Generalized model-checking over locally tree-decomposable classes

Markus Frick

Institut für Mathematische Logik, Eckerstr.1, 79104 Freiburg, Germany
frick@sunpool.mathematik.uni-freiburg.de

Abstract. It has been proved in [12] that properties of graphs or other relational structures that are definable in first-order logic can be *decided* in linear time when the input structures are restricted to come from a *locally tree-decomposable* class of structures. Examples of such classes are the class of planar graphs or classes of graphs of bounded valence.
In this paper, we consider more general computational problems than decision problems. We prove that *construction*, *listing*, and *counting* problems definable in first-order logic can be solved in linear time on locally tree-decomposable classes of structures.

1 Introduction

Model-checking problems are general algorithmic problems that can be used to model a wide range of concrete problems from different areas of computer science, among them database theory, algorithm theory and constraint satisfaction (AI). In the basic version of model-checking, we have given a relational structure $\mathfrak{A}$ and a sentence φ of some logic $\mathcal{L}$ and ask if φ is satisfied by $\mathfrak{A}$. We call this problem the *basic model-checking problem* for $\mathcal{L}$. Note that the basic model-checking problem can only be used to model decision problems. But in practice, we are not only confronted with decision problems. For example, in the context of databases, we usually do not want to know, whether some employee lives in London, but want the system to return these employees, or at least one of them.

An area where model-checking turned out to be a very successful tool is algorithm theory. Standard algorithmic problems like 3-COLORABILITY, CLIQUE or SET-COVER are instances of model-checking. Consider, for instance, the clique-problem that is, we have given a $k \geq 1$ and a graph $\mathcal{G} = (G, E)$ and want to decide if there is a clique of size k in $\mathcal{G}$. This problem is equivalent to decide if the sentence $\varphi^k_{\text{clique}} := \exists x_1, \ldots, x_k \bigwedge_{1 \leq i < j \leq k} Ex_ix_j$ holds in $\mathcal{G}$, hence, k-clique can be seen as a special case of the model-checking problem.

Up to now, we only considered decision problems. As already mentioned, it is natural not only to ask if there is a k-clique but, if so, construct a solution, e.g. return a k-clique of $\mathcal{G}$. On the logic side, this can be modeled by using formulas with free variables instead of sentences, and by finding a satisfying assignment. We call this problem the *construction* problem for model-checking. Other natural extensions are the *listing problem*, which asks for all satisfying assignments and the *counting problem* asking for the number of satisfying assignments. In this taxonomy, the basic model-checking problem is called the *decision problem*, or just *model-checking* problem (cf. [15] for a survey).

We consider these generalized model-checking problems of first-order logic. In the realm of databases this logic has a predominant role, since it closely resembles the commercial standard query language SQL. Unfortunately, model-checking for first-order logic is of very high complexity. Already over structures with only two elements it is PSPACE-complete [18]. This suggests that the bulk of complexity is contributed by the query and not by the structure. One way to explore this in more detail is to exhibit which part of the input contributes to what extend to the complexity of the problem. In a general setting this has been done systematically by Downey and Fellows in [6]. Applied to our problem, we declare φ to be the parameter and look if there is an algorithm working in time $O(f(\|\varphi\|) \cdot \|\mathfrak{A}\|^c)$ for some function $f : \mathbb{N} \to \mathbb{N}$ and $c > 0$. If this is the case,

we call a problem *fixed-parameter tractable*. In some sense this corresponds to our intuition that we evaluate "small" formulas in "big" structures. In [7] it is shown that parameterized model-checking for first-order logic is AW[1]-complete, which makes it very unlikely for model-checking to be fixed-parameter tractable.

One way to overcome this negative news is to consider certain restrictions on the admitted input structures. A landmark in this direction is Courcelle's result: over graphs of tree-width bounded by some constant w, monadic second-order model-checking can be done in *fixed-parameter linear time*, i.e. in time $O(f(\|\varphi\|) \cdot \|\mathfrak{A}\|)$ for some (here fast growing) function f [5]. This result was extended to the counting case [2] and recently to the construction and listing case [10]. Note that for the listing problem, linear time means linear in the size of the input plus the output.

There exist other successful restrictions take make algorithms more efficient. *Planarity* and *bounded valence* are two important and natural examples. It is hopeless to expect for these classes such a strong result like Courcelle's, since 3-COLORABILITY, which is definable in monadic second-order logic, remains NP-complete over planar graphs of valence at most 4 [14].

Nevertheless, such graphs share the nice property that their tree-width only depends on the diameter. This was first exploited by Baker [3] and later investigated more systematically by Eppstein [9]. Their ideas led Frick and Grohe [12] to the notion of *locally tree-decomposable* classes of structures. Important locally tree-decomposable classes are graphs of bounded genus, bounded valence and of bounded tree-width. For such classes they it was furthermore shown that model-checking for first-order logic is in fixed-parameter linear time [12]. Observe here that this result only concerns the decision problem.

We prove that all generalized model-checking problems, that is, the construction, listing, and counting problem over locally tree-decomposable classes[1] $\mathcal{C}$ are in fixed-parameter linear time. In particular, there are fixed-parameter linear time algorithms for the following problems: given a first-order formula $\varphi(\bar{x})$ and a structure $\mathfrak{A} \in \mathcal{C}$:

(1) compute an $\bar{a}$ such that $\mathfrak{A} \models \varphi(\bar{a})$ (construction)
(2) compute all $\bar{a}$ such that $\mathfrak{A} \models \varphi(\bar{a})$ (listing)
(3) compute the number of $\bar{a}$ such that $\mathfrak{A} \models \varphi(\bar{a})$ (counting)

These results can be seen as a further step in a systematic analysis of the parameterized complexity of model-checking: given a logical language $\mathcal{L}$, which restrictions on the inputs allow fixed-parameter tractability. Or with more emphasis on combinatorics, it can be considered as one of the first systematic studies of fixed parameter tractability of construction, listing and counting problems.

But apart from the systematic importance of these results, there are several practical implications on standard algorithmic problems. For example, our result shows that there is a linear time algorithm that counts the number of 4-cliques in a given graph of valence 5, or a linear time algorithm counting the number of dominating sets of size 10 in a given planar graph.

More generally, we see that a dominating set of size k in planar graphs not only can be computed (this was shown in [6]), but we can also list and count them in linear time. Another example is k-SET-COVER, which asks whether a given familiy $\mathcal{F}$ of sets has a subfamily $\mathcal{S}$ of size k such that $\bigcup \mathcal{S} = \bigcup \mathcal{F}$. Its first-order formulation restricted to structures of bounded valence implies that for instances $\mathcal{F}$, such that each $x \in \bigcup \mathcal{F}$ is contained in at most constant many $F \in \mathcal{F}$ and all $F \in \mathcal{F}$ having bounded size, we get linear time algorithms for the mentioned tasks. The same holds for all first-order properties like $\mathcal{H}$-HOMOMORPHISM (is there a homomorphic copy of $\mathcal{H}$?), k-DOMINATING-SET and so forth.

Organization: After the preliminaries, we introduce the concept of locally tree-decomposable classes (section 3). In section 4, we present the main results and give an overview of the structure

[1] Actually we use a slightly more restrictive notion. But this notion still comprises all important examples.

of the proofs. The outline of these proofs occupy the rest of the paper, i.e. section 5 exhibits a normal form for local first-order formulas. Then section 6 provides an alternative characterization for satisfying assignments, which will be used to solve the listing problem and construction problem in section 7. Finally, in section 8, we sketch the algorithm for the counting problem. Due to space limitations, we have deferred exact proofs of the theorems and proofs of helping lemmas to an appendix.

2 Preliminaries

For $n \geq 1$, we set $[n] := \{1, \ldots, n\}$. By $\mathrm{Pow}(A)$ we denote the power set of A and $\mathrm{Pow}^{\leq l}(A)$ is the set of elements of $\mathrm{Pow}(A)$ that have cardinality $\leq l$.

A *vocabulary* is a finite set τ of relation symbols each of which has an associated natural number, its *arity*. A *τ-structure* $\mathfrak{A}$ consists of a non-empty, finite set A called the *universe* of $\mathfrak{A}$ and a relation $R^{\mathfrak{A}} \subseteq A^r$ for each r-ary $R \in \tau$. For $B \subseteq A$ $\langle B \rangle^{\mathfrak{A}}$ denotes the *substructure* of $\mathfrak{A}$ induced by B. If the context is clear we omit superscripts. We only consider finite structures.

A *graph* is an $\{E\}$-structure $(G, E^{\mathcal{G}})$ where $E^{\mathcal{G}}$ is an anti-reflexive and symmetric binary relation (in other words: we consider simple undirected graphs without loops). A *colored graph* is a structure $\mathfrak{B} = (B, E^{\mathfrak{B}}, P_1^{\mathfrak{B}}, \ldots, P_m^{\mathfrak{B}})$, where $(B, E^{\mathfrak{B}})$ is a graph and the unary relations $P_1^{\mathfrak{B}}, \ldots, P_m^{\mathfrak{B}}$ form a partition of the universe B.

Let τ be a vocabulary. The set $\mathrm{FO}[\tau]$ of *first-order formulas* is built up in the usual way from an infinite supply of variables $x, y, x_1, x_2, \ldots$, the relation symbols $R \in \tau$ and $=$, the connectives $\vee, \wedge, \neg, \rightarrow$ and the quantifiers $\forall x, \exists x$ ranging over elements of the universe of the structure. A *free variable* in a formula φ is a variable x occurring in φ that is not in the scope of a quantifier $\exists x, \forall x$. We write $\varphi(x_1, \ldots, x_m)$ to indicate that the free variables in φ are exactly $x_1, \ldots, x_m$. A *sentence* is a formula that contains no free variables. The semantics of FO should be clear. For instance, $\varphi(x_1, x_2) := \exists x_3(Ex_1x_2 \wedge Ex_2x_3 \wedge Ex_3x_1)$ says that x_1 and x_2 are part of a triangle.

For a τ-structure $\mathfrak{A}$, $a_1, \ldots, a_m \in A$ and a formula $\varphi(x_1, \ldots, x_m) \in \mathrm{FO}[\tau]$ we write $\mathfrak{A} \models \varphi(a_1, \ldots, a_m)$ to say that $\mathfrak{A}$ satisfies φ, if the $x_1, \ldots, x_m$ are interpreted by the elements $a_1, \ldots, a_m$, respectively. For a structure $\mathfrak{A}$ and a formula $\varphi(x_1, \ldots, x_m)$ we let

$$\varphi(\mathfrak{A}) := \{(a_1, \ldots, a_m) \in A^m \mid \mathfrak{A} \models \varphi(a_1, \ldots, a_m)\}.$$

In case of no free variables in φ, we define $\varphi(\mathfrak{A}) := \{\emptyset\}$, if $\mathfrak{A} \models \varphi$ and $:= \emptyset$ otherwise. If the vocabularies of $\mathfrak{A}$ and φ do not agree, we set $\varphi(\mathfrak{A}) := \emptyset$. Here the convention that all variables of φ enclosed within the brackets actually occur freely in φ becomes crucial.

Algorithms: Our underlying model of computation is the standard RAM-model with addition and subtraction as arithmetic operations [1]. We assume the *uniform cost measure*, hence arithmetics can always be done in constant time. Observe here that the results concerning construction and listing also hold under the assumption of the logrithmic cost measure.

A relational structure $\mathfrak{A}$ is coded by a word w consisting of the vocabulary, the elements of the universe A, and the relations $R^{\mathfrak{A}}$ for $R \in \tau$. This is considered the standard coding for algorithms. It becomes particular important due to the fact that we aim at algorithms running in time linear in the size of the input structure. For graphs, we use its adjacency list representation, i.e. we code a graph by the list of its vertices, and associated with each vertex the list of its adjacent vertices. Formulas are coded in some reasonable way, e.g. by a string or its parse tree. For further details, the reader is referred to [10]. To improve readibility, we sometimes omit the O-notation, despite that there are always constants involved depending on the hardware and used data structures.

In running time estimations we use $\|o\|$ to denote the actual size of the object o, w.r.t some encoding, whereas $|o|$ refers to the cardinality of the set o. To emphasize the difference let $\mathcal{T}$ be a familiy of sets. Then $|\mathcal{T}|$ refers to the cardinality of $\mathcal{T}$, while $\|\mathcal{T}\|$ stands for $\sum_{X \in \mathcal{T}} \|X\|$.

Model-checking problems: The basic model-checking problem asks if a given structure satisfies a given sentence. Here we shall also consider formulas with free variables. For each class $\mathcal{C}$ of finite structures this gives rise to the following four problems: the input is a structure $\mathfrak{A} \in \mathcal{C}$ and a formula $\varphi(\bar{x}) \in \mathrm{FO}$

The decision problem. Decide if $\varphi(\mathfrak{A}) \neq \emptyset$. For sentences, this problem coincides with the basic model-checking problem, and therefore we refer to it as FO-MODEL-CHECKING on $\mathcal{C}$.

The construction problem. Compute one tuple $\bar{a} \in \varphi(\mathfrak{A})$, if one exists. We refer to this problem as FO-CONSTRUCTION on $\mathcal{C}$.

The listing problem. Compute the set $\varphi(\mathfrak{A})$. Because of its important application in database theory, we refer to this problem as FO-EVALUATION on $\mathcal{C}$.

The counting problem. Compute the cardinality of the set $\varphi(\mathfrak{A})$. We refer to this problem as FO-COUNTING on $\mathcal{C}$.

Note that we always investigate the parameterized complexity of these problems. We call a model-checking problem on $\mathcal{C}$ *fixed-parameter linear time*, if there is an algorithm that, given $\varphi(\bar{x}) \in \mathrm{FO}$ and $\mathfrak{A} \in \mathcal{C}$ solves the problem in time linear in the size of the structure $\mathfrak{A}$ plus the size of output. Observe that for the listing problem it is necessary to include the output in the running time estimation. For further details and examples of generalized model-checking the reader is referred to [15]. For more general background on this taxonomy of combinatorial problems, see [16, 17].

Although our algorithms depend heavily on the notion of *tree-width*, it is not necessary to introduce it formally. The interested reader is referred to [**?**,5, 10]. Intuitively, the tree-width measures the similarity of a structure with a tree. For instance, a tree $\mathcal{T}$ has tree-width $\mathrm{tw}(\mathcal{T}) = 1$ and a cycle of arbitrary length has treewidth = 2. In contrast to these structures of "small" tree-width, an $n \times n$-grid has treewidth = n. To exploit tree-likeness in algorithms we need to compute so called *tree-decompositions*. By a result of Bodlaender [**?**] this can be done in time linear in the size of the input structure.

In the context of model-checking, tree-decompositions allow the application of automata. This led to the following results, which actually have been proved for monadic second-order logic. All model-checking algorithms that will be presented in the sequel will use these results.

Theorem 1 ([5, 2, 10]). *Let $\mathcal{C}$ be a class of structures of bounded tree-width.* FO-MODEL-CHECKING, FO-CONSTRUCTION, FO-EVALUATION and FO-COUNTING on $\mathcal{C}$ are all in fixed-parameter linear time.

3 Locally tree-decomposable classes

Let τ be a vocabulary and $\mathfrak{A}$ a τ-structure. The Gaifman graph $\mathcal{G}(\mathfrak{A})$ of $\mathfrak{A}$ is the graph $(G, E^{\mathcal{G}})$ with $G = A$ and $E^{\mathcal{G}} := \{(a, b) \mid$ there is a k-ary $R \in \tau$ and $\bar{a} \in R^{\mathfrak{A}}$ such that $a, b \in \{a_1, \ldots, a_k\}\}$. Then $d^{\mathfrak{A}}(a, b)$ denotes the distance between a and $b \in A$ in $\mathcal{G}(\mathfrak{A})$. Consequently, for $r \geq 1$ we define $N_r^{\mathfrak{A}}(a) := \{b \in A \mid d^{\mathfrak{A}}(a, b) \leq r\}$, the *$r$-neighborhood* of $a \in A$ and $N_r^{\mathfrak{A}}(X) := \bigcup_{a \in X} N_r^{\mathfrak{A}}(a)$ for some $X \subseteq A$.

It is easy to see that for every $r \geq 1$ there is a formula $\delta_r(x, y) \in \mathrm{FO}[\tau]$ such that for all τ-structures $\mathfrak{A}$ and $a, b \in A$ we have $\mathfrak{A} \models \delta_r(a, b) \Leftrightarrow d^{\mathfrak{A}}(a, b) \leq r$. Within formulas we write $d(x, y) \leq r$ instead of $\delta_r(x, y)$ and $d(x, y) > r$ instead of $\neg\delta_r(x, y)$.

If $\varphi(x)$ is a first-order formula, then $\varphi^{N_r(x)}(x)$ is the formula obtained from $\varphi(x)$ by relativizing all quantifiers to $N_r(x)$, that is, by replacing every subformula of the form $\exists y \psi(x, y, \bar{z})$ by $\exists y(d(x, y) \leq r \wedge \psi(x, y, \bar{z}))$ and every subformula of the form $\forall y \psi(x, y, \bar{z})$ by $\forall y(d(x, y) \leq r \rightarrow \psi(x, y, \bar{z}))$. Generally, we consider formulas relativized to neighborhoods around tuples, in particular a formula $\psi(\bar{x})$ of the form $\varphi^{N_r(\bar{x})}(\bar{x})$, for some $\varphi(\bar{x})$, is called *r-local* around $\bar{x}$.

The next theorem is due to Gaifman and states that properties expressible in first-order logic are local.

Theorem 2 (Gaifman [13]). *Every first-order formula is equivalent to a Boolean combination χ of t-local formulas and sentences of the form*

$$\exists x_1 \dots \exists x_m \big(\bigwedge_{1 \le i < j \le m} d(x_i, x_j) > 2r \wedge \bigwedge_{1 \le i \le m} \psi(x_i) \big),$$

for suitable $t, r, m \ge 1$ and an r-local $\psi(x)$.

In [12] the notion of locally tree-decomposable classes of structures was introduced. We use a slightly more restrictive version of this notion (condition (3) is new).

Definition 3. *Let $r, l \ge 1$ and $g : \mathbb{N} \to \mathbb{N}$. A* nice (r, l, g)-tree cover *of a structure $\mathfrak{A}$ is a family $\mathcal{T}$ of subsets of A such that*

(1) For every $a \in A$ there exists a $U \in \mathcal{T}$ such that $N_r^{\mathfrak{A}}(a) \subseteq U$.
(2) for each $U \in \mathcal{T}$ there are less than l many $V \in \mathcal{T}$ such that $U \cap V \neq \emptyset$
(3) for all $U_1, \dots, U_q \in \mathcal{T}$ and $q \ge 1$ we have

$$tw(\langle U_1 \cup \dots \cup U_q \rangle^{\mathfrak{A}}) \le g(q).$$

A class $\mathcal{C}$ of structures is nicely locally tree-decomposable*, if there is a linear time algorithm that, given $\mathfrak{A} \in \mathcal{C}$ and $r \ge 1$, computes a nice (r, l, g)-tree cover of $\mathfrak{A}$, for suitable l, g.*

The intention of this notion is to cover a structure $\mathfrak{A}$ by subsets U such that each $a \in A$ is covered together with some sufficiently big neighborhood (condition (1)). This makes it possible to evaluate an r-local formula $\varphi(a)$ correctly in $\langle U \rangle^{\mathfrak{A}}$. Additionally, we have a bound on the treewidth of $\langle U \rangle^{\mathfrak{A}}$, which allows efficient model-checking in this part (condition (3)). Finally, the second condition limits the dependency of the different parts of the cover. This is a merely pragmatic definition whose objective is to allow efficient model-checking, while at the same time being general enough to comprise sufficiently rich classes.

Examples of nicely locally tree-decomposable classes ([9, 12, 11]).

(1) *Structures of bounded tree-width*
A class of structures of bounded tree-width is trivially nicely locally tree-decomposable.
(2) *Planar graphs*
Let $\mathcal{G}$ be a planar graph, $r \ge 1$ and $v \in G$ an arbitrary vertex. For $i \ge 1$ define $G_i := \{x \in G \mid (i-1)r \le d^{\mathcal{G}}(v, x) < (i+1)r\}$. Then the family $\{G_i \mid i \ge 1, G_i \neq \emptyset\}$ forms a nice $(r, 5, q \mapsto 9qr)$-tree cover of $\mathcal{G}$. To see this, first note that a planar graph with diameter r has tree-width at most $3r$ [**?**]. For $k \ge 3$ consider the graph induced by $\bigcup_{i=1}^{k} G_i$ and contract the inner $k-2$ rings $G_1, \dots, G_{k-2}$ to the vertex v; we obtain $\langle G_k \rangle$ with v in the middle. Since this graph is still planar and has diameter $\le 3r$, we get $9r$ as an upper bound on the tree-width of $\langle G_k \rangle$. This argumentation easily extends to unions of several G_i.
Note that in this way we get nice tree covers for all minor-closed classes $\mathcal{C}$ of graphs whose *local tree-width* is bounded uniformly by some function $f : \mathbb{N} \to \mathbb{N}$, i.e. for all $\mathcal{G} \in \mathcal{C}$ and $r \ge 1$: $\max_{v \in G} \mathrm{tw}(\langle N_r^{\mathcal{G}}(v) \rangle) \le f(r)$. This holds e.g. for graphs of bounded crossing number and graphs embeddable into some fixed surface.
(3) *Structures of bounded valence*
For structures $\mathfrak{A}$ of valence bounded by $d \ge 1$, we simply take the cover $\{N_r(x) \mid x \in A\}$. It is easy to confirm that this forms a nice $(r, (d-1)d^{r-1}, q \mapsto q \cdot (d-1)d^{r-1})$-tree cover.

In the sequel we always omit the term *nice* and just use *tree cover*. Nevertheless, all covers we use here are actually nice. The starting point for our investigation is the following theorem

Theorem 4 ([12, 11]). *Let $\mathcal{C}$ be a locally tree-decomposable class of structures. Then* FO-MODEL-CHECKING *on $\mathcal{C}$ is in fixed-parameter linear time.*

4 The main results

Our main result extends the previous theorem and essentially says that all four generalized model-checking problems over locally tree-decomposable classes are in fixed-parameter linear time. This is particularly surprising, since generally counting is assumed to be more difficult than the other problems.

Theorem 5. *Let $\mathcal{C}$ be a locally tree-decomposable class of structures. Then* FO-CONSTRUCTION, FO-EVALUATION *and* FO-COUNTING *on $\mathcal{C}$ is in fixed-parameter linear time.*

The crucial observation is that otherwise intractable local problems can be efficiently solved on structures that are locally tree-like. In [12] this observation has been applied to first-order logic and yielded a fixed-parameter linear time algorithm which decides if a first-order sentence holds in a locally tree-decomposable structure (cf. theorem 4). Our results require the treatment of formulas. As a matter of fact, by Gaifman's theorem and the model-checking result for sentences, we can restrict our attention to formulas $\varphi(\bar{x}) \in \mathrm{FO}$ being t-local around $\bar{x}$ for some $t \geq 1$. The proof of theorem 5 is structured as follows: in section 5, we provide a suitable normal form for $\varphi(\bar{x})$, i.e. we represent $\varphi(\bar{x})$ as a disjoint disjunction of t-local conjunctions $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x})$ (formula (2)). The parts of these conjunctions have (i) disjoint sets of free variables and (ii) these sets $\bar{x}_i$ of free variables are forced to be pairwise "far" away (cf. formula (4) for the appearance of these conjunctions).

Then section 6 combines this normal form with a suitable tree cover $\mathcal{T}$ of the input structure $\mathfrak{A} \in \mathcal{C}$. For that, we define an equivalence relation on the cover sets to track their mutual dependency (w.r.t. evaluation of the local conjuncts). This gives us an alternative characterization of the objective set $\varphi(\mathfrak{A})$ as a union of Cartesian products (cf. formula (8)), each of which is locally computable (hence efficiently computable, by local tree-likeness). Finally, we exhibit that the indices admitted in the aforemention union correspond to independent tuples of a colored graph $\mathcal{G}$ of bounded valence. In this context, independent tuples are independent sets ordered with respect to their color.

Based on this transformation of the starting problem, the algorithms solving the listing and the construction problems are presented in section 7. These algorithms essentially mimic the mentioned transformation and reduce the problem to finding all or a single of those independent tuples in $\mathcal{G}$. Since $\mathcal{G}$ has bounded valence, these are easy to enumerate and find, respectively.

The counting algorithm needs further effort. After a more involved transformation of the initial problem (not presented in detail), it is reduced to the problem of computing the sum over all independent tuples of a new colored graph $\mathcal{G}'$ with integers attached to its vertices (lemma 12). Then, by a dynamic programming approach, this sum is evaluated in linear time completing the sought result (theorem 13).

5 A normal form for local formulas

Let $k \geq 1$. A *k-distance-type* is an undirected graph $\epsilon := ([k], E^\epsilon)$. For a natural number t and a structure $\mathfrak{A}$ we say that a k-tuple $\bar{a} \in A^k$ *realizes* ϵ w.r.t. $\mathfrak{A}$ and t ($\bar{a} \models_{\mathfrak{A},t} \epsilon$), if there is an edge between distinct vertices i and j of ϵ if, and only if $d(a_i, a_j) \leq 2t + 1$.

Note that realization of a k-distance type ϵ w.r.t. t can be defined in first-order logic by a t-local formula $\rho_{t,\epsilon}(\bar{x})$. Now assume that ϵ splits into connected components $\epsilon_1, \dots, \epsilon_p$. We let $\epsilon_{i,\nu}$ stand for the ν-th coordinate of the i-th component (with respect to the natural ordering on the indices). With $\bar{a} \restriction \epsilon_i$ we denote the projection of $\bar{a}$ onto the coordinates contained in ϵ_i.

The next lemma provides a normal form for t-local formulas, which will allow a separate treatment of the variables corresponding to different ϵ-components. The proof is a straightforward application of Ehrenfeucht-Fraisse games, using the fact that local strategies on non-intersecting substructures extend to strategies on their union (see [8]).

Lemma 6. *Given a t-local $\varphi(\bar{x}) \in \mathrm{FO}$ for some $t \geq 1$. Then for every distance-type ϵ with connected components $\epsilon_1, \dots, \epsilon_p$ we can find a Boolean combination $F(\bar{\varphi}_1(\bar{x} \restriction \epsilon_1), \dots, \bar{\varphi}_p(\bar{x} \restriction \epsilon_p))$ of formulas $\varphi_{i,j}(\bar{x} \restriction \epsilon_i)$, $1 \leq i \leq p$ and $1 \leq j \leq m_i$ such that*

(1) for all i, the free variables of $\bar{\varphi}_i(\bar{x})$ are among $\bar{x} \restriction \epsilon_i$,
(2) the $\bar{\varphi}_i(\bar{x})$ are t-local around their free variables, and
(3) $\rho_{t,\epsilon}(\bar{x}) \models (\varphi(\bar{x}) \leftrightarrow F(\bar{\varphi}_1(\bar{x}), \dots, \bar{\varphi}_p(\bar{x})))$

Observe that using Gödel's completeness theorem, we can effectively compute this normal form. Let $\varphi(\bar{x})$ be a t-local formula, $\epsilon = \epsilon_1 \cup \dots \cup \epsilon_p$ a distance-type and let $F, \varphi_{i,j}(\bar{x} \restriction \epsilon_i)$ denote the formulas obtained by lemma 6. Next we transform $F((\bar{\varphi}_1(\bar{x} \restriction \epsilon_1), \dots, \bar{\varphi}_p(\bar{x} \restriction \epsilon_p))$ into disjunctive normal form, such that variables from different ϵ-components never occur in the same conjunct. This we prove very useful in the remainder. For that, denote the set of satisfying assignments of F by $\mathrm{SAT}(F)$. Given a Boolean assignment $\alpha_{i,j}$ ($1 \leq i \leq p$ and $1 \leq j \leq m_i$) for F define

$$\varphi^{t,\epsilon_i}_{\bar{\alpha}_i}(\bar{x}) := \rho_{t,\epsilon_i}(\bar{x}) \wedge \bigwedge_{j:\alpha_{i,j}=\mathrm{true}} \varphi_{i,j}(\bar{x}) \wedge \bigwedge_{j:\alpha_{i,j}=\mathrm{false}} \neg\varphi_{i,j}(\bar{x}).$$

To put these formulas together, we let

$$\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x}) := \varphi^{t,\epsilon_1}_{\bar{\alpha}_1}(\bar{x} \restriction \epsilon_1) \wedge \dots \wedge \varphi^{t,\epsilon_p}_{\bar{\alpha}_p}(\bar{x} \restriction \epsilon_p) \wedge \bigwedge_{1 \leq i < j \leq p} d(\bar{x} \restriction \epsilon_i, \bar{x} \restriction \epsilon_j) > 2t, \tag{1}$$

where $d(\bar{a}, \bar{b}) > s$ means that for all $\nu, \mu : d(a_\mu, b_\nu) > s$. Using these definitions, it is not difficult to see that $\varphi(\bar{x})$ is equivalent to

$$\bigvee_{\epsilon} \bigvee_{(\bar{\alpha}_1,\dots,\bar{\alpha}_p) \in \mathrm{SAT}(F)} \varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x}), \tag{2}$$

which is of the desired form.

6 Including the tree cover

Adopt the assumptions at the end of the last section. The last equivalence gives us an alternative characterization of $\varphi(\mathfrak{A})$, i.e.

$$\varphi(\mathfrak{A}) = \bigcup_{\epsilon} \bigcup_{(\bar{\alpha}_1,\dots,\bar{\alpha}_p)} \varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\mathfrak{A}). \tag{3}$$

Since both unions are disjoint, it is convenient to assume henceforth that the input formula $\varphi(\bar{x})$ is of the form $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x})$ for some ϵ (cf. equation (1)), i.e.

$$\varphi(\bar{x}) = \varphi_1(\bar{x} \restriction \epsilon_1) \wedge \dots \wedge \varphi_p(\bar{x} \restriction \epsilon_p) \wedge \bigwedge_{1 \leq i < j \leq p} d(\bar{x} \restriction \epsilon_i, \bar{x} \restriction \epsilon_j) > 2t + 1, \tag{4}$$

with $\varphi_i(\bar{x} \restriction \epsilon_i)$ being t-local around $\bar{x} \restriction \epsilon_i$. To introduce the input structure appropriately, let $\mathcal{T} = \{U_1, \dots, U_m\}$ be a nice (r, l, g)-tree cover of the input structure $\mathfrak{A}$, for $r := k(2t+1)$ and appropriate $l \geq 1, g : \mathbb{N} \to \mathbb{N}$ (recall that k is the number of free variables).

We use capital letters for the indices $I \in [m]$ over the tree cover. These indices often occur as tuples, which will be indexed by lower-case letters (e.g. $I_j \in [m]$).

Including the cover:
For a set $X \subseteq A$ and $s \geq 1$ we define $K^s(X)$ (the s-kernel of X) to be the set of vertices $a \in X$ such that $N^{\mathfrak{A}}_s(a) \subseteq X$. We have chosen r to assure that, if for some $\bar{a}$ and J we have $\bar{a} \restriction \epsilon_i \cap K^r(U_J) \neq \emptyset$

then $\bar{a} \restriction \epsilon_i \subseteq K^t(U_J)$, hence $\langle U_J \rangle^{\mathfrak{A}} \models \varphi_i(\bar{a} \restriction \epsilon_i)$ iff $\mathfrak{A} \models \varphi_i(\bar{a} \restriction \epsilon_i)$. This follows from the t-locality of $\varphi_i(\bar{x} \restriction \epsilon_i)$. Now it is easy to see that

$$\varphi(\mathfrak{A}) = \bigcup_{(J_1,\dots,J_p)\in[m]^p} A_{J_1,\dots,J_p}, \tag{5}$$

where we let

$$A_{J_1,\dots,J_p} := \left\{ \bar{a} \;\middle|\; \bar{a} \restriction \epsilon_i \cap K^r(U_{J_i}) \neq \emptyset \text{ and } \langle U_{J_1} \cup \dots \cup U_{J_p} \rangle \models \varphi(\bar{a}) \right\}. \tag{6}$$

To capture the relevant topological information of a tree cover $\mathcal{T}$, we introduce the graph $\mathrm{c}(\mathcal{T}) := ([m], E^{\mathrm{c}(\mathcal{T})})$ with $E^{\mathrm{c}(\mathcal{T})} := \{(I,J) \mid U_I \cap U_J \neq \emptyset\}$. We let the *p-ctype* (contiguousness-type) of a tuple $(J_1,\dots,J_p)$ w.r.t. $\mathcal{T}$ be the graph $\kappa = ([p], E^\kappa)$ such that $(i,j) \in E^\kappa$ if, and only if $(J_i, J_j) \in E^{\mathrm{c}(\mathcal{T})}$ (and write $(J_1,\dots,J_p) \models \kappa$). By $\mathrm{Mod}^{\mathcal{T}}(\kappa)$ we denote the set of tuples of indices that *realize* κ: $\mathrm{Mod}^{\mathcal{T}}(\kappa) := \{(J_1,\dots,J_p) \mid (J_1,\dots,J_p) \models \kappa\}$.

Assume that we have given a tuple of indices $(J_1,\dots,J_p) \in [m]^p$ and that this tuple realize the p-ctype κ. It is easy to see that if i and j belong to different κ-components, then the distance between elements of their respective kernels $K^r(U_{J_i}), K^r(U_{J_j})$ is $> 2r$. This fact will be the key for the next characterization of $\varphi(\mathfrak{A})$.

Convention: In the remainder, let κ be a p-ctype with connected components $\kappa_1,\dots,\kappa_q$. For convenience, we arrange the coordinates of a tuple $(J_1,\dots,J_p)$ realizing κ according to the κ-components they belong to. Tacitly assuming that the ordering remains unchanged, we will write $(\bar{J}_1,\dots,\bar{J}_q)$ for the rearranged tuple; e.g. $A_{\bar{J}_1,\dots,\bar{J}_q}$ refers to the set defined in equation (6), although the indices are permuted. Furthermore, we appoint the convention that if we write J_j, this refers to the j'th coordinate of $\bar{J}$, while $J_{i,\nu}$ refers to ν'th coordinate contained in κ_i.

Let $\epsilon(\kappa_i) := \bigcup_{j\in\kappa_i} \epsilon_j$ be the union of the ϵ-components contained in κ_i. Then a straightforward restriction of the definition of $A_{J_1,\dots,J_p}$ (formula (6)) to the indices corresponding to the components $\epsilon(\kappa_i)$ gives rise to the following definition:

$$A^i_{\bar{J}_i} := \{\bar{a} \restriction \epsilon(\kappa_i) \mid \text{for all } j,\mu \text{ such that } j \text{ is the } \mu\text{'th element of } \kappa_i,$$
$$\bar{a} \restriction \epsilon_j \cap K^r(U_{J_{i,\mu}}) \neq \emptyset \text{ and } \mathfrak{A} \models \rho_{t,\epsilon(\kappa_i)}(\bar{x} \restriction \epsilon(\kappa_i)) \wedge \bigwedge_{j\in\kappa_i} \varphi_j(\bar{a} \restriction \epsilon_j)\}. \tag{7}$$

Observe that these sets can be computed in the substructure of $\mathfrak{A}$ induced by the set $U_{\bar{J}_i} := \bigcup_{\nu=1}^{w} U_{J_{i,\nu}}$, where w is the size of the tuple $\bar{J}_i$. As a matter of fact, $A^i_{\bar{J}_i}$ is the set of tuples satisfying the formula

$$\rho_{t,\epsilon(\kappa_i)}(\bar{x} \restriction \epsilon(\kappa_i)) \wedge \bigwedge_{j\in\kappa_i} \big(\varphi_j(\bar{x} \restriction \epsilon_j) \wedge \bigvee_{\nu\in\epsilon_j} P_\nu x_\nu\big),$$

where the P_ν are new unary relation symbols, interpreted by $K^r(U_{J_\nu})$. Observe that, by t-locality, this formula can be evaluated correctly[2] in $\langle U_{\bar{J}_i} \rangle$. These local parts $A^i_{\bar{J}_i}$ are the projections of $A_{\bar{J}_1,\dots,\bar{J}_q}$. The goal of the foregoing investigation was to show that the converse also holds.

Lemma 7. *Let κ be a ctype as above and $(\bar{J}_1,\dots,\bar{J}_q)$ a tuple of indices realizing κ. Then*

$$A_{\bar{J}_1,\dots,\bar{J}_q} = A^1_{\bar{J}_1} \times \dots \times A^q_{\bar{J}_q}.$$

[2] Note that, even if some $x_\nu \in \bar{x} \restriction \epsilon_i$ does not occur freely in φ_i, it must occur in the set. This is against the convention made in the preliminaries, but is necessary here.

Using this, we can rewrite equation (5) as follows:

$$\varphi(\mathfrak{A}) \;=\; \bigcup_{\kappa} \bigcup_{(\bar{J}_1,\dots,\bar{J}_q)\models\kappa} A_{\bar{J}_1,\dots,\bar{J}_q} \;=\; \bigcup_{\kappa} \bigcup_{(\bar{J}_1,\dots,\bar{J}_q)\models\kappa} A^1_{\bar{J}_1} \times \cdots \times A^q_{\bar{J}_q}.$$

This characterization already allows to build up $\varphi(\mathfrak{A})$ from parts that are locally computable. What remains is to provide some suitable characterization of the admissible index-tuples.

The contiguousness-graph:
Next we provide a characterization of the set of admissible indices $(\bar{J}_1, \dots, \bar{J}_q)$ in the foregoing equation. Let κ be a p-ctype with components $\kappa_1, \dots, \kappa_q$. The *cgraph* (contiguousness-graph) $\mathrm{c}(\mathcal{T}, \kappa)$ of $\mathcal{T}$ with respect to κ has vertex set $\mathrm{Mod}(\kappa_1) \dot\cup \cdots \dot\cup \mathrm{Mod}(\kappa_q)$ and edge relation $E^{\mathrm{c}(\mathcal{T},\kappa)}$ defined as follows:

$$(\bar{J}_i, \bar{J}_j) \in E^{\mathrm{c}(\mathcal{T},\kappa)} \text{ iff there are } \nu, \mu : (J_{i,\nu}, J_{j,\mu}) \in E^{\mathrm{c}(\mathcal{T})}.$$

Additionally, we give the vertices corresponding to $\mathrm{Mod}(\kappa_i)$ the color i, or more formally, we add unary relations $C_1, \dots, C_q$ with $C_i^{\mathrm{c}(\mathcal{T},\kappa)} := \mathrm{Mod}(\kappa_i)$. Then it is easy to see that $(\bar{J}_1, \dots, \bar{J}_q) \models \kappa$ iff $(\bar{J}_1, \dots, \bar{J}_q)$ is independent in $\mathrm{c}(\mathcal{T}, \kappa)$. So our problem essentially reduces to the problem of finding independent sets in $\mathrm{c}(\mathcal{T}, \kappa)$. This clearly depends on the combinatorial properties of $\mathrm{c}(\mathcal{T}, \kappa)$ of which the relevant ones are:

Lemma 8. *Let $\mathcal{T}$ be a (r, l, g)-tree cover of $\mathfrak{A}$. Then for all κ:*

(1) $\mathrm{c}(\mathcal{T}, \kappa)$ *can be calculated in time* $O(\|\mathcal{T}\|)$.
(2) $\mathrm{c}(\mathcal{T}, \kappa)$ *has maximum valence bounded by some function on* l, k.
(3) for all $i = 1, \dots, q$: $\{U_{\bar{J}_i} \mid \bar{J}_i \in C_i^{\mathrm{c}(\mathcal{T},\kappa)}\}$ *is a* $(r, l', g \circ h_i)$*-tree cover of* $\mathfrak{A}$, *where* $h_i(n) := |\kappa_i| \cdot n$ *and* l' *depends on* l *and* p.

7 The construction- and the listing problem

In this section we present both an algorithm for the FO-listing and the FO-construction problem. Essentially, we proceed as follows: w.l.o.g. we are given a t-local $\varphi(\bar{x}) \in \mathrm{FO}$ of the form $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x})$. The reduction to this setting was described previously. After computing a suitable tree cover $\mathcal{T}$ of the input structure $\mathfrak{A}$, we fix a ctype κ and compute $\mathrm{c}(\mathcal{T}, \kappa)$. Now elements of the sought set $\varphi(\mathfrak{A})$ roughly correspond to independent tuples of $\mathrm{c}(\mathcal{T}, \kappa)$.

Theorem 9. *Let $\mathcal{C}$ be a nicely locally tree-decomposable class of structures. Then* FO-EVALUATION *on $\mathcal{C}$ is in fixed-parameter linear time, i.e. there is an algorithm that for an* FO*-formula $\varphi(\bar{x})$ an a structure $\mathfrak{A} \in \mathcal{C}$ computes $\varphi(\mathfrak{A})$ in time*

$$f(\|\varphi\|) \cdot (\|\mathfrak{A}\| + |\varphi(\mathfrak{A})|),$$

for some function $f : \mathbb{N} \to \mathbb{N}$.

The algorithm evaluating FO-queries looks as follows:

Algorithm 1: Computing $\varphi(\mathfrak{A})$

Input: Structure $\mathfrak{A} \in \mathcal{C}$, FO-formula $\varphi(\bar{x})$
Output: $\varphi(\mathfrak{A})$

1. Compute the Boolean combination $\varphi'(\bar{x}) = F(\varphi_1(\bar{x}), \dots, \varphi_{l_1}(\bar{x}), \psi_1, \dots, \psi_{l_2})$ equivalent to $\varphi(\bar{x})$, where all $\varphi_i(\bar{x})$ are t-local around $\bar{x}$.
2. For all $i = 1, \dots, l_2$ decide, if $\mathfrak{A} \models \psi_i$ and replace ψ_i in $\varphi'(\bar{x})$ with its truth value. Denote the resulting formula with $\varphi''(\bar{x})$.
3. Compute a nice $(k(2t+1), l, g)$-tree cover $\mathcal{T}$ for appropriate $l \geq 1, g : \mathbb{N} \to \mathbb{N}$.
4. For all distance-types $\epsilon = \epsilon_1 \cup \dots \cup \epsilon_p$
   i calculate $F(\bar{\varphi}_1(\bar{x}), \dots, \bar{\varphi}_p(\bar{x}))$ from lemma 6 for the formula $\varphi''(\bar{x})$
   ii For all $(\bar{\alpha}_1, \dots, \bar{\alpha}_p) \in \mathrm{SAT}(F)$, compute $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\mathfrak{A})$
5. Return $\bigcup_\epsilon \bigcup_{\bar{\alpha}} \varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\mathfrak{A})$

The first line needs time linear in $\|\varphi'(\bar{x})\|$, which itself only depends on $\|\varphi(\bar{x})\|$.

By definition, a nice $(k(2t+1), l, g)$-tree cover can be computed in time $O(\|\mathfrak{A}\|)$, the same holds for the decision of $\mathfrak{A} \models \chi_i$ (by theorem 4). Thus line 2 and line 3 need time linear in the structure size.

The loop in line 4 is the crucial one. First, the number of different ϵ and $(\bar{\alpha}_1, \dots, \bar{\alpha}_p)$ is constant (depends only on k). Hence, the subroutine computing $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\mathfrak{A})$ is called constantly often. Calculating $\varphi^{t,\epsilon}(\bar{x})$ in line 4(i) needs time only depending on $\|\varphi\|$.

Now consider line 4(ii), where we compute $\varphi^{t,\epsilon}_{\bar{\alpha}}(\mathfrak{A})$. This set coincides with $\bigcup_{(\bar{J}_1,\dots,\bar{J}_q)} A^1_{\bar{J}_1} \times \dots \times A^q_{\bar{J}_q}$, where the tuples admitted in the union are those independent in $\mathcal{G} := \mathrm{c}(\mathcal{T}, \kappa)$. So we proceed as follows: in a first step, we compute $\mathcal{G}$ (by lemma 8 this takes linear time), remove all vertices $\bar{J}_i \in C^{\mathcal{G}}_i$ with $A^i_{\bar{J}_i} = \emptyset$ and get $\mathcal{G}'$ (since the sets $U_{\bar{J}_i}$ do not intersect very much, this can be done in linear time by an application of theorem 1 and lemma 8). Then in a second step, we collect all $(\bar{J}_1, \dots, \bar{J}_q)$ independent in $\mathcal{G}'$. Since $\mathcal{G}$ has bounded valence, this can be done in time linear in the number of independent tuples. Now, computing each set $A^i_{\bar{J}_i}$ only once, we build the sought union over the Cartesian products. Altogether this takes time $O(|\varphi(\mathfrak{A})|)$. The crucial obervations to obtain this tight time bound are (i) that each $\bar{a} \in \varphi(\mathfrak{A})$ is contained in at most a constant number of sets $A_{\bar{J}_1,\dots,\bar{J}_q}$ and (ii) that each $A^i_{\bar{J}_i}$ is only computed once. □

The task to find a witness for a formula reduces to finding an independent tuple $(\bar{J}_1, \dots, \bar{J}_q)$ in the cgraph $\mathrm{c}(\mathcal{T}, \kappa)$ for some κ.

Theorem 10. *Let $\mathcal{C}$ be a nicely tree-decomposable class of structures. Then* FO-CONSTRUCTION *on $\mathcal{C}$ is in fixed-parameter linear time.*

Proof: The algorithm resembles very much the one of the listing case (and we omit a separate presentation). Only the loop starting at line 4 has to be modified. Instead of computing the entire sets $\varphi^{t,\epsilon}_{\bar{\alpha}}(\mathfrak{A})$ now we just have to look for a satisfying tuple.

As we already saw in the listing case, satisfying tuples correspond to independent tuples in the contiguousness-graph, and one of these can be found in linear time in the size of the cgraph.

Now it is easy to find a tuple of $\varphi^{t,\epsilon}_{\bar{\alpha}}(\mathfrak{A})$. Again, we compute $\mathcal{G} := \mathrm{c}(\mathcal{T}, \kappa)$ and remove all vertices $\bar{J} \in C^{\mathcal{G}}_i$ such that $A^i_{\bar{J}} = \emptyset$. This takes time linear in $\mathfrak{A}$ and gives a graph $\mathcal{G}'$. Then we find a independent tuple $(\bar{J}_1, \dots, \bar{J}_q)$, if one exists, and return an arbitrary $\bar{a} \in \prod_{i=1}^q A^i_{\bar{J}_i}$. □

8 The counting problem

This last section is dedicated to the counting problem. Remember that we assume that arithmetical operations can be done in constant time.

Theorem 11. *Let $\mathcal{C}$ be a nicely tree-decomposable class of structures. Then* FO-COUNTING *on $\mathcal{C}$ is in fixed-parameter linear time.*

Proof: We start with an immediate consequence of formula (3) for t-local $\varphi(\bar{x})$:

$$|\varphi(\mathfrak{A})| = \sum_\epsilon \sum_{(\bar{\alpha}_1,\dots,\bar{\alpha}_p)} |\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\mathfrak{A})|.$$

Since there are at most constant many different $\epsilon, \bar{\alpha}$ we again restrict our attention to formulas of the form $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x})$. Let $\varphi(\bar{x})$ be of this form, for a distance type ϵ with components $\epsilon_1, \dots, \epsilon_p$ and a suitable Boolean assignment $(\bar{\alpha}_1, \dots, \bar{\alpha}_p)$. Like always, k denotes the number of free variables of $\varphi(\bar{x})$ and let $\mathcal{T} = \{U_1, \dots, U_m\}$ be a nice (r, l, g)-tree cover for $\mathfrak{A}$ (again we choose $r := k(2t+1)$).

We let $A_{\mathcal{J}} := \bigcap_{J \in \mathcal{J}} A_J$ for an index set $\mathcal{J}$. The next equation characterizes the cardinality of $\varphi(\mathfrak{A})$, and is a consequence of equation (5) and the principle of inclusion and exclusion.

$$|\varphi(\mathfrak{A})| \overset{(5)}{=} |\bigcup_{(J_1,\dots,J_p)\in[m]^p} A_{J_1,\dots,J_p}| \overset{\text{PIE}}{=} \sum_{\emptyset \neq \mathcal{J} \subseteq [m]^p} (-1)^{|\mathcal{J}|+1} |A_{\mathcal{J}}| \tag{8}$$

To be able to separate results of remote parts, we pass from sets of indices to their projections. Formally, for $\emptyset \neq \mathcal{J} \subseteq [m]^p$ we define $\mathcal{J} \restriction j := \{J_j \mid \text{ there is a } (J_1, \dots, J_j, \dots, J_p) \in \mathcal{J}\}$, the projection of $\mathcal{J}$ onto coordinate j. The natural extension of the definition of $A_{J_1,\dots,J_p}$ (formula (6)) to sets of indices looks as follows:

$$A_{\mathcal{J}_1,\dots,\mathcal{J}_p} := A_{\mathcal{J}_1 \times \cdots \times \mathcal{J}_p}.$$

Observe here that we have $A_{\mathcal{J}} = A_{\mathcal{J}\restriction 1,\dots,\mathcal{J}\restriction p}$ for all $\mathcal{J} \subseteq [m]^p$. Since we want to proceed like in the listing case, we perform a couple of arithmetical reductions until we can separate independent parts again (as we did before). Essentially, we group the indices $\mathcal{J}$ of the sum (8) into classes, such that all elements $\mathcal{J}$ of the same class induce the same tuple $(\mathcal{J}\restriction 1, \dots, \mathcal{J}\restriction p)$. Since for such tuples we get a unique value $|A_{\mathcal{J}}|$, we have to consider each of these only once (of course weighted by the size of the class).

For these separate coordinates we can establish a contiguousness relation, very much like in the listing case. Informally, we get $|A_{\mathcal{J}\restriction 1,\dots,\mathcal{J}\restriction p}| = |A_{\bar{\mathcal{J}}_1}| \cdots |A_{\bar{\mathcal{J}}_q}|$ for tuples $\mathcal{J}_1, \dots, \mathcal{J}_p$ satisfying an (extended) ctype κ (compare lemma 7 in the listing case) with q components. Altogether, this allows a rewriting of our starting problem as a sum of products of values corresponding to tuples independent in the new contigousness-graph (the Cartesian products in the listing case translate to multiplications). This reduction is formalized in the next lemma.

Lemma 12. *There is a linear time algorithm that computes, given a formula $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x})$ and a tree cover $\mathcal{T}$ as above, a colored graph $(\mathcal{G}, C_1, \dots, C_q)$ of bounded valence and integers $\gamma(v) \in \mathbb{N}$ for all $v \in G$ such that*

$$|\varphi(\mathfrak{A})| = \sigma(\mathcal{G}, \gamma) := \sum_{\substack{\bar{v} \text{ independent} \\ v_i \in C_i}} \gamma(v_1) \cdots \gamma(v_q).$$

The next theorem fills the gap still missing in the proof of theorem 11

Theorem 13. *There is an algorithm that, given $\mathcal{G}$ and γ as in the last lemma, computes $\sigma(\mathcal{G}, \gamma)$ in time $O(\|\mathcal{G}\|)$.*

If we plug in this algorithm into the gap left open above, then the proof of theorem 11 is completed.

9 Concluding remarks

In our algorithms there are various bottlenecks. Generally, it can be said that there is no elementary upper bound for the dependency on the input formula. This bad news makes the algorithm almost useless for practice.

So the main contribution of this work can be seen as a meta-theorem in the style of Courcelle's theorem for more general classes of structures. That is, we give means to recognize whether a certain problem is linear time computable on certain structures. And once we recognized a problem being solvable in linear time, problably we can find practical algorithms by analyzing the specific combinatorial properties of the problem.

The work can also be seen as the completion of the model-checking problem for first-order logic over locally-tree decomposable classes which was initiated in [12].

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.
2. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
3. B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 265–273, 1983.
4. Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.
5. B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, pages 194–242. Elsevier Science Publishers, 1990.
6. R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer-Verlag, 1999.
7. R.G. Downey, M.R. Fellows, and U. Taylor. On the parametric complexity fo relational databases queries and a sharper characterization of $W[1]$. In D. Bridges, C. Calude, J. Gibbons, S. Reeves, and I. Witten, editors, *Combinatorics, Complexity and Logic*, DMTCS, pages 194–213. Springer-Verlag, 1996.
8. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer Verlag, 1995.
9. D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 1999.
10. J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 22–38. Springer, 2001.
11. M. Frick. *Easy Instances for Model Checking*. PhD thesis, Universität Freiburg, http://www.freidok.uni-freiburg.de/volltexte/229, 2001.
12. M. Frick and M. Grohe. Checking first-order properties of tree-decomposable graphs. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *26th International Colloquium, ICALP'99, Prague*, volume 1644 of *Lecture Notes in Computer Science*, pages 331–340. Springer, 1999.
13. H. Gaifman. On local and non-local properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.
14. M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.
15. M. Grohe. Generalized model-checking problems for first-order logic. In A.Ferreira and H.Reichel, editors, *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *Lecture Note in Computer Science*, pages 12–26. Springer Verlag, Berlin, 2001.
16. M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.
17. L.G. Valiant. The complexity of combinatorial computations: An introduction. In *GI - 8. Jahrestagung*, number 16 in Informatik-Fachberichte, pages 326–337. Springer, 1978.
18. M.Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146, 1982.

A Appendix - the proofs

A.1 Proof of lemma 7:

Let κ be given with components $\kappa_1, \dots, \kappa_q$ and $(\bar{J}_1, \dots, \bar{J}_q)$ be a tuple of indices realising κ. First, note that $\bar{a} \restriction \epsilon_j \cap K^r(U_{J_j}) \neq \emptyset$ for all $j \in [p]$ must be satisfied on both sides of the equation, hence in the sequel we assume that it holds. Under this condition, $\bar{a} \in A_{\bar{J}_1,\dots,\bar{J}_q}$ is equivalent to $\mathfrak{A} \models \rho_{t,\epsilon}(\bar{a}) \wedge \varphi_1(\bar{a} \restriction \epsilon_1) \wedge \dots \wedge \varphi_p(\bar{a} \restriction \epsilon_p)$. This implies $\bar{a} \restriction \epsilon(\kappa_i) \in A^i_{\bar{J}_i}$ (equation (7)) and proves the first inclusion.

To prove the other direction, consider a tuple $\bar{a}$ with $\bar{a} \restriction \epsilon(\kappa_i) \in A^i_{\bar{J}_i}$, for all $i = 1, \dots, q$. We show that for j_1, j_2 from different κ-components, remoteness is satisfied, i.e. $d(\bar{a} \restriction \epsilon_{j_1}, \bar{a} \restriction \epsilon_{j_2}) > 2t + 1$. This gives $\bar{a} \models \epsilon$, since $\bar{a} \restriction \epsilon(\kappa_i) \models \epsilon(\kappa_i)$, for all i, is explicitly claimed in the definition of $A^i_{\bar{J}_i}$.

For $j_1 \neq j_2$ there are $\nu_1 \in \epsilon_{j_1}, \nu_2 \in \epsilon_{j_2}$ such that $a_{\nu_1} \in K^r(U_{J_{j_1}})$ and $a_{\nu_2} \in K^r(U_{J_{j_2}})$. Using the definition of kernels this gives us the inequality $d(a_{\nu_1}, a_{\nu_2}) > 2r = 2k(2t+1)$. On the other hand, the connectedness of the ϵ-components yields $\bar{a} \restriction \epsilon_{j_1} \subseteq N_{k(2t+1)}(a_{\nu_1})$, $\bar{a} \restriction \epsilon_{j_2} \subseteq N_{k(2t+1)}(a_{\nu_2})$, which together give us the desired inequality $d(\bar{a} \restriction \epsilon_{j_1}, \bar{a} \restriction \epsilon_{j_2}) > 2t + 1$ (since either of the ϵ-components has length less than k).
The definition of $A^i_{\bar{J}_i}$ claims

$$\langle U_{\bar{J}_i} \rangle \models \rho_{t,\epsilon(\kappa_i)}(\bar{a} \restriction \epsilon(\kappa_i)) \wedge \bigwedge_{j \in \kappa_i} \varphi_j(\bar{a} \restriction \epsilon_j),$$

for all $i = 1, \dots, q$, hence $\mathfrak{A} \models \rho_{t,\epsilon}(\bar{x}) \wedge \varphi_1(\bar{a} \restriction \epsilon_1) \wedge \dots \wedge \varphi_p(\bar{a} \restriction \epsilon_p)$. □

A.2 Proof of lemma 8:

Let $\mathcal{T} = \{U_1, \dots, U_m\}$ be an (r, l, g)-tree cover of $\mathfrak{A}$ and κ a ctype with components $\kappa_1, \dots, \kappa_q$. Denote the size of κ_i by p_i and recall that $\mathrm{c}(\mathcal{T})$ has maximal valence $\leq l$. Let $J \in [m]$ and $i \leq q$. We define $\kappa_i(J) := \{\bar{J} \mid \bar{J} \models \kappa_i \text{ and } J_j = J \text{ for some } j\}$, the set of all vertices in $C_i^{\mathrm{c}(\mathcal{T},\kappa)}$ that contain J as a part.

For the proof of (1) consider algorithm 2 computing $\mathrm{c}(\mathcal{T}, \kappa)$. We treat the different colors one after another. For each color, we first generate the set of vertices, and then append to each of them its adjacency list. At the end of the algorithm, $\mathrm{c}(\mathcal{T}, \kappa)$ is contained in the arrays `vertex` and `adj`. Their semantics is as follows: $\texttt{vertex}[i, j] = \bar{J}$ means that the j'th vertex of color i is $\bar{J}$. $\texttt{adj}[i, j]$ contains the list of the vertices adjacent to the $\bar{J}$. Note that these arrays have only one unbounded dimension (the one that corresponds to the index j). The others are bounded, hence the arrays are essentially one dimensional. The lists generated by the procedure may contain multiples, but by an application of the well-known radix-sort, these can be eliminated in linear time. The correctness of the algorithm is immediate.

The running time: For the linear time bound, note that $|\kappa_i(J)| \leq (l^{p_i})^{p_i - 1} = l^{p_i(p_i-1)}$ for all $J \in [m]$; we simply choose p_i many indices from the $(p_i - 1)$-neigborhood (in $\mathrm{c}(\mathcal{T})$) of J. Thus every loop, apart from the loop over J, is constant. Hence the algorithm needs time $O(m)$. Implicitly, this forces $\mathrm{c}(\mathcal{T}, \kappa)$ to have linear size, in fact, $|C_i^{\mathrm{c}(\mathcal{T})}| \leq \sum_{J \in \mathrm{c}(\mathcal{T})} |\kappa_i(J)| \leq m \cdot l^{p_i(p_i-1)} = O(m)$ (this is the maximum value of the variable n).

To prove (2) consider the loop from line 9 to line 17. Recall that $N_1(J)$ is the set of all vertices adjacent to J. In this loop, we generate the adjacency list for each vertex $\bar{J}$, which has size $\leq p_i \cdot l \cdot l^{p_i(p_i-1)}$.

At last, for (3), we have to confirm the conditions of definition 3. Condition (1) transfers to covers with the property that each set in the old cover is contained in a set of the new cover.

```
1  proc calc_cgraph(T, κ)
2    for i = 1 to q do  /* the colors */
3      n := 1;
4      for J = 1 to m do
5        for J̄ ∈ κ_i(J) do
6          vertex[i, n] := J̄;
7          comment: calculate the adjacency list
8          adj[i, n] := NULL;
9          for μ = 1 to p_i do
10           for I ∈ N_1^{c(T)}(J_μ) do
11             for j = 1 to q do
12               for Ī ∈ κ_j(I) do
13                 append(adj[i, n], Ī);
14               od
15             od
16           od
17         od
18         n := n + 1;
19       od
20     od
21   od
22 .
```

Algorithm 2. Compute $\mathrm{c}(\mathcal{T}, \kappa)$

Furthermore, there is an edge between $\bar{J}$ and $\bar{J}'$ iff $U_{\bar{J}} \cap U_{\bar{J}'} \neq \emptyset$, hence part (2) of this lemma proves property (2) of definition 3 (a bound on the non-empty intersections).

To examine the treewidth, fix some i and $\bar{J} \models \kappa_i$. Because $\mathcal{T}$ is a nice (r, l, g)-tree cover, the set $\langle U_{\bar{J}} \rangle$ has treewidth bounded by $g(p_i)$. Building the union over s such sets we get $g(p_i \cdot s)$ as an upper bound on the treewidth what confirms condition (3). □

A.3 Exact proof of theorem 9:

Adopt the assumptions of algorithm 1 before the call of the subroutine in line 4ii, which does the main computation. That is, we have given a formula $\varphi(\bar{x})$ of the form $\varphi^{t,\epsilon}_{\bar{\alpha}_1,\dots,\bar{\alpha}_p}(\bar{x})$ (recall formula (4)), and an (r, l, g)-tree cover $\mathcal{T}$ of $\mathfrak{A}$. The procedure `enum_assign` that enumerates all tuples $\bar{a}$ with $\mathfrak{A} \models \varphi(\bar{a})$ is displayed as algorithm 3.

Correctness: `enum_assign` first calculates $\mathcal{G} := \mathrm{c}(\mathcal{T}, \kappa)$ and then eliminates all vertices $\bar{J} \in C_i^{\mathrm{c}(\mathcal{T},\kappa)}$ with $A^i_{\bar{J}} = \emptyset$. Note that a Cartesian product is empty if, and only if, one of its parts is empty. This means that we throw away those indices that, in any case, do not contribute to the result. The subsequent call of `indep_tuples`$(\mathcal{G})$ (displayed as algorithm 4) then computes the set of independent tuples $\mathcal{I}$ (line 9), which directly provides the set $\bigcup_{(\bar{J}_1,\dots,\bar{J}_q)\in\mathcal{I}} A^1_{\bar{J}_1} \times \cdots \times A^q_{\bar{J}_q}$. This proves the correctness of the algorithm. And as it will turn out, this, performed in a careful way (lines 10 to 19), is enough to obtain an optimal time bound.

Running time: Concerning the running time, we have to go more into details: Note that the outermost loop over κ is passed through a constant number of times. So consider the first part between line 4 and line 8: by definition, $A^i_{\bar{J}_i} = \emptyset$ can be decided in the structure $\langle U_{\bar{J}_i} \rangle$, which, applying the additional property of nice tree covers, has bounded treewidth (by $g(|\kappa_i|)$, cf. lemma 8). Hence, we can apply the algorithm of theorem 1 and decide emptiness in time $f'(\|\varphi\|) \cdot |U_{\bar{J}_i}|$, for some $f' : \mathbb{N} \to \mathbb{N}$. Furthermore, each $a \in A$ is contained in at most l sets U_J, hence $\sum_{\bar{J}_i \in C_i^{\mathcal{G}}} |U_{\bar{J}_i}| \leq l^{|\epsilon(\kappa_i)|} \cdot |A|$ for all $i \in [q]$. Therefore, the running time for this loop is linear in the size of $\mathfrak{A}$. From

```
1  proc enum_assign(𝒯, φ(x̄))
2    for κ a p-contiguousness-type do
3      𝒢 := c(𝒯, κ);
4      for i = 1 to q do
5        for J̄ ∈ C_i^𝒢 do
6          if A^i_J̄ = ∅ then G := G \ {J̄} fi
7        od
8      od
9      ℐ := indep_tuples(𝒢);
10     for i = 1 to q do
11       ℐ_i := ℐ ↾ i;
12       for J̄ ∈ ℐ_i do
13         compute A^i_J̄;
14       od
15     od
16     B := ∅;
17     for (J̄_1, ..., J̄_q) ∈ ℐ do
18       B := B ∪ A^1_J̄1 × ··· × A^q_J̄q;
19     od
20   od
21   return(B);
22 .
```

Algorithm 3. Evaluating a t-local formula

now on, $\mathcal{G}$ denotes the graph after removing these "superflous" vertices. $\mathcal{G}$ still has bounded valence (recall lemma 8(2)), say, by d.
In line 9, we call the function `indep_tuples`$(\mathcal{G})$ (algorithm 4) to compute the set of independent tuples of $\mathcal{G}$.

Claim: `indep_tuples`$(\mathcal{G})$ returns all independent tuples of $\mathcal{G}$ and needs time $O(|\texttt{indep\_tuples}(\mathcal{G})|)$.

Proof of the Claim: Consider algorithm 4 and assume that $\mathcal{G}$ has maximal valence d (it is an induced subgraph of a contiguousness-graph). In a first loop, we partition the set of indices into "big" and "small" ones (lines 4 to 8). The actual computation proceeds in two steps. In the first step, we look exhaustively for all independent tuples over small indices (note that there are at most $(2dq)^q$ many of them).

Then, in the second step, we extend these "partial" tuples to tuples over $[q]$, which gives the sought result. The code should be clear.

Let us estimate the time necessary for the second part of the search (the first part trivially needs constant time): for simplicity, we assume that all indices are big; the set of all possible tuples of indices $(\bar{J}_1, \ldots, \bar{J}_q)$ has size $\prod_{i=1}^q |C_i^{\mathcal{G}}|$. A lower bound for the number of independent tuples is easily seen to be $|C_1^{\mathcal{G}}| \cdot (|C_2^{\mathcal{G}}| - d) \cdots (|C_q^{\mathcal{G}}| - (q-1)d)$. Since our loop goes through all tuples in $\prod_{i=1}^q C_i^{\mathcal{G}}$, it is enough to show that there is a $c > 0$ (actually, we choose $c := 2^q$) such that

$$\prod_{i=1}^{q} |C_i| \leq c \cdot \prod_{i=1}^{q} (|C_i| - dq).$$

To prove this, observe that by $|C_i| > 2dq$ we have $|C_i| \leq 2 \cdot (|C_i| - dq)$. Applied to the above situation we get

$$\prod_{i=1}^{q} |C_i| \quad \leq \quad 2(|C_1| - dq) \cdot \prod_{i=2}^{q} |C_i| \quad \leq \quad \cdots \quad \leq \quad 2^q \prod_{i=1}^{q} (|C_i| - dq)$$

```
1  proc indep_tuples(𝒢)
2    ℐ := ∅;
3    d := max-valence(𝒢); big = NULL; small = NULL;
4    for i = 1 to q do
5        if |C_i^𝒢| > 2dq
6          then append(big, i) else append(small, i)
7        fi
8    od
9    if small = NULL
10     then
11           ℐ_s := {()} else
12           ℐ_s :=  all independent v̄ ∈ ∏_{i∈small} C_i^𝒢
13   fi
14   for v̄ ∈ ℐ_s do
15       for ū ∈ ∏_{i∈big} C_i^𝒢
16           if (v̄, ū) independent
17             then ℐ := ℐ ∪ {(v̄, ū)}
18           fi
19       od
20   od
21   return(ℐ);
22 .
```

Algorithm 4. Calculate all independent tuples

as an upper bound on the number of times the loop (between lines 14 and 20) is passed. Checking, if a tuple $(\bar{J}_1, \ldots, \bar{J}_q)$ is independent can be done in constant time (a simple lookup in q adjacency lists, all of constant size). Together, we need time $\prod_{i=1}^{q} |C_i^{\mathcal{G}}| = O(\prod_{i=1}^{q}(|C_i| - dq))$. □

So let us revert to the analysis of the running time of `enum_assign`. We just computed the set $\mathcal{I}$ of independent tuples of $\mathcal{G} = \mathrm{c}(\mathcal{T}, \kappa)$ in time $O(|\mathcal{I}|)$ (line 9). Because $A^i_{\bar{J}} \neq \emptyset$, for all $\bar{J} \in C_i^{\mathcal{G}}$, the set $\mathcal{I}$ has at most $\sum_{(\bar{J}_1,\ldots,\bar{J}_q)\in\mathcal{I}} |A^1_{\bar{J}_1} \times \cdots \times A^q_{\bar{J}_q}| = O(|\varphi(\mathfrak{A})|)$ elements. Hence the loop needs $O(|\varphi(\mathfrak{A})|)$ steps.

Now consider the subsequent loop for each i (lines 10 to 15): first, we calculate the projection of $\mathcal{I}$ onto coordinate i. This can be done in $O(|\mathcal{I}|)$ steps. A bit more involved, and actually the reason for the elimination of some vertices of $\mathcal{G}$ is the loop over $\mathcal{I}_i$. Observe that $A^i_{\bar{J}_i}$ can be calculated in $\langle U_{\bar{J}_i} \rangle$. This substructure has bounded treewidth (recall the first loop), thus we can calculate $A^i_{\bar{J}_i}$ in time $O(|U_{\bar{J}_i}| + |A^i_{\bar{J}_i}|)$ (recall theorem 1). Hence, the entire loop requires

$$\sum_{\bar{J}_i \in \mathcal{I}_i} \Big(|U_{\bar{J}_i}| + |A^i_{\bar{J}_i}| \Big) = O(|A| + \sum_{\bar{J}_i \in \mathcal{I}_i} |A^i_{\bar{J}_i}|)$$

steps (for the equality, cf. the first loop). The crucial step in the running time analysis is to show that $\sum_{\bar{J}_i \in \mathcal{I}_i} |A^i_{\bar{J}_i}| = O(|\varphi(\mathfrak{A})|)$: For a tuple $\bar{b}$ let $I(\bar{b}) := \{\bar{J}_i \mid \bar{b} \in U_{\bar{J}_i}\}$ be the number of times a $\bar{b}$ is computed and assume $\bar{J}_i \in I(\bar{b})$. This entails for all $\nu : \bar{J}_i \cap \{J \mid b_\nu \in U_J\} \neq \emptyset$, hence

$$|I(\bar{b})| \leq \Big(|\kappa_i| \cdot l^{|\epsilon(\kappa_i)|} \Big)^{|\kappa_i|}.$$

Thus, each tuple occurring in $\bigcup_{\bar{J}_i \in \mathcal{I}_i} A^i_{\bar{J}_i}$ is computed at most a constant number times, i.e.

$$\sum_{\bar{J}_i \in \mathcal{I}_i} |A^i_{\bar{J}_i}| \leq \Big(|\kappa_i| \cdot l^{|\epsilon(\kappa_i)|} \Big)^{|\kappa_i|} \cdot |\bigcup_{\bar{J}_i \in \mathcal{I}_i} A^i_{\bar{J}_i}|.$$

The same argument gives us: $\mathcal{I} = O(|\varphi(\mathfrak{A})|)$. Finally, we injectively map each $\bar{b} \in \bigcup_{\bar{J}_i \in \mathcal{I}_i} A^i_{\bar{J}_i}$ to the lexicographically minimal $\bar{a} \in \varphi(\mathfrak{A})$ such that $\bar{b} = \bar{a} \restriction \epsilon(\kappa_i)$, hence the size of this union is $O(|\varphi(\mathfrak{A})|)$, which show or claim. Such a minimal $\bar{a}$ exists, since (i) we removed all $\bar{J}_i$ with empty $A^i_{\bar{J}_i}$ and (ii) in $\mathcal{I}_i$ there are only indices, which are part of an independent tuple (actually, this was the reason for calculating the projections $\mathcal{I}_i$). For else, we would possibly compute "big" sets $A^i_{\bar{J}_i}$ that do not contribute to $\varphi(\mathfrak{A})$ (because $\bar{J}_i$ is not part of an independent tuple). This completes the proof that the loop between lines 10 and 15 obeys the time bound.

To finish, we sum up the times we just estimated. We have $c_1 \cdot |A|$ for the first loop, and $O(|\varphi(\mathfrak{A})|)$ for the second one. As we have seen, the last two loops need time $O(|\varphi(\mathfrak{A})|)$. The same holds for the last loop, which completes the analysis. □

A.4 Proof of lemma 12:

To prove lemma 12 we proceed as follows: first, we rewrite equation (8) using the principle of inclusion and exclusion. Then, after some arithmetical transformations and helping lemmas, we again introduce contiguousness-types to structure the tree cover. In a last step, we compute the colored graph $\mathcal{G}$ (an extended contiguousness-graph) and the labelling γ.

We rewrite equation (8) as follows:

$$\begin{aligned}
|\varphi(\mathfrak{A})| &= \sum_{j=1}^{m^p} (-1)^{j+1} \sum_{\mathcal{J} \subseteq [m]^p, |\mathcal{J}|=j} |A_{\mathcal{J}}| \\
&= \sum_{j=1}^{m^p} (-1)^{j+1} \sum_{\mathcal{J}_1,\dots,\mathcal{J}_p \subseteq [m]} \sum_{\substack{\mathcal{J}, |\mathcal{J}|=j \\ \mathcal{J} \restriction i = \mathcal{J}_i}} |A_{\mathcal{J}}| \qquad (9) \\
&= \sum_{\mathcal{J}_1,\dots,\mathcal{J}_p \subseteq [m]} \sum_{j=1}^{|\mathcal{J}_1| \cdots |\mathcal{J}_p|} (-1)^{j+1} \cdot \mu_j(|\mathcal{J}_1|, \dots, |\mathcal{J}_p|) \cdot |A_{\mathcal{J}_1,\dots,\mathcal{J}_p}|, \qquad (10)
\end{aligned}$$

for suitable functions $\mu_j : [m]^p \to \mathbb{N}$, counting the number of $\mathcal{J}$ such that for all i: $\mathcal{J} \restriction i = \mathcal{J}_i$ for given $\mathcal{J}_1, \dots, \mathcal{J}_p$. The first two equations are easy confirmed, for the third one the functions μ_j are chosen suitably (recall that if $\mathcal{J} \restriction i = \mathcal{J}' \restriction i$ for all i then $A_{\mathcal{J}} = A_{\mathcal{J}'}$).

The next lemma gathers the properties relevant to evaluate formula (10). Recall that l denotes the maximum valence of $\mathrm{c}(\mathcal{T})$.

Lemma 14. *Let $\mathcal{J}_1, \dots, \mathcal{J}_p$ be subsets of $[m]$. If $A_{\mathcal{J}_1,\dots,\mathcal{J}_p} \neq \emptyset$ then $|\mathcal{J}_i| \leq k \cdot l$ for all $i = 1, \dots, p$. For $j > (k \cdot l)^p$ the function μ_j vanishes, and for $j \leq (k \cdot l)^p$, μ_j is a function from $[k \cdot l]^p$ to $[2^{(k \cdot l)^p}]$.*

Proof: Let $\mathcal{J}_1, \dots, \mathcal{J}_p$ be given and assume that $|\mathcal{J}_i| > k \cdot l$ for an i. By contradiction, assume further that there is an $\bar{a} \in A_{\mathcal{J}_1,\dots,\mathcal{J}_p}$, particularly, $\bar{a} \restriction \epsilon_i \cap U_{J_i} \neq \emptyset$ for all $J_i \in \mathcal{J}_i$ and $\mathfrak{A} \models \varphi(\bar{a})$. But each coordinate of $\bar{a} \restriction \epsilon_i$ may be contained in at most l neighborhoods, summing up to at most $|\epsilon_i| \cdot l \leq k \cdot l$ neighborhoods (under the assumption that all coordinates lie in different neighborhoods). This contradicts the assumption $|\mathcal{J}_i| > k \cdot l$.

Assume $|\mathcal{J}| > (k \cdot l)^p$. Then there must exist an i such that $|\mathcal{J} \restriction i| > k \cdot l$, hence $A_{\mathcal{J}} = \emptyset$ (by the first claim).

For the last claim $\mathcal{J}_1, \dots, \mathcal{J}_p \subseteq [m]$ are given. By the first claim, all $\mathcal{J}_i$ have $\leq k \cdot l$ elements. So fix the cardinalities $m_i := |\mathcal{J}_i|$. We are looking for the number of sets $\mathcal{J} \subseteq [m]^p$ such that $\mathcal{J} \restriction i = \mathcal{J}_i$, for all $i = 1, \dots, p$. If $(J_1, \dots, J_p) \in \mathcal{J}$ then $J_i \in \mathcal{J}_i$, hence there are at most $m_1 \cdots m_p \leq (k \cdot l)^p$ many such tuples, hence less than $2^{(k \cdot l)^p}$ sets of such tuples. This proves the lemma. □

This lemma implies that (i) the outer sum goes over index sets $\mathcal{J}_i$ of size $\leq k \cdot l$, and (ii) the inner sum only has a constant number of addends. Furthermore, the functions μ_j can be calculated in time only depending on k, l and p.

We continue as in the listing case, and introduce the *ctype* of a tuple $(\mathcal{J}_1, \dots, \mathcal{J}_p) \in \mathrm{Pow}([m])^p$ w.r.t $\mathcal{T}$. Recall the definition of $\mathrm{c}(\mathcal{T})$. We define $\tilde{\mathrm{c}}(\mathcal{T})$ to be the graph $(\mathrm{Pow}^{\leq kl}([m]) \setminus \{\emptyset\}, E^{\tilde{\mathrm{c}}(\mathcal{T})})$ where

$$(\mathcal{I}, \mathcal{J}) \in E^{\tilde{\mathrm{c}}(\mathcal{T})} \text{ iff for all } I \in \mathcal{I} \text{ and } J \in \mathcal{J} : (I, J) \in \mathrm{c}(\mathcal{T}).$$

The *p-ctype* of a tuple $(\mathcal{J}_1, \dots, \mathcal{J}_p) \in \mathrm{Pow}([m])^p$ w.r.t. $\mathcal{T}$ is the graph $\kappa := ([p], E^\kappa)$, such that $(i, j) \in E^\kappa$ iff $(\mathcal{J}_i, \mathcal{J}_j) \in \tilde{\mathrm{c}}(\mathcal{T})$ (again, we write $(\mathcal{J}_1, \dots, \mathcal{J}_p) \models \kappa$ in this situation).

In the sequel, let be κ a ctype with components $\kappa_1, \dots, \kappa_q$. Like before, we group tuples $(\mathcal{J}_1, \dots, \mathcal{J}_p)$ according to the κ-components the coordinates belong to (cf. the listing case). Since the κ partition the set of admissible indices and for $j > (kl)^p$ we have a factor equal to zero, we get:

$$|\varphi(\mathfrak{A})| = \sum_\kappa \sum_{(\bar{\mathcal{J}}_1, \dots, \bar{\mathcal{J}}_q) \models \kappa} \sum_{j=1}^{(kl)^p} (-1)^{j+1} \cdot \mu_j(|\bar{\mathcal{J}}_1|, \dots, |\bar{\mathcal{J}}_q|) \cdot |A_{\bar{\mathcal{J}}_1, \dots, \bar{\mathcal{J}}_q}| \tag{11}$$

$$= \sum_j^{(kl)^p} (-1)^{j+1} \sum_{\bar{m} \in [kl]^p} \mu_j(\bar{m}) \cdot \sum_\kappa \sum_{(\bar{\mathcal{J}}_1, \dots, \bar{\mathcal{J}}_q) \models \kappa, |\mathcal{J}_\nu| = m_\nu} |A^1_{\bar{\mathcal{J}}_1}| \cdots |A^q_{\bar{\mathcal{J}}_q}| \tag{12}$$

To get the second equality let $A^i_{\bar{\mathcal{J}}_i}$ be the restriction of the definition of $A_{\mathcal{J}_1, \dots, \mathcal{J}_p}$ to single indices. In particular, let

$$\begin{aligned} A^i_{\bar{\mathcal{J}}_i} := \{\bar{a} \restriction \epsilon(\kappa_i) \mid \text{for all } j, \mu \text{ s.t. } j \text{ is the } \mu\text{'th element of } \kappa_i \\ \Rightarrow \bar{a} \restriction \epsilon_j \cap K^r(U_J) \neq \emptyset \text{ for all } J \in \mathcal{J}_{i,\mu} \text{ and} \\ \mathfrak{A} \models \rho_{t, \epsilon(\kappa_i)}(\bar{x} \restriction \epsilon(\kappa_i)) \wedge \bigwedge_{j \in \kappa_i} \varphi_j(\bar{a} \restriction \epsilon_j)\}, \end{aligned} \tag{13}$$

fix the arguments of μ_j (to $\bar{m}$) and pull the sum over the tuples $(\bar{\mathcal{J}}_1, \dots, \bar{\mathcal{J}}_q)$ inwards. Implicitly we used an analogue of lemma 7 adopted to this extended setting to write $|A_{\mathcal{J}_1, \dots, \mathcal{J}_p}|$ as the product $|A^1_{\bar{\mathcal{J}}_1}| \cdots |A^q_{\bar{\mathcal{J}}_q}|$.

To define an extended contiguousness-graph, let $\mathcal{T} = \{U_1, \dots, U_m\}$ be a nice (r, l, g)-tree cover. For κ and $\bar{m} \in [k \cdot l]^p$ the *model powerset* is the set $\mathrm{PMod}(\kappa, \bar{m}) := \{\bar{\mathcal{J}} \mid \bar{\mathcal{J}} \models \kappa \text{ and } |\mathcal{J}_j| = m_j\}$. This is the obvious extension of Mod to the present case. Let furthermore κ be a p-ctype with components $\kappa_1, \dots, \kappa_q$ and $(\bar{m}_1, \dots \bar{m}_q) \in [k \cdot l]^p$. The *cgraph* $\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})$ of $\mathcal{T}$ with respect to κ and $\bar{m}$ has vertex set $\mathrm{PMod}(\kappa_1, \bar{m}_1) \dot\cup \cdots \dot\cup \mathrm{PMod}(\kappa_q, \bar{m}_q)$ and edge relation $E^{\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})}$ defined as follows:

$$(\bar{\mathcal{J}}_i, \bar{\mathcal{J}}_j) \in E^{\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})} \quad \text{iff} \quad \text{there are } \nu, \mu : (\mathcal{J}_{i,\nu}, \mathcal{J}_{j,\mu}) \in \tilde{\mathrm{c}}(\mathcal{T})$$

Additionally, we give the vertices corresponding to $\mathrm{PMod}(\kappa_i)$ the color i, or more formally, we add unary relations $C_1, \dots, C_q$ with $C_i^{\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})} := \mathrm{PMod}(\kappa_i)$.

The next lemma corresponds to lemma 8 and gathers the relevant properties of cgraphs $\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})$ for a tree cover $\mathcal{T}$.

Lemma 15. *Let $\mathcal{T}$ be a (r, l, g)-tree cover of a structure $\mathfrak{A}$. Then for all κ and $\bar{m} \in [k \cdot l]^p$:*

(1) $\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})$ can be calculated in time $O(\|\mathcal{T}\|)$.
(2) $\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})$ has bounded valence

Proof: To prove (1) we present an algorithm computing $\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})$. In the first step we compute the graph $\tilde{\mathrm{c}}(\mathcal{T})$ without the unnecessary vertices $\mathcal{J}$ (those for which $U_{\mathcal{J}} = \emptyset$). This procedure is displayed as algorithm 5. For the correctness, note that if for $\mathcal{J} \subseteq [m]$ we have $U_{\mathcal{J}} \neq \emptyset$, then there is a $J \in \mathcal{J}$ with $\mathcal{J} \subseteq N^{\mathrm{c}(\mathcal{T})}(J)$. Hence, we really get all relevant vertices of $\tilde{\mathrm{c}}(\mathcal{T})$. For the loop which generates the adjacency lists, note further that $\mathcal{I} \times \mathcal{J} \subseteq E^{\mathrm{c}(\mathcal{T})}$ if and only if $(\mathcal{I}, \mathcal{J}) \in E^{\tilde{\mathrm{c}}(\mathcal{T})}$. Furthermore, for any two adjacent $\mathcal{I}, \mathcal{J}$ there must be $I \in \mathcal{I}$ and $J \in \mathcal{J}$ such that I and J are adjacent in $\mathrm{c}(\mathcal{T})$.

It is easy to see that the running time is bounded by $m \cdot 2^l \cdot l \cdot 2^l \cdot c$ where c is the time necessary to check $\mathcal{I} \times \mathcal{J} \subseteq E^{\mathrm{c}(\mathcal{T})}$ in a graph of bounded valence, given in adjacency list representation (this constant is independent from m).

```
 1 proc calc_pgraph(T)
 2   for J = 1 to m do
 3       i := 1;
 4       for I ⊆ N^c(T)(J) do
 5           vertex[i] := I; adj[i] := NULL;
 6           /* add adjacencies */
 7           for I ∈ N^c(T)(J) do
 8               for J ⊆ N^c(T)(I) do
 9                   if I × J ⊆ E^c(T)
10                     then
11                         append(adj[i], J);
12                   fi
13               od
14           od
15           i := i + 1;
16       od
17   od
18 .
```

Algorithm 5. Calculate $\tilde{\mathrm{c}}(\mathcal{T})$

To continue, recall algorithm 2, which calculates the cgraph $\mathrm{c}(\mathcal{T}, \kappa)$, and the definition of $\kappa_i(J)$ for $J \in [m]$ and $1 \leq i \leq q$. This definition is naturally extended to

$$\kappa_i(\mathcal{J}, \bar{m}) := \{\bar{\mathcal{J}} \in \mathrm{PMod}^{\mathcal{T}}(\kappa_i, \bar{m}) \mid \mathcal{J} = \mathcal{J}_j \text{ for some } j\}.$$

Algorithm 6 essentially works like algorithm 2, with roles of $\tilde{\mathrm{c}}(\mathcal{T})$ and $\mathrm{c}(\mathcal{T})$ interchanged.

Correctness is immediate by the definition of the $\tilde{\mathrm{c}}(\mathcal{T}, \kappa, \bar{m})$. For example, there is an edge between $\bar{\mathcal{I}}$ and $\bar{\mathcal{J}}$ iff there are μ,ν such that $(\mathcal{I}_\nu, \mathcal{J}_\mu) \in E^{\tilde{\mathrm{c}}(\mathcal{T})}$. The μ is delivered by the corresponding for-loop, and ν is implicit in the expression $\bar{\mathcal{I}} \in \kappa_l(\mathcal{I}, \bar{m}_l)$ (the ν such that $\mathcal{I} = \mathcal{I}_\nu$). Finally $\mathcal{I} \in N^{\tilde{\mathrm{c}}(\mathcal{T})}(\mathcal{J}_\mu)$ assures that the tuples are adjacent.

For the running time, recall that $\bar{\mathcal{J}} \in \kappa_i(\mathcal{J}, \bar{m}_i)$ can be checked in constant time (cf. the case of primitive vertices). Thus, all loops except from the one cycling over $\mathcal{J} \in \tilde{\mathrm{c}}(\mathcal{T})$ are bounded by a constant of the form $f(q, l)$ for a function $f : \mathbb{N}^2 \to \mathbb{N}$. Together, we get the running time dominated by $f(q, l) \cdot 2^l \cdot m$.

To prove (2), note that $\tilde{\mathrm{c}}(\mathcal{T})$ has maximum valence 2^l. Thus, $|\kappa_i(\mathcal{J}, \bar{m}_i)| \leq ((2^l)^{p_i})^{p_i - 1}$ for a vertex $\mathcal{J} \in \tilde{\mathrm{c}}(\mathcal{T})$. Thus, summing up over all loops that compute the adjacency list, we get $p_i \cdot q \cdot 2^l \cdot ((2^l)^p)^{p-1}$ as an upper bound on the valence (the maximal number of different $\bar{\mathcal{I}}$ the algorithm can choose in the loops between line 10 and line 18). □

Finally, by the following lemma we can assume the values $|A^i_{\bar{\mathcal{J}}_i}|$ being known. Observe that only here we really need that our tree covers are nice.

Lemma 16. *The values $|A^i_{\bar{\mathcal{J}}_i}|$ for $i = 1, \ldots, q$ and $\bar{\mathcal{J}}_i \in C_i^{c(\mathcal{T}, \kappa, \bar{m})}$ can be computed in linear time.*

```
1  proc calc_cgraph(𝒯, κ)
2    comment: calculate the vertex set
3    for i = 1 to q do  /* the colors */
4      j := 1;
5      for 𝒥 ∈ c̃(𝒯) do
6        for 𝒥̄ ∈ κ_i(𝒥, m̄_i) do
7          vertex[i, j] := 𝒥̄;
8          comment: calculate the adjacency list
9          adj[i, j] := NULL;
10         for μ = 1 to p_i do
11           for l = 1 to q do
12             for ℐ ∈ N^{c̃(𝒯)}(𝒥_μ) do
13               for ℐ̄ ∈ κ_l(ℐ, m̄_l) do
14                 append(adj[i, j], ℐ̄);
15               od
16             od
17           od
18         od
19         j := j + 1;
20       od
21     od
22   od
23   normalize the graph (vertex[·, ·], adj[·, ·]);
24 .
```

Algorithm 6. Calculate $\tilde{c}(\mathcal{T}, \kappa, \bar{m})$

Proof: We compute the $|A^{\kappa_i}_{\bar{\mathcal{J}}_i}|$ for all $\bar{\mathcal{J}}_i \in C_i$ and fixed $1 \leq i \leq q$ straightforwardly by a simple loop over $\bar{\mathcal{J}}_i$ and calls of the procedure from theorem 1. For that, observe that $|A^{\kappa_i}_{\bar{\mathcal{J}}_i}|$ can be computed in the structure $\langle U_{\bar{J}} \rangle^{\mathfrak{A}}$ for an index $\bar{J}$ with $J_\nu \in \mathcal{J}_{i,\nu}$, for all ν. Furthermore, for an index $\bar{J}$ there are at most $(2^l)^{|\kappa_i|}$ many $\bar{\mathcal{J}}_i$ satisfying this condition. Like before, since each index $\bar{J}$ is used constantly often, also each $a \in A$ is contained constantly often in a set $\langle U_{\bar{J}} \rangle$, hence the entire computation needs linear time. □

Coming back to the proof of lemma 12, fix a $\bar{m} \in [k \cdot l]^q$ and a ctype κ with components $\kappa_1, \ldots, \kappa_q$. Define $\mathcal{G} := \tilde{c}(\mathcal{T}, \kappa, \bar{m})$ and set $\gamma(\bar{\mathcal{J}}_i) := |A^i_{\bar{\mathcal{J}}_i}|$ for $\bar{\mathcal{J}}_i \in C_i := \mathrm{PMod}(\kappa_i, \bar{m}_i)$. Then the tuples $(\bar{\mathcal{J}}_1, \ldots, \bar{\mathcal{J}}_q) \in C_1 \times \cdots \times C_q$ admitted in the sum are exactly the ones that are independent in $\mathcal{G}$. This completes the proof of lemma 12. □

Proof of theorem 13:
Let $\mathcal{G}$ and a labelling γ be given in the theorem. We group the tuples $\bar{v}$ according to the subgraph they induce in $\mathcal{G}$. Fix a graph π over the set $[q]$ and define

$$\mathrm{Hom}(\pi) := \{\bar{v} \mid (i,j) \in E^\pi \Rightarrow (v_i, v_j) \in E^{\mathcal{G}}\}$$

and

$$\mathrm{Iso}(\pi) := \{\bar{v} \mid (i,j) \in E^\pi \Leftrightarrow (v_i, v_j) \in E^{\mathcal{G}}\}.$$

It is easy to see that $\bar{v} \in \mathrm{Iso}(\pi)$ iff $\bar{v} \in \mathrm{Hom}(\pi)$ and for all $\pi' \supsetneq \pi$: $\bar{v} \notin \mathrm{Hom}(\pi')$. Using these index sets, we define $c_\pi := \sum_{\bar{v} \in \mathrm{Hom}(\pi)} \prod_{i \in \pi} \gamma(v_i)$ and $d_\pi := \sum_{\bar{v} \in \mathrm{Iso}(\pi)} \prod_{i \in \pi} \gamma(v_i)$. It is easy to see that $\sigma(\mathcal{G}, \gamma)$ coincides with d_π for π being the graph consisting of q isolated vertices.

We start with a lemma that allows us to calculate the values c_π.

Lemma 17. *Let π be a arbitrary graph over $[q]$. Then c_π can be calculated in time $O(qd^q \cdot \|\mathcal{G}\|)$, where d is the maximal valence of $\mathcal{G}$.*

Proof: Let π be a graph over $[q]$ splitting up into components $\pi_1, \ldots, \pi_r$. By definition, we have $c_\pi := c_{\pi_1} \cdots c_{\pi_r}$ (just apply the distributive law). Hence it is enough to show that the claim holds for connected π.

So assume that π is connected and take an arbitrary $j \in [q]$. If $\bar{v} \in \mathrm{Hom}(\pi)$ then, by connectedness, $v_i \in N_q^{\mathcal{G}}(v) \cap C_i$ for all $i \neq j$. Since $\mathcal{G}$ has valence at most d, $N_q^{\mathcal{G}}(v)$ contains $\leq d^q$ elements. Observe that the situation is similar to that in lemma 8, where we calculated $\mathrm{c}(\mathcal{T}, \kappa)$ for a tree cover $\mathcal{T}$ and a ctype κ.

Now the algorithm proceeds as follows: In a first phase, using the just derived characterization, we compute a list containing all tuples $\bar{v}$ that are in $\mathrm{Hom}(\pi)$. This list may contain multiples. Then we remove multiple occurrences from the list, and, in a last step, we sum up the $\gamma(\bar{v}) := \gamma(v_1) \cdots \gamma(v_q)$ for $\bar{v}$ from the list. This is displayed as algorithm 7.

```
1  proc calc_pi(G, γ)
2    comment: calculate the vertex set
3    h := ∅; j ∈ [q];
4    for v ∈ C_j do
5        for v̄ such that v_j = v and all v_i ∈ N_q(v) ∩ C_i do
6            if v̄ ∈ Hom(π) then
7                                    append(h, v̄);
8            fi
9        od
10   od
11   remove multiples from h;
12   return(∑_{v̄∈h} γ(v_1) ··· γ(v_q))
13 .
```

Algorithm 7. Calculate c_π

It is easy to see that the algorithm works correctly. The time bounds are easily verified. For instance, the loops require time $O(|G| \cdot d^q)$ (since $|N_q(v)| \leq d^q$). Multiple occurrences of tuples can be removed by a simple radix sort and the sum in the last line is computed in time linear in the size of h. □

With this lemma we are almost finished. By the above observation on the relation between homomorphisms and isomorphisms, we have

$$d_\pi = c_\pi - \sum_{\pi' \supsetneq \pi} d_{\pi'}, \tag{14}$$

which leads us to the following procedure:

Algorithm 8: Computing $\sigma(\mathcal{G}, \gamma)$

Input: A graph $(\mathcal{G}, C_1, \ldots, C_q)$ and $\gamma : G \to \mathbb{N}$
Output: $\sigma(\mathcal{G}, \gamma)$

1. Compute c_π for all π over $[q]$
2. Compute d_π for all π using formula (14)
3. Return d_π for π the empty graph

The correctness is immediate. Since there are $\leq q^2$ different π, the first line needs time $O(\|\mathcal{G}\|)$ (by lemma 17). Line 2 needs some more explanations: by induction, we proceed downwards starting with the complete graph π (for which we have $c_\pi = d_\pi$). Having computed all d_π for π with $l+1$ edges, we can compute d_π for π with l edges in constant time (simply apply formula (14)).

Altogether we stay within the time bounds claimed in theorem 13. □