = 2^k$. Call this data structure \mathbb{P} .
- For every $\mathbf{z} \subseteq \mathbf{x}$, the data structure of Lemma 25 for parameter k and the set of variables \mathbf{z} . Call this data structure $\mathbb{W}_{\mathbf{z}}$.

Moreover, using Lemma 4, we compute for each time $s \in [n - 1]$ the trigraph Relevant_s^p , where $p := r(|\mathbf{x}| + 1)$. These objects constitute our data structure, so by Lemmas 20, 25, and 4, the construction time is $\mathcal{O}_{d,\varphi}(n)$ as promised. It remains to show how to implement queries.

Suppose we are given a tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$ and we would like to decide whether $G \models \varphi(\mathbf{w})$. By Lemma 13, to answer this it suffices to compute $\text{ltp}_n^k(\mathbf{w})$. In the following, for $\mathbf{z} \subseteq \mathbf{x}$, by $\mathbf{w}_{\mathbf{z}}$ we denote the restriction of \mathbf{w} to the variables of \mathbf{z} .

For each time $s \in [n]$, let H_s be the graph on vertex set \mathbf{x} such that $y, y' \in \mathbf{x}$ are adjacent in H_s if and only if $\text{dist}_s(\mathbf{w}\langle s \rangle(y), \mathbf{w}\langle s \rangle(y')) \leq r$. The following are immediate:

- For all $1 \leq s \leq t \leq n$, H_t is a supergraph of H_s . That is, if $y, y' \in \mathbf{x}$ are adjacent in H_s , then they are also adjacent in H_t .
- If $\mathbf{z} \subseteq \mathbf{x}$ is such that $H_s[\mathbf{z}]$ is connected for some $s \in [n]$, then $\mathbf{w}_{\mathbf{z}}\langle s \rangle$ is r -close at the time s .

Using the data structure \mathbb{P} , we may compute $\text{firstClose}_r(y, y')$ for all $\{y, y'\} \in \binom{\mathbf{x}}{2}$ in total time $\mathcal{O}_{d,\varphi}(\log \log n)$. Let $S \subseteq [n]$ be the set of all those numbers, and include 1 and n in S in addition. Thus $|S| \leq 2 + \binom{|\mathbf{x}|}{2} \leq \mathcal{O}_{\varphi}(1)$. We imagine S as ordered by the standard order \leq , hence we may talk about consecutive elements of S .

Note that the knowledge of the numbers $\text{firstClose}_r(y, y')$ for $\{y, y'\} \in \binom{\mathbf{x}}{2}$ allows us to compute the graphs H_s for all $s \in S$. Further, observe that if $t \in [S]$ is such that $s \leq t < s'$ for some $s, s' \in S$ that are consecutive in S , then $H_t = H_s$.

123:16 Twin-Width and Types

Let L be the set of all pairs of the form (\mathbf{z}, s) where $s \in S$, \mathbf{z} is a connected component of H_s , and either $s = 1$ or \mathbf{z} is not connected in H_{s-1} . Clearly, L has size $\mathcal{O}_{d,\varphi}(1)$ and can be computed in time $\mathcal{O}_{d,\varphi}(1)$. Observe the following.

▷ **Claim 27.** For each $(\mathbf{z}, s) \in L$, we have $(\mathbf{w}_{\mathbf{z}}\langle s \rangle, s) \in T_{r,\mathbf{z}}$. If moreover $s > 1$, then for every $\mathbf{y} \subseteq \mathbf{z}$ that is a connected component of H_{s-1} , we have $(\mathbf{w}_{\mathbf{y}}\langle s-1 \rangle, s-1) \in T_{r,\mathbf{y}}$.

Proof. If $s = 1$, then \mathbf{z} being a connected component of H_1 means that \mathbf{z} is a constant tuple. Hence $\mathbf{w}_{\mathbf{z}}\langle 1 \rangle$ is r -close at the time 1, implying that $(\mathbf{w}_{\mathbf{z}}\langle 1 \rangle, 1) \in T_{r,\mathbf{z}}$.

Assume then that $s > 1$. As \mathbf{z} is not connected in H_{s-1} and is connected in H_s , it follows that for every connected component \mathbf{y} of H_{s-1} that is contained in \mathbf{z} , we have $\text{dist}_s(\mathbf{w}_{\mathbf{y}}\langle s \rangle, B_s) \leq r$, and in particular $\text{dist}_s(\mathbf{w}_{\mathbf{z}}\langle s \rangle, B_s) \leq r$. The latter statement implies that $(\mathbf{w}_{\mathbf{z}}\langle s \rangle, s) \in T_{r,\mathbf{z}}$ due to fulfilling the second condition in the definition of $T_{r,\mathbf{z}}$. Further, since \mathbf{y} is a connected component of H_{s-1} , $\mathbf{w}_{\mathbf{y}}\langle s-1 \rangle$ is r -close at the time $s-1$. So $(\mathbf{w}_{\mathbf{y}}\langle s-1 \rangle, s-1) \in T_{r,\mathbf{y}}$ due to fulfilling the third condition in the definition of $T_{r,\mathbf{y}}$. ◁

We now compute the types $\text{ltp}_s^k(\mathbf{w}_{\mathbf{z}})$ for all $(\mathbf{z}, s) \in L$. We do this in any order on L with non-decreasing s , hence when processing (\mathbf{z}, s) we may assume that the corresponding types have already been computed for all $(\mathbf{y}, t) \in L$ with $t < s$.

Assume first that $s = 1$. Then $(\mathbf{z}, 1) \in L$ means that \mathbf{z} is a connected component of H_1 , which in turn means that $\mathbf{w}_{\mathbf{z}}$ is a constant tuple. In this case $\text{ltp}_1^k(\mathbf{w}_{\mathbf{z}})$ can be computed trivially.

Assume then that $s > 1$. Since $(\mathbf{z}, s) \in L$, we have that \mathbf{z} is a connected component of H_s , but in H_{s-1} , \mathbf{z} breaks into two or more smaller connected components.

Consider any such component \mathbf{y} ; that is, \mathbf{y} is a connected component of H_{s-1} that is contained in \mathbf{z} . By Claim 27, we have $(\mathbf{w}_{\mathbf{y}}\langle s-1 \rangle, s-1) \in T_{r,\mathbf{y}}$. Let then $t \leq s-1$ be the smallest time such that \mathbf{y} is a connected component of H_t ; clearly we have $t \in S$ and $(\mathbf{y}, t) \in L$. By Claim 27 again, $(\mathbf{w}_{\mathbf{y}}\langle t \rangle, t) \in T_{r,\mathbf{y}}$. Since the type $\text{ltp}_t^k(\mathbf{w}_{\mathbf{y}})$ has been already computed before, we may use one query to $\mathbb{W}_{\mathbf{y}}$ to compute the type $\text{ltp}_{s-1}^k(\mathbf{w}_{\mathbf{y}})$.

Having performed the procedure described above for every connected component \mathbf{y} of H_{s-1} that is contained in \mathbf{z} , we may repeatedly use Lemma 15 to compute the type $\text{ltp}_{s-1}^k(\mathbf{w}_{\mathbf{z}})$. Note that for different components \mathbf{y}, \mathbf{y}' as above, we have $\text{dist}_{s-1}(\mathbf{w}_{\mathbf{y}}\langle s-1 \rangle, \mathbf{w}_{\mathbf{y}'}\langle s-1 \rangle) > r$ due to \mathbf{y} and \mathbf{y}' being non-adjacent in H_{s-1} . Furthermore, all trigraphs required in the applications of Lemma 15 can be easily deduced from the trigraph Relevant_{s-1}^p and the description of the contraction performed at the time $s-1$; these are stored in our data structure.

Finally, it remains to apply Lemma 17 to compute the type $\text{ltp}_s^k(\mathbf{w}_{\mathbf{z}})$ from $\text{ltp}_{s-1}^k(\mathbf{w}_{\mathbf{z}})$. Again, the trigraphs needed in this application can be easily deduced from Relevant_{s-1}^p and the contraction performed at the time $s-1$. This finishes the computation of types $\text{ltp}_s^k(\mathbf{w}_{\mathbf{z}})$ for all $(\mathbf{z}, s) \in L$; note that the running time is $\mathcal{O}_{d,\varphi}(1)$.

Finally, let t be the smallest time such that \mathbf{x} is connected in H_t . Such t exists since \mathbf{x} is connected in H_n . Clearly, $t \in S$. By definition we have $(\mathbf{x}, t) \in L$, so the type $\text{ltp}_t^k(\mathbf{w})$ has been computed. By Claim 27, $(\mathbf{w}\langle t \rangle, t) \in T_{r,\mathbf{x}}$. So we can now use the data structure $\mathbb{W}_{\mathbf{x}}$ one last time to compute $\text{ltp}_n^k(\mathbf{w})$. This finishes the proof of Theorem 1.

5 Query enumeration

In this section we prove Theorem 2, which we recall below for convenience.

► **Theorem 2.** *Suppose we are given an n -vertex graph G specified through a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of width d , and a first-order formula $\varphi(\mathbf{x})$, where \mathbf{x} is a set of variables. Then after preprocessing in time $\mathcal{O}_{d,\varphi}(n)$, one can enumerate all tuples $\mathbf{w} \in V(G)^\times$ such that $G \models \varphi(\mathbf{w})$ with $\mathcal{O}_{d,\varphi}(1)$ delay.*

First, we need to define our notion of enumerators, and prepare some tools for working with them.

Enumerators. Let x_1, \dots, x_n be a sequence of elements. An *enumerator* of the sequence x_1, \dots, x_n is a data structure that implements a single method, such that at the i th invocation of the method, it outputs the element x_j of the sequence, where $j = i \bmod n$, and reports an “end of sequence” message if $j = 0$. We say that the enumerator has *delay* t if each invocation takes at most t computation steps, including the steps needed to output the element x_j (assuming each element has a fixed representation). An enumerator for a set X is an enumerator for any sequence x_1, \dots, x_n with $\{x_1, \dots, x_n\} = X$ and $n = |X|$. Enumerators for Cartesian products and disjoint unions of sets can be obtained in an obvious way:

► **Lemma 28.** *Suppose we are given an enumerator for a set X with delay t and an enumerator for a set Y with delay t' , where $t, t' \geq 1$. Then we can construct in time $\mathcal{O}(1)$ an enumerator with delay $t + t' + \mathcal{O}(1)$ for the set $X \times Y$ and – if X and Y are disjoint – for the set $X \uplus Y$.*

We will also construct enumerators for disjoint unions of families of sets, as follows.

► **Lemma 29.** *Suppose X_1, \dots, X_n are pairwise disjoint, nonempty sets, such that X_i has an enumerator \mathcal{E}_i with delay t . Suppose furthermore we have an enumerator for the sequence $\mathcal{E}_1, \dots, \mathcal{E}_n$ with delay t' . Then one can construct, in time $\mathcal{O}(1)$ an enumerator for the set $\bigcup_{1 \leq i \leq n} X_i$ with delay $t + t' + \mathcal{O}(1)$.*

Finally, we will use the following lemma, proved in [17, Lemma 7.15].

► **Lemma 30.** *Fix a finite set Q of size q and a set of functions $\mathcal{F} \subseteq Q^Q$. There is a constant c computable from q and an algorithm that, given a rooted tree T , in which each edge vw (v child of w) is labelled by a function $f_{vw}: Q \rightarrow Q$, computes in time $c \cdot |T|$ a collection $(\mathcal{E}_w)_{w \in V(T)}$ of enumerators, where each \mathcal{E}_w is an enumerator with delay c that enumerates all descendants v of w such that the composition of the functions labeling the edges of the path from v to w , belongs to \mathcal{F} .*

This yields the following.

► **Corollary 31 (♠).** *Fix a number q . There is a constant c computable from q and an algorithm that, given a rooted tree T , in which each node v is labeled by a set X_v with $|X_v| \leq q$ and a set $Y_v \subseteq X_v$, and each edge vw (v child of w) is labelled by a function $f_{vw}: X_v \rightarrow X_w$, computes in time $c \cdot |T|$ a collection $(\mathcal{E}_w^\tau)_{w \in V(T), \tau \in X_w}$ of enumerators, where each \mathcal{E}_w^τ is an enumerator with delay c that enumerates all descendants v of w such that there is some $\sigma \in Y_v$ that is mapped to τ by the composition $f: X_v \rightarrow X_w$ of the functions labeling the edges of the path from v to w .*

Proof of Theorem 2. We now proceed to the proof of Theorem 2.

Fix a number k , a set of variables \mathbf{x} , an n -vertex graph G , together with its contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$. Denote $r := 2^k$.

For every $s \in [n]$ and \mathbf{x} -tuple $\mathbf{u} \in \mathcal{P}_s^{\mathbf{x}}$, and local type $\tau \in \text{Types}_{\mathbf{u},s}^k$, denote

$$S_{\mathbf{u},s}^\tau := \{\mathbf{w} \in V(G)^{\mathbf{x}} \mid \mathbf{w}\langle s \rangle = \mathbf{u} \text{ and } \text{ltp}_s^k(\mathbf{w}) = \tau\}.$$

Recall that the root of $T_{r,\mathbf{x}}$ is the pair (\mathbf{r}, n) , where \mathbf{r} is the constant \mathbf{x} -tuple with all components equal to the unique part of \mathcal{P}_n . Then $S_{\mathbf{r},n}^\tau$ is the set of all \mathbf{x} -tuples $\mathbf{w} \in V(G)^{\mathbf{x}}$ with $\text{ltp}_n^k(\mathbf{w}) = \tau$. From Lemma 13 and Lemma 7 we get:

► **Lemma 32.** *Fix a formula $\varphi(\mathbf{x})$ of quantifier-rank k . Then there is a set $\Gamma \subseteq \text{Types}_{\mathbf{r},n}^n$ such that $\varphi(G) := \{\mathbf{w} \in V(G)^{\mathbf{x}} \mid G \models \varphi(\mathbf{w})\}$ is the disjoint union of the family of sets $\{S_{\mathbf{r},n}^\tau \mid \tau \in \Gamma\}$.*

Therefore, an enumerator for $\varphi(G)$ can be obtained by concatenating enumerators for the sets $S_{\mathbf{r},n}^\tau$, for $\tau \in \Gamma$. Note that here we are concatenating only $\mathcal{O}_{k,d,\mathbf{x}}(1)$ enumerators, by Lemma 12, so, by applying Lemma 28 repeatedly, the resulting enumerator can be obtained in time $\mathcal{O}_{k,d,\mathbf{x}}(1)$ and has delay $\mathcal{O}_{k,d,\mathbf{x}}(1)$. So to prove Theorem 2, it suffices to prove that we can efficiently compute an enumerator for each of the sets $S_{\mathbf{r},n}^\tau$.

Recall that $T_{r,\mathbf{x}}$ is the tree of r -close \mathbf{x} -tuples (see Def. 23), and can be computed in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$, by Lemma 24. In the following proposition, we will show how to compute enumerators for all of the sets $S_{\mathbf{u},s}^\tau$, for $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$. All the enumerators jointly will be computed in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$.

► **Proposition 33.** *Fix a nonempty set \mathbf{x} of variables and $k \in \mathbb{N}$. Assume G is a graph on n vertices provided on input through a contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ of width d . Then one can in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ construct a data structure that associates, to every node (\mathbf{u}, s) of $T_{r,\mathbf{x}}$ and every local type $\tau \in \text{Types}_{\mathbf{u},s}^k$, an enumerator for all tuples in $S_{\mathbf{u},s}^\tau$ with delay $\mathcal{O}_{d,k,\mathbf{x}}(1)$.*

As noted above, Theorem 2 follows from Proposition 33, using Lemma 32. The rest of Section 5 is devoted to proving Proposition 33.

We prove Proposition 33 by induction on $|\mathbf{x}|$. So suppose the statement holds for all strict subsets of \mathbf{x} . Recall that we may construct the tree $T_{r,\mathbf{x}}$, in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$, using Lemma 24.

Let v, u be two nodes of $T_{r,\mathbf{x}}$ with $v = (\mathbf{v}, t)$ and $u = (\mathbf{u}, s)$ and $u \preceq v$. By Lemma 25, there is a function $f_{vu} : \text{Types}_{\mathbf{v},t}^k \rightarrow \text{Types}_{\mathbf{u},s}^k$ such that for every $\mathbf{w} \in V(G)^{\mathbf{x}}$, with $\mathbf{u} = \mathbf{w}\langle t \rangle$ we have $f_{vu}(\text{ltp}_t^k(\mathbf{w})) = \text{ltp}_s^k(\mathbf{w})$.

For a tuple $\mathbf{w} \in V(G)^{\mathbf{x}}$, let $s \in [n]$ be the first time such that $\mathbf{w}\langle s \rangle$ is r -close at time s , where $r = 2^k$. We then say that \mathbf{w} registers at (\mathbf{u}, s) , where $\mathbf{u} = \mathbf{w}\langle s \rangle$. By Claim 27, in this case, the pair (\mathbf{u}, s) is a node of $T_{r,\mathbf{x}}$.

For each node (\mathbf{u}, s) of $T_{r,\mathbf{x}}$ and type $\tau \in \text{Types}_{\mathbf{u},s}^k$, denote:

$$R_{\mathbf{u},s}^\tau = \{\mathbf{w} \in V(G)^{\mathbf{x}} \mid \mathbf{w} \text{ registers at } (\mathbf{u}, s) \text{ and } \text{ltp}_s^k(\mathbf{w}) = \tau\}.$$

Fix a node $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and a type $\tau \in \text{Types}_{\mathbf{u},s}^k$. Clearly, every tuple $\mathbf{w} \in S_{\mathbf{u},s}^\tau$ registers at exactly one descendant $v = (\mathbf{v}, t)$ of $u = (\mathbf{u}, s)$ (possibly, $v = u$), and moreover, $f_{vu}(\text{ltp}_t^k(\mathbf{w})) = \tau$. This proves the following.

► **Lemma 34.** *For every node $u \in T_{r,\mathbf{x}}$ and type $\tau \in \text{Types}_u^k$, the set S_u^τ is the disjoint union of all the sets R_v^σ , for $v \in T_{r,\mathbf{x}}$ with $v \succcurlyeq u$ and $\sigma \in \text{Types}_v^k$ such that $f_{vu}(\sigma) = \tau$.*

We prove the following two lemmas.

► **Lemma 35** (♠). *For every given node $u = (\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and type $\tau \in \text{Types}_{\mathbf{u},s}^k$, an enumerator for the set $R_{\mathbf{u},s}^\tau$ with delay $\mathcal{O}_{d,k,\mathbf{x}}(1)$ can be constructed in time $\mathcal{O}_{d,k,\mathbf{x}}(1)$.*

► **Lemma 36** (♠). *One can construct in time $\mathcal{O}_{d,k,\mathbf{x}}(n)$ a collection of enumerators \mathcal{E}_u^τ , one per each node $u = (\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and type $\tau \in \text{Types}_{\mathbf{u},s}^k$, where \mathcal{E}_u^τ has delay $\mathcal{O}_{d,k,\mathbf{x}}(1)$ and enumerates all descendants $v = (\mathbf{v}, t)$ of u in $T_{r,\mathbf{x}}$ such that there is some $\sigma \in \text{Types}_{\mathbf{v},t}^k$ with $f_{vu}(\sigma) = \tau$ and $R_{\mathbf{v},t}^\sigma \neq \emptyset$.*

Combining Lemma 34, Lemma 35, Lemma 36 and Lemma 29 yields the required collection of enumerators for each of the sets $S_{\mathbf{u},s}^k$, thus proving Proposition 33 and Theorem 2. The proof of Lemma 36 uses Corollary 31, and is omitted, due to space constraints. We now sketch the proof of Lemma 35.

Proof sketch for Lemma 35. The case when u is a leaf, that is, $s = 1$, is easily solved, since in this case $R_{\mathbf{u},s}^\tau$ is either empty, or consists of a single tuple. In the case when u is an inner node, consider the set

$$\mathcal{V} := \{\mathbf{v} \in \mathcal{P}_{s-1}^{\mathbf{x}} \mid \mathbf{v}\langle s-1 \rightarrow s \rangle = \mathbf{u}\}.$$

It is easy to see that \mathcal{V} has size $\mathcal{O}_{k,\mathbf{x},d}(1)$, and can be computed in this time, given $u \in T_{r,\mathbf{x}}$.

Fix $\mathbf{v} \in \mathcal{V}$, and consider the graph $H_{\mathbf{v}}$ with vertices \mathbf{x} where any two distinct $y, y' \in \mathbf{x}$ are adjacent whenever $\text{dist}_s(\mathbf{v}(y), \mathbf{v}(y')) \leq r$. Let $C_{\mathbf{v}}$ denote the set of connected components of $H_{\mathbf{v}}$, where each connected component is viewed as a set $\mathbf{y} \subseteq \mathbf{x}$ of vertices of $H_{\mathbf{v}}$. Then each of the sets $C_{\mathbf{v}}$, can be computed in time $\mathcal{O}_{k,\mathbf{x},d}(1)$, given $(\mathbf{u}, s) \in T_{r,\mathbf{x}}$ and $\mathbf{v} \in \mathcal{V}$. Call a tuple $\mathbf{v} \in \mathcal{V}$ *disconnected* if $H_{\mathbf{v}}$ is such. Note that if \mathbf{v} is disconnected and $\mathbf{y} \in C_{\mathbf{v}}$, then $|\mathbf{y}| < |\mathbf{x}|$.

Fix a disconnected tuple $\mathbf{v} \in \mathcal{V}$ and $\mathbf{y} \in C_{\mathbf{v}}$. Denote by $\mathbf{v}_{\mathbf{y}}$ the restriction of \mathbf{v} to \mathbf{y} . Note that the pair $(\mathbf{v}_{\mathbf{y}}, s-1)$ is a node of $T_{r,\mathbf{y}}$. Indeed, by assumption, $\text{dist}_s(B_s, \mathbf{u}) \leq r$ holds, so $\text{dist}_s(B_s, \mathbf{v}\langle s \rightarrow s+1 \rangle) \leq r$, and in particular $\text{dist}_s(B_s, \mathbf{v}_{\mathbf{y}}\langle s \rightarrow s+1 \rangle) \leq r$. Moreover, $\mathbf{v}_{\mathbf{y}}$ is r -close, since \mathbf{y} is a connected component of $H_{\mathbf{v}}$. As $|\mathbf{y}| < |\mathbf{x}|$, by inductive assumption, we have already computed enumerators for the sets $S_{\mathbf{v}_{\mathbf{y}},s-1}^\sigma$, for all adequate local types σ .

From Lemma 15 we deduce that the set $R_{\mathbf{u},s}^\tau$ is the disjoint union of sets of the form $S_{\mathbf{v},s-1}^{\bar{\tau}}$, where:

1. $\mathbf{v} \in \mathcal{V}$ is disconnected;
2. $\bar{\tau} := (\tau_{\mathbf{y}} : \mathbf{y} \in C_{\mathbf{v}})$ is a tuple of types with $\tau_{\mathbf{y}} \in \text{Types}_{\mathbf{v}_{\mathbf{y}},s-1}^k$ and which is “merged” into the type τ , using the function from Lemma 15;
3. the set $S_{\mathbf{v},s-1}^{\bar{\tau}}$ is a Cartesian product of the sets $S_{\mathbf{v}_{\mathbf{y}},s-1}^{\tau_{\mathbf{y}}}$.

Since for the sets $S_{\mathbf{v}_{\mathbf{y}},s-1}^{\tau_{\mathbf{y}}}$ we have enumerators by inductive assumption, by Lemma 28, we can compute in time $\mathcal{O}_{k,d,\mathbf{x}}(1)$ an enumerator for the set $S_{\mathbf{v},s-1}^{\bar{\tau}}$, with delay $\mathcal{O}_{k,d,\mathbf{x}}(1)$. Finally, we obtain an enumerator for $R_{\mathbf{u},s}^\tau$, by concatenating the $\mathcal{O}_{k,d,\mathbf{x}}(1)$ enumerators for the sets $S_{\mathbf{v},s-1}^{\bar{\tau}}$. This finishes the proof sketch for Lemma 35. ◀

References

- 1 Matthias Aschenbrenner, Alf Dolich, Deirdre Haskell, Dugald Macpherson, and Sergei Starchenko. Vapnik-chervonenkis density in some theories without the independence property, i. *Transactions of the American Mathematical Society*, 368, September 2011.

- 2 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proceedings of the 15th Annual Conference on Computer Science Logic, CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. doi:10.1007/11874683_11.
- 3 J. T. Baldwin and S. Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.
- 4 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021. Accepted to STOC 2022. arXiv:2102.03117.
- 5 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. *CoRR*, abs/2111.00282, 2021. To appear in the proceedings of SODA 2022. arXiv:2111.00282.
- 6 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation, IPEC 2021*, volume 214 of *LIPICs*, pages 10:1–10:16. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.IPEC.2021.10.
- 7 Édouard Bonnet, Eun Jung Kim, Stéphan Thomasse, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *Proceedings of the IEEE 61st Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 601–612. IEEE Computer Society, 2020.
- 8 Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.
- 9 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1131–1145. SIAM, 2011.
- 10 Thomas Colcombet. A combinatorial theorem for trees. In *ICALP 2007*, volume 4596 of *Lecture Notes in Computer Science*, pages 901–912. Springer, 2007.
- 11 Zdenek Dvorák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013.
- 12 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.
- 13 Jakub Gajarský, Michał Pilipczuk, and Szymon Toruńczyk. Stable graphs of bounded twin-width. *CoRR*, abs/2107.03711, 2021. arXiv:2107.03711.
- 14 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25:1–25:12, 2013. doi:10.1145/2528928.
- 15 Wojciech Kazana and Luc Segoufin. First-order queries on classes of structures with bounded expansion. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:25)2020.
- 16 Adam Paszke and Michał Pilipczuk. VC density of set systems definable in tree-like graphs. In *Proceedings of the 45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020*, volume 170 of *LIPICs*, pages 78:1–78:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.MFCS.2020.78.
- 17 Michał Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Toruńczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. *CoRR*, abs/2111.03725, 2021. arXiv:2111.03725.
- 18 Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. On the number of types in sparse graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 799–808. ACM, 2018. doi:10.1145/3209108.3209178.
- 19 Michał Pilipczuk, Marek Sokółowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. *CoRR*, abs/2110.08106, 2021. arXiv:2110.08106.

- 20 Wojciech Przybyszewski. VC-density and abstract cell decomposition for edge relation in graphs of bounded twin-width. *CoRR*, abs/2202.04006, 2022. [arXiv:2202.04006](https://arxiv.org/abs/2202.04006).
- 21 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.
- 22 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.
- 23 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.