Reducing CMSO Model Checking to Highly Connected Graphs

Daniel Lokshtanov[1]
University of Bergen, Norway
daniello@ii.uib.no

M. S. Ramanujan[2]
University of Warwick, United Kingdom
R.Maadapuzhi-Sridharan@warwick.ac.uk

Saket Saurabh[3]
The Institute of Mathematical Sciences, HBNI, Chennai, India
saket@imsc.res.in

Meirav Zehavi
Ben-Gurion University, Israel
Zehavimeirav@gmail.com

Abstract

Given a Counting Monadic Second Order (CMSO) sentence ψ, the CMSO[ψ] problem is defined as follows. The input to CMSO[ψ] is a graph G, and the objective is to determine whether $G \models \psi$. Our main theorem states that for every CMSO sentence ψ, if CMSO[ψ] is solvable in polynomial time on "globally highly connected graphs", then CMSO[ψ] is solvable in polynomial time (on general graphs). We demonstrate the utility of our theorem in the design of parameterized algorithms. Specifically we show that technical problem-specific ingredients of a powerful method for designing parameterized algorithms, recursive understanding, can be replaced by a black-box invocation of our main theorem. We also show that our theorem can be easily deployed to show fixed parameterized tractability of a wide range of problems, where the input is a graph G and the task is to find a connected induced subgraph of G such that "few" vertices in this subgraph have neighbors outside the subgraph, and additionally the subgraph has a CMSO-definable property.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Fixed Parameter Tractability Model Checking Recursive Understanding

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.135

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.01453`.

1 Introduction

Algorithmic meta-theorems are general algorithmic results applicable to a whole range of problems. Many prominent algorithmic meta-theorems are about model checking; such theorems state that for certain kinds of logic L, and all classes $\mathcal{C}$ that have a certain property,

[1] Supported by *Pareto-Optimal Parameterized Algorithms*, ERC Starting Grant 715744
[2] Supported by *BeHard*, Bergen Research Foundation and *X-Tract*, Austrian Science Fund (FWF, project P26696)
[3] Supported by *Parameterized Approximation*, ERC Starting Grant 306992.

there is an algorithm that takes as input a formula $\phi \in L$ and a structure $S \in \mathcal{C}$ and efficiently determines whether $S \models \phi$. Results in this direction include the seminal theorem of Courcelle [8, 7, 9] for model checking Monadic Second Order Logic (MSO) on graphs of bounded treewidth (see also [1, 2, 4, 10, 14]), as well as a large body of work on model checking first-order (FO) logic [5, 12, 16, 18, 19, 21, 20, 23, 28].

Another kind of algorithmic meta-theorems *reduce* the task of designing one type of algorithm for a problem, to one of designing a different kind of algorithm for the same problem. The hope is, of course, that the second type of algorithms are significantly easier to design than the first. A prototype example of such results is *Bidimensionality* [13], which reduces the design of sub-exponential time parameterized algorithms for a problem on planar (or H-minor free) graphs, to the design of single exponential time algorithms for the same problem when parameterized by the treewidth of the input graph.

In this paper we prove a result of the second type for model checking Counting Monadic Second Order Logic (CMSO), an extension of MSO with atomic sentences for determining the cardinality of vertex and edge sets modulo any (fixed) integer. For every CMSO sentence ψ define the CMSO[ψ] problem as follows. The input is a graph G on n vertices, and the task is to determine whether $G \models \psi$.

Our main result states that for every CMSO sentence ψ, if there is a $\mathcal{O}(n^d)$ time algorithm ($d > 4$) for CMSO[ψ] for the special case when the input graph is required to be "highly connected everywhere", then there is a $\mathcal{O}(n^d)$ time algorithm for CMSO[ψ] without any restrictions. In other words, our main theorem reduces CMSO model checking to model checking the same formula on graphs which are "highly connected everywhere".

In order to complete the description of our main result we need to define what we mean by "highly connected everywhere". For two integers s and c, we say that a graph G is (s, c)-*unbreakable* if there does not exist a partition of the vertex set into three sets X, C, and Y such that

- C is a separator: there are no edges from X to Y,
- C is small: $|C| \leq c$, and
- X and Y are large: $|X|, |Y| \geq s$.

For example, the set of $(1, c)$-unbreakable graphs contains precisely the $(c + 1)$-connected graphs, i.e. the connected graphs for which removing any set of at most c vertices leaves the graph connected. We can now state our main result:

▶ **Theorem 1.** *Let ψ be a CMSO sentence. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves* CMSO[ψ] *on (s, c)-unbreakable graphs in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves* CMSO[ψ] *on general graphs in time $\mathcal{O}(n^d)$.*

For Theorem 1 to be useful, there must exist problems that can be formulated in CMSO, for which it is easier to design algorithms for the special case when the input graphs are unbreakable, than it is to design algorithms that work on general graphs. Such problems can be found in abundance in *parameterized complexity.* Indeed, the *recursive understanding* technique, which has been used to solve several problems [6, 22, 24, 25, 27, 26] in parameterized complexity, is based precisely on the observation that for many graph problems it is much easier to design algorithms if the input graph can be assumed to be unbreakable.

Designing algorithms using the recursive understanding technique typically involves a technical and involved argument akin to doing dynamic programming on graphs of bounded treewidth (see Chitnis et al. [6] for an exposition). These arguments reduce the original problem on general graphs to a generalized version of the problem on (s, c)-unbreakable graphs, for appropriate values of s and c. Then an algorithm is designed for this generalized problem on (s, c)-unbreakable graphs, yielding an algorithm for the original problem.

For all applications of the recursive understanding technique known to the authors [6, 22, 24, 25, 27, 26], the problem in question (in which recursive understanding has been applied) can be formulated as a CMSO model checking problem, and therefore, the rather cumbersome application of recursive understanding can be completely replaced by a black box invocation of Theorem 1. Using Theorem 1 in place of recursive understanding has the additional advantage that it reduces problems on general graphs to *the same* problem on unbreakable graphs, facilitating also the last step of designing an algorithm on unbreakable graphs.

As an example of the power of Theorem 1 we use it to give a fixed parameter tractable (FPT) algorithm for the Vertex Multiway Cut Uncut problem. Details can be found in the full version of the paper on arXiv.org. Here, we are given a graph G together with a set of terminals $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on the set T, and an integer k, and the objective is to test whether there exists a set $S \subseteq V(G) \setminus T$ of at most k vertices such that for any $u, v \in T$, the vertices u and v belong to the same connected component of $G \setminus S$ if and only if $(u, v) \in \mathcal{R}$. Since finding the desired set S satisfying the above property can be formulated in CMSO, we are able to completely sidestep the necessity to define a technically involved annotated version of our problem, and furthermore, we need only focus on the base case where the graph is unbreakable. To solve the base case, a simple procedure that is based on the enumeration of connected sets with small neighborhood is sufficient. For classification purposes, our approach is significantly simpler than the problem-specific algorithm in [6]. Finally, we show how Theorem 1 can be effortlessly deployed to show fixed parameterized tractability of a wide range of problems, where the input is a graph G and the task is to find a connected induced subgraph of G of bounded treewidth such that "few" vertices outside this subgraph have neighbors inside the subgraph, and additionally the subgraph has a CMSO-definable property.

Our techniques. The proof of Theorem 1 is based heavily on the idea of graph replacement, which dates back to the work of Fellows and Langston [17]. We combine this idea with Courcelle's theorem [8, 7, 9], which states that every CMSO-definable property σ has finite state on a bounded-size separation/boundary. In other words, for any CMSO-definable property σ and fixed $t \in \mathbb{N}$, there is an equivalence relation defined on the set of all t-boundaried graphs (graphs with a set of at most t distinguished vertices) with a finite number, say ζ (where ζ depends only on σ and t) of equivalence classes such that if we replace any t-boundaried subgraph H of the given graph G with another t-boundaried graph, say H', from the same equivalence class to obtain a graph G', then G has the property σ if and only if G' has the property σ. In case of (s, c)-unbreakable graphs, $t = 2c$. Let $R_1, \ldots, R_\zeta$ denote a set containing one " minimal" $2c$-boundaried graph from each equivalence class (for the fixed CMSO-definable property σ). Let r denote the size of the largest among these minimal representatives.

The main technical content of our paper is in the description of an algorithm for a generalization of our question. To be precise, we will describe how one can, given a $2c$-boundaried graph G, locate the precise equivalence class in which G is contained and how one could compute the corresponding smallest representative from the set $\{R_1, \ldots, R_\zeta\}$. We refer to this task as "understanding" G.

In order to achieve our objective, we first give an algorithm $\mathcal{A}$ that allows one to understand $2c$-boundaried $(s - r, c)$-unbreakable graphs (for a choice of s which is sufficiently large compared to r and c). This algorithm is built upon the following observation. The equivalence class of any $2c$-boundaried graph G is determined exactly by the subset of $\{G \oplus R_1, G \oplus R_2, \ldots, G \oplus R_\zeta\}$ on which σ evaluates to true. Here, the graph $G \oplus R_i$ is the graph obtained by taking the disjoint union of the graphs G and R_i and then identifying the vertices of the boundaries of these graphs with the same label. Since s is chosen to be sufficiently large compared to c and r, it follows that for every $i \in \{1, \ldots, \zeta\}$, the graph $G \oplus R_i$ is (s, c)-unbreakable and we can use the assumed algorithm for CMSO[ψ] on (s, c)-unbreakable graphs to design an algorithm that *understands* $2c$-boundaried $(s - r, c)$-unbreakable graphs. This constitutes the 'base case' of our main algorithm.

In order to understand a general ($(s - r, c)$-breakable) $2c$-boundaried graph, we use known algorithms from [6] to compute a partition of the vertex set of G into X, C, and Y such that C is a separator, $|C| \leq c$ and $|X|, |Y| \geq \frac{s-r}{2^c}$. Let $G_1 = G[X \cup C]$ and let $G = G[Y \cup C]$. Without loss of generality, we may assume that at most half the vertices in the boundary of G lie in $X \cup C$. Consequently, the graph G_1 is a $2c$-boundaried graphs where the boundary vertices are the vertices in C along with the boundary vertices of G contained in $X \cup C$. We then recursively understand the strictly smaller $2c$-boundaried graph G_1 to find its representative $\hat{R} \in \{R_1, \ldots, R_\zeta\}$. Since the evaluation of σ on G is the same as the evaluation of σ on $G_2 \oplus \hat{R}$ (where the gluing happens along C), we only need to understand the $2c$-boundaried graph $G_2 \oplus \hat{R}$ (where the boundary is carefully defined from that of G and $\hat{R}$) and we do this by recursively executing the "understand" algorithm on this graph.

At this point we also need to remark on two drawbacks of Theorem 1. The first is that Theorem 1 is *non-constructive*. Given an algorithm for CMSO[ψ] on (s, c)-unbreakable graphs, Theorem 1 allows us to infer the existence of an algorithm for CMSO[ψ] on general graphs, but it does not provide us with the actual algorithm. This is due to the subroutine $\mathcal{S}$ requiring a representative $2c$-boundaried subgraph for each equivalence class, to be part of its 'source code'. Thus, the parameterized algorithms obtained using Theorem 1 are *non-uniform* (see Section 4), as opposed to the algorithms obtained by recursive understanding.

The second drawback is that Theorem 1 incurs a gargantuan constant factor overhead in the running time, where this factor depends on the formula ψ and the cut size c. We leave removing these two drawbacks as intriguing open problems.

2 Preliminaries

In order to present a rigorous proof of our lemmas in a way that is consistent with existing notation used in related work, we follow the notation from the paper [3].

Graphs and treewidth. Throughout this paper, we use the term "graph" to refer to a multigraph rather than only a simple graph. Given a graph G, we let $V(G)$ and $E(G)$ denote the vertex and edge sets of G, respectively. When G is clear from the context, we denote $n = |V(G)|$ and $m = |E(G)|$. Given two subsets of $V(G)$, A and B, we let $E(A, B)$ denote the set of edges of G with one endpoint in A and the other endpoint in B. Given $U \subseteq V(G)$, we let $G[U]$ denote the subgraph of G induced by U, and we let $N(U)$ and $N[U]$ denote the open and closed neighborhoods of U, respectively. Moreover, we denote $G \setminus U = G[V(G) \setminus U]$. Given $v \in V(G)$, we denote $N(v) = N(\{v\})$ and $N[v] = N[\{v\}]$. Given $E \subseteq E(G)$, we denote $G \setminus E = (V(G), E(G) \setminus E)$. Moreover, we let $V[E]$ denote the set of every vertex in $V(G)$ that is incident to at least one edge in E, and we define $G[E] = (V[E], E)$. A graph G is a *cluster graph* if there exists a partition $(V_1, V_2, \ldots, V_r)$ of $V(G)$ for some $r \in \mathbb{N}_0$ of $V(G)$ such that for all $i \in [r]$, $G[V_i]$ is a clique, and for all $j \in [r] \setminus \{i\}$, $E(V_i, V_j) = \emptyset$.

▶ **Definition 2.** A *tree decomposition* of a graph G is a pair (T,β) of a tree T and $\beta : V(T) \to 2^{V(G)}$, such that (a) $\bigcup_{t\in V(T)} \beta(t) = V(G)$, and (b) for any edge $e \in E(G)$, there exists a node $t \in V(T)$ such that both endpoints of e belong to $\beta(t)$, and (c) for any vertex $v \in V(G)$, the subgraph of T induced by the set $T_v = \{t \in V(T) : v \in \beta(t)\}$ is a tree.

The *width* of (T,β) is $\max_{v\in V(T)}\{|\beta(v)|\} - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G.

Unbreakability. To formally introduce the notion of unbreakability, we rely on the definition of a separation:

▶ **Definition 3** (Separation). A pair (X,Y) where $X \cup Y = V(G)$ is a *separation* if $E(X \setminus Y, Y \setminus X) = \emptyset$. The order of (X,Y) is $|X \cap Y|$.

Roughly speaking, a graph is breakable if it is possible to “break” it into two large parts by removing only a small number of vertices. Formally,

▶ **Definition 4** ((s,c)-Unbreakable graph). Let G be a graph. If there exists a separation (X,Y) of order at most c such that $|X \setminus Y| > s$ and $|Y \setminus X| > s$, called an (s,c)*-witnessing separation*, then G is (s,c)*-breakable*. Otherwise, G is (s,c)*-unbreakable*.

The following lemma implies that it is possible to determine (approximately) whether a graph is unbreakable or not, and lemmata similar to it can be found in [6].

▶ **Lemma 5.** *There exists an algorithm,* Break-ALG*, that given $s,c \in \mathbb{N}$ and a graph G, in time $2^{\mathcal{O}(c\log(s+c))} \cdot n^3 \log n$ either returns an $(\frac{s}{2^c},c)$-witnessing separation or correctly concludes that G is (s,c)-unbreakable.*

Boundaried Graphs. Roughly speaking, a boundaried graph is a graph where some vertices are labeled. Formally,

▶ **Definition 6** (Boundaried graph). A *boundaried graph* is a graph G with a set $\delta(G) \subseteq V(G)$ of distinguished vertices called *boundary vertices*, and an injective labeling $\lambda_G : \delta(G) \to \mathbb{N}$. The set $\delta(G)$ is the *boundary* of G, and the *label set* of G is $\Lambda(G) = \{\lambda_G(v) \mid v \in \delta(G)\}$.

We remark that we also extend the definition of (s,c)-(un)breakability from graphs, to boundaried graphs in the natural way. That is, we ignore the boundary vertices when considering the existence of an (s,c)-witnessing separation. For ease of presentation, we sometimes abuse notation and treat equally-labeled vertices of different boundaried graphs, as well as the vertex that is the result of the identification of two such vertices, as the same vertex. Given a finite set $I \subseteq \mathbb{N}$, $\mathcal{F}_I$ denotes the class of all boundaried graphs whose label set is I, and $\mathcal{F}_{\subseteq I} = \bigcup_{I'\subseteq I} \mathcal{F}_{I'}$. A boundaried graph in $\mathcal{F}_{\subseteq[t]}$ is called a t*-boundaried* graph. Finally, $\mathcal{F}$ denotes the class of all boundaried graphs. The main operation employed to unite two boundaried graphs is the one that glues their boundary vertices together. Formally,

▶ **Definition 7** (Gluing by $\oplus$). Let G_1 and G_2 be two boundaried graphs. Then, $G_1 \oplus G_2$ is the (not-boundaried) graph obtained from the disjoint union of G_1 and G_2 by identifying equally-labeled vertices in $\delta(G_1)$ and $\delta(G_2)$.[4]

[4] Each edge in G_1 (or G_2) whose endpoints are boundaried vertices in G_1 (or G_2) is preserved as a unique edge in $G_1 \oplus G_2$.

Structures. We first define the extension of graphs to *structures* in the context of our paper.

▶ **Definition 8** (Structure). A *structure* α is a tuple whose first element is a graph, denoted by G_α, and each of the remaining elements is a subset of $V(G_\alpha)$, a subset of $E(G_\alpha)$, a vertex in $V(G_\alpha)$ or an edge in $E(G_\alpha)$. The number of elements in the tuple is the *arity* of the structure.

Given a structure α of arity p and an integer $i \in [p]$, we let $\alpha[i]$ denote the i'th element of α. Note that $\alpha[1] = G_\alpha$. By *appending* a subset S of $V(G_\alpha)$ (or $E(G_\alpha)$) to a structure α of arity p, we produce a new structure, denoted by $\alpha' = \alpha \diamond S$, of arity $p+1$ with the first p elements of α' being the elements of α and $\alpha'[p+1] = S$. For example, consider the structure $\alpha = (G_\alpha, S, e)$ of arity 3, where $S \subseteq V(G_\alpha)$ and $e \in E(G_\alpha)$. Let S' be some subset of $V(G_\alpha)$. Then, appending S' to α results in the structure $\alpha' = \alpha \diamond S' = (G_\alpha, S, e, S')$.

Next, we define the notions of a *type* of a structure and a *property* of structures.

▶ **Definition 9** (Type). Let α be a structure of arity p. The *type* of α is a tuple of arity p, denoted by $\mathbf{type}(\alpha)$, where the first element, $\mathbf{type}(\alpha)[1]$, is graph, and for every $i \in \{2, 3, \dots, p\}$, $\mathbf{type}(\alpha)[i]$ is vertex if $\alpha[i] \in V(G_\alpha)$, edge if $\alpha[i] \in E(G_\alpha)$, vertex set if $\alpha[i] \subseteq V(G_\alpha)$, and edge set otherwise.[5]

▶ **Definition 10** (Property). A *property* is a function σ from the set of all structures to {true, false}.

Finally, we extend the notion of unbreakability to structures.

▶ **Definition 11** ((s,c)-Unbreakable structure). Let α be a structure. If G_α is an (s,c)-unbreakable graph, then we say that α is an (s,c)-*unbreakable* structure, and otherwise we say that α is an (s,c)-*breakable* structure.

Counting Monadic Second Order Logic. The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall$ and $\exists$, which can be applied to these variables, and five binary relations: (a) $u \in U$, where u is a vertex variable and U is a vertex set variable; (b) $d \in D$, where d is an edge variable and D is an edge set variable; (c) $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident to u; (d) $\mathbf{adj}(u, v)$, where u and v are vertex variables, and the interpretation is that u and v are adjacent; (e) equality of variables representing vertices, edges, vertex sets and edge sets.

Counting Monadic Second Order Logic (CMSO) extends MSO by including atomic sentences testing whether the cardinality of a set is equal to q modulo r, where q and r are integers such that $0 \le q < r$ and $r \ge 2$. That is, CMSO is MSO with the following atomic sentence: $\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \pmod r$, where S is a set. We refer to [2, 8, 9] for a detailed introduction to CMSO.

Evaluation. To evaluate a CMSO-formula ψ on a structure α, we instantiate the free variables of ψ by the elements of α. In order to determine which of the free variables of ψ are instantiated by which of the elements of α, we introduce the following conventions. First, each free variable x of a CMSO-formula ψ is associated with a rank, $r_x \in \mathbb{N} \setminus \{1\}$. Thus, a

[5] Note that we distinguish between a set containing a single vertex (or edge) and a single vertex (or edge).

CMSO-formula ψ can be viewed as a string accompanied by a tuple of integers, where the tuple consists of one integer r_x for each free variable x of ψ.

Given a structure α and a CMSO-formula ψ, we say that $\mathbf{type}(\alpha)$ *matches* ψ if **(i)** the arity of α is at least $\max r_x$, where the maximum is taken over each free variable x of ψ, and **(ii)** for each free variable x of ψ, $\mathbf{type}(\alpha)[r_x]$ is compatible with the type of x. For example, if x is a vertex set variable, then $\mathbf{type}(\alpha)[r_x] = \mathsf{vertex\ set}$. Finally, we say that α *matches* ψ if $\mathbf{type}(\alpha)$ matches ψ. Given a free variable x of a CMSO sentence ψ and a structure α that matches ψ, the element corresponding to x in α is $\alpha[r_x]$.

▶ **Definition 12.** [Property σ_ψ] Given a CMSO-formula ψ, the property σ_ψ is defined as follows. Given a structure α, if α does not match ψ, then $\sigma_\psi(\alpha)$ equals false, and otherwise $\sigma_\psi(\alpha)$ equals the result of the evaluation of ψ where each free variable x of ψ is instantiated by $\alpha[r_x]$.

Note that some elements of α may not correspond to any variable of ψ. However, ψ may still be evaluated on the structure α—in this case, the evaluation of ψ does not depend on all the elements of the structure. If the arity of α is 1, then we use $\sigma_\psi(G_\alpha)$ to denote $\sigma_\psi(\alpha)$.

▶ **Definition 13.** [CMSO-definable property] A property σ is *CMSO-definable* if there exists a CMSO-formula ψ such that $\sigma = \sigma_\psi$. In this case, we say that ψ *defines* σ.

Structures. The notion of a *boundaried structure* is an extension of the notion of a *boundaried graph* and is defined as follows.

▶ **Definition 14** (Boundaried structure)**.** A *boundaried structure* is a tuple whose first element is a boundaried graph G, denoted by G_α, and each of the remaining elements is a subset of $V(G)$, a subset of $E(G)$, a vertex in $V(G)$, an edge in $E(G)$, or the symbol $\star$. The number of elements in the tuple is the *arity* of the boundaried structure.

Given a boundaried structure α of arity p and an integer $i \in [p]$, we let $\alpha[i]$ denote the i'th element of α. We remark that we extend the definition of (s, c)-(un)breakability of structures, to boundaried structures.

▶ **Definition 15** (Type)**.** Let α be a boundaried structure of arity p. The *type* of α is a tuple of arity p, denoted by $\mathbf{type}(\alpha)$, where the first element, $\mathbf{type}(\alpha)[1]$, is $\mathsf{boundaried\ graph}$, and for every $i \in \{2, 3, \ldots, p\}$, $\mathbf{type}(\alpha)[i]$ is vertex if $\alpha[i] \in V(G_\alpha)$, edge if $\alpha[i] \in E(G_\alpha)$, $\mathsf{vertex\ set}$ if $\alpha[i] \subseteq V(G_\alpha)$, $\mathsf{edge\ set}$ if $\alpha[i] \subseteq E(G_\alpha)$ and $\star$ otherwise.

Now, given a boundaried structure and a CMSO-formula ψ, we say that $\mathbf{type}(\alpha)$ *matches* ψ if **(i)** the arity of α is at least $\max r_x$, where the maximum is taken over each free variable x of ψ, and **(ii)** for each free variable x of ψ, $\mathbf{type}(\alpha)[r_x]$ is compatible with the type of x. Moreover, we say that α matches ψ if $\mathbf{type}(\alpha)$ *matches* ψ.

Given $p \in \mathbb{N}$, $\mathcal{A}^p$ denotes the class of all boundaried structures of arity p, and given a finite set $I \subseteq \mathbb{N}$, $\mathcal{A}^p_I$ ($\mathcal{A}^p_{\subseteq I}$) denotes the class of all boundaried structures of arity p whose boundaried graph belongs to $\mathcal{F}_I$ (resp. $\mathcal{F}_{\subseteq I}$). A boundaried structure in $\mathcal{A}^p_{\subseteq [t]}$ is called a *t-boundaried structure*. Finally, we let $\mathcal{A}$ denote the class of all boundaried structures.

▶ **Definition 16** (Compatiblity)**.** Two boundaried structures α and β are *compatible* (notationally, $\alpha \sim_c \beta$) if the following conditions are satisfied.

- α and β have the same arity p.
- For every $i \in [p]$:
  - $\mathbf{type}(\alpha)[i] = \mathbf{type}(\beta)[i] \neq \star$, or

  - $\mathbf{type}(\alpha)[i] \in \{\text{vertex,edge}\}$ and $\mathbf{type}(\beta)[i] = \star$, or
  - $\mathbf{type}(\beta)[i] \in \{\text{vertex,edge}\}$ and $\mathbf{type}(\alpha)[i] = \star$.
- For every $i \in [p]$ such that both $\alpha[i]$ and $\beta[i]$ are vertices: $\alpha[i] \in \delta(G_\alpha)$, $\beta[i] \in \delta(G_\beta)$ and $\lambda_{G_\alpha}(\alpha[i]) = \lambda_{G_\beta}(\beta[i])$.
- For every $i \in [p]$ such that both $\alpha[i]$ and $\beta[i]$ are edges: $\alpha[i] \in E(G_\alpha[\delta(G_\alpha)])$, $\beta[i] \in E(G_\beta[\delta(G_\beta)])$ and $\{\lambda_{G_\alpha}(x_{\alpha[i]}), \lambda_{G_\alpha}(y_{\alpha[i]})\} = \{\lambda_{G_\beta}(x_{\beta[i]}), \lambda_{G_\beta}(y_{\beta[i]})\}$, where $\alpha[i] = (x_{\alpha[i]}, y_{\alpha[i]})$ and $\beta[i] = (x_{\beta[i]}, y_{\beta[i]})$. That is, x_j and y_j are the endpoints of the edge $j \in \{\alpha[i], \beta[i]\}$.

▶ **Definition 17** (Gluing by $\oplus$). Given two compatible boundaried structures α and β of arity p, the operation $\alpha \oplus \beta$ is defined as follows.
- $\alpha \oplus \beta$ is a structure γ of arity p.
- $G_\gamma = G_\alpha \oplus G_\beta$.
- For every $i \in [p]$:
  - if $\alpha[i]$ and $\beta[i]$ are sets, $\gamma[i] = \alpha[i] \cup \beta[i]$;
  - if $\alpha[i]$ and $\beta[i]$ are vertices/edges, $\gamma[i] = \alpha[i] = \beta[i]$;
  - if $\alpha[i] = \star$, $\gamma[i] = \beta[i]$;
  - if $\beta[i] = \star$, $\gamma[i] = \alpha[i]$.

State. This subsection states a variant of the classical Courcelle's Theorem [8, 7, 9] (see also [10]), which is a central component in the proof of our main result. To this end, we first define the *compatibility equivalence relation* $\equiv_c$ on boundaried structures as follows. We say that $\alpha \equiv_c \beta$ if $\Lambda(G_\alpha) = \Lambda(G_\beta)$ and for every boundaried structure γ, $\alpha \sim_c \gamma \iff \beta \sim_c \gamma$. Now, we define the *canonical equivalence relation* $\equiv_\sigma$ on boundaried structures.

▶ **Definition 18** (Canonical equivalence). Given a property σ of structures, the *canonical equivalence relation* $\equiv_\sigma$ on boundaried structures is defined as follows. For two boundaried structures α and β, we say that $\alpha \equiv_\sigma \beta$ if **(i)** $\alpha \equiv_c \beta$, and **(ii)** for all boundaried structures γ compatible with α (and thus also with β), we have $\sigma(\alpha \oplus \gamma) = \text{true} \Leftrightarrow \sigma(\beta \oplus \gamma) = \text{true}$.

It is easy to verify that $\equiv_\sigma$ is indeed an equivalence relation. Given a property σ of structures, $p \in \mathbb{N}$ and $I \subseteq \mathbb{N}$, we let $\mathcal{E}_{\equiv_\sigma}[\mathcal{A}^p_{\subseteq I}]$ denote the set of equivalence classes of $\equiv_\sigma$ when restricted to $\mathcal{A}^p_{\subseteq I}$.

▶ **Definition 19** (Finite state). A property σ of structures is *finite state* if, for every $p \in \mathbb{N}$ and $I \subseteq \mathbb{N}$, $\mathcal{E}_{\equiv_\sigma}[\mathcal{A}^p_{\subseteq I}]$ is finite.

Given a CMSO sentence ψ, the canonical equivalence relation associated with ψ is $\equiv_{\sigma_\psi}$, and for the sake of simplicity, we denote this relation by $\equiv_\psi$. We are now ready to state the variant of Courcelle's Theorem which was proven in [3] (see also [8, 7, 9]) and which we use in this paper.

▶ **Lemma 20** ([3]). *Every CMSO-definable property on structures has finite state.*

Parameterized Complexity. An instance of a parameterized problem is a pair of the form (x, k), where k is a non-negative integer called the *parameter*. Thus, a parameterized problem Π is a subset of $\Sigma^* \times \mathbb{N}_0$, for some finite alphabet Σ.

Two central notions in parameterized complexity are those of *uniform fixed-parameter tractability* and *non-uniform fixed-parameter tractability*. In this paper, we are interested in the second notion, which is defined as follows.

▶ **Definition 21** (Non-uniform fixed-parameter tractability (FPT))**.** Let Π be a parameterized problem. We say that Π is *non-uniformly fixed-parameter tractable (*FPT*)* if there exists a fixed d such that for every fixed $k \in \mathbb{N}_0$, there exists an algorithm A_k that for every $x \in \Sigma^*$, determines whether $(x,k) \in \Pi$ in time $\mathcal{O}(|x|^d)$.

Note that in Definition 21, d is independent of k. We refer the reader to the books [15, 11] for a detailed introduction to parameterized complexity.

3 CMSO Model Checking

Given a CMSO formula ψ, the CMSO$[\psi]$ problem is defined as follows. The input of CMSO$[\psi]$ is a structure α that matches ψ, and the objective is to output $\sigma_\psi(\alpha)$. In this section, we prove the following result, which then implies Theorem 1.

▶ **Theorem 22.** *Let ψ be a CMSO formula. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves* CMSO$[\psi]$ *on (s,c)-unbreakable structures in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves* CMSO$[\psi]$ *on general structures in time $\mathcal{O}(n^d)$.*

In the context of parameterized complexity, MIN-CMSO$[\psi]$ (MIN-EDGE-CMSO$[\psi]$) is defined as follows. The input of MIN-CMSO$[\psi]$ is a structure α, where for all $S \subseteq V(G_\alpha)$ (resp. $S \subseteq E(G_\alpha)$), $\alpha \diamond S$ matches ψ, and a parameter k. The objective is to determine whether there exists $S \subseteq V(G_\alpha)$ (resp. $S \subseteq E(G_\alpha)$) of size at most k such that $\sigma_\psi(\alpha \diamond S)$ is true. Similarly, we define MAX-CMSO$[\psi]$ (resp. MAX-EDGE-CMSO$[\psi]$), where the size of S should be at least k, and EQ-CMSO$[\psi]$ (resp. EQ-EDGE-CMSO$[\psi]$), where the size of S should be exactly k. Then, as a consequence of Theorem 22, we derive the following result.

▶ **Theorem 23.** *Let* X$\in$ {MIN,MAX,EQ,MIN-EDGE,MAX-EDGE,EQ-EDGE}*, and let $\widehat{\psi}$ be a CMSO sentence. For all $\widehat{c} : \mathbb{N}_0 \to \mathbb{N}_0$, there exists $\widehat{s} : \mathbb{N}_0 \to \mathbb{N}_0$ such that if* X-CMSO$[\widehat{\psi}]$ *parameterized by k is* FPT *on $(\widehat{s}(k), \widehat{c}(k))$-unbreakable structures, then* X-CMSO$[\widehat{\psi}]$ *parameterized by k is* FPT *on general structures.*

From now on, to prove Theorem 22, we assume a fixed CMSO formula ψ and a fixed $c \in \mathbb{N}$. Moreover, we fix p as the number of free variables of ψ, and $I = [2c]$. We also let $s \in \mathbb{N}$ be fixed, where its exact value (that depends only on ψ and c) is determined later. Finally, we assume that there exists an algorithm, Solve-Unbr-ALG, that solves CMSO$[\psi]$ on (s,c)-unbreakable structures in time $\mathcal{O}(n^d)$ for some $d > 4$.

3.1 Understanding an instance of the CMSO$[\psi]$ Problem

To solve CMSO$[\psi]$, we consider a generalization of CMSO$[\psi]$, called UNDERSTAND$[\psi]$. The definition of this generalization is based on an examination of $\mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$. Given a boundaried structure $\alpha \in \mathcal{A}^p_{\subseteq I}$, we let E_α denote the equivalence class in $\mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ that contains α. For every equivalence class $E_q \in \mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$, let α_{E_q} denote some boundaried structure in E_q such that there is no boundaried structure $\alpha \in E_q$ where the length of the string encoding α is smaller than the length of the string encoding α_{E_q}. Accordingly, denote $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}] = \{\alpha_{E_q} : E_q \in \mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]\}$. These will be the representatives of the equivalence classes induced by $\equiv_\psi$. By Lemma 20, there is a fixed $r \in \mathbb{N}$ (that depends only on ψ and c) such that both $|\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]|$ and the length of encoding of any boundaried structure in $\mathcal{R}_{\equiv_\psi}$ are upper bounded by r as well as $c \le r$. Note that the encoding explicitly lists all vertices and edges. By initially choosing s appropriately, we ensure that $s \ge 2r2^c + r$.

The UNDERSTAND[ψ] problem is defined as follows. The input is a boundaried structure $\alpha \in \mathcal{A}^p_{\subseteq I}$ that matches ψ, and the objective is to output a boundaried structure $\beta \in \mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ such that $E_\alpha = E_\beta$. We proceed by showing that to prove Theorem 22, it is sufficient to prove that there exists an algorithm that solves UNDERSTAND[ψ] on general boundaried structures in time $\mathcal{O}(n^d)$.

▶ **Lemma 24.** *If there exists an algorithm that solves* UNDERSTAND[ψ] *on general boundaried structures in time $\mathcal{O}(n^d)$, then there exists an algorithm that solves* CMSO[ψ] *on general structures in time $\mathcal{O}(n^d)$.*

In light of Lemma 24, the rest of this section focuses on the proof of the following result.

▶ **Lemma 25.** *There exists an algorithm that solves* UNDERSTAND[ψ] *on general boundaried structures in time $\mathcal{O}(n^d)$.*

3.2 Understand[ψ] on Unbreakable Structures

Recall that $s \geq 2r2^c + r$. In this subsection, we show that Algorithm Solve-Unbr-ALG can be used as a subroutine in order to efficiently solve UNDERSTAND[ψ] on $(s-r,c)$-unbreakable boundaried structures. For this, we follow the method of test sets (see for example, [Section 12.5, [15]]). The high level idea here is as follows. We first enumerate the relevant subset of the finite set of minimal representatives. In other words, we simply list those minimal representatives which can be glued in a meaningful way to the structure under consideration, call it α. We now observe that gluing each of these representatives to α results in an (s,c)-unbreakable structure, which is what we need to call Solve-Unbr-ALG. In this way we solve the instance obtained by gluing α to each minimal representative.

Now, for every (not necessarily distinct) pair of minimal representatives, we glue them together and do the same. This way, we can identify the specific minimal representative whose behaviour when glued with *every* minimal representative, precisely resembles that of the structure α when we do the same with α. Consequently, we obtain a solution for UNDERSTAND[ψ]. We now formalize this intuition in the following lemma.

▶ **Lemma 26.** *There exists an algorithm* Understand-Unbr-ALG*, that solves* UNDERSTAND[ψ]*, where it is guaranteed that inputs are $(s-r,c)$-unbreakable boundaried structures, in time $\mathcal{O}(n^d)$.*[6]

Proof. We design the algorithm Understand-Unbr-ALG as follows. Let α be an input, which is an $(s-r,c)$-unbreakable boundaried structure. Moreover, let $\mathcal{C} = \{\gamma \in \mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}] : \gamma \equiv_c \alpha\}$, and let $\mathcal{T}$ denote the set of boundaried structures in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ that are compatible with α. In the first phase, the algorithm performs the following computation. Notice that for every $\beta \in \mathcal{T}$, since $|V(G_\beta)| \leq r$, it holds that $\alpha \oplus \beta$ is an (s,c)-unbreakable structure. Thus, for every $\beta \in \mathcal{T}$, Understand-Unbr-ALG can call Solve-Unbr-ALG with $\alpha \oplus \beta$ as input, and it lets $\mathsf{ans}(\alpha,\beta)$ denote the result.

In the second phase, the algorithm performs the following computation. Notice that for every $\gamma \in \mathcal{C}$ and $\beta \in \mathcal{T}$, since $|V(G_\beta)|, |V(G_\gamma)| \leq r$, it holds that $\gamma \oplus \beta$ is a $(2r,c)$-unbreakable structure. Thus, since $s \geq 2r2^c + r$, for all $\beta \in \mathcal{C}$ and $\gamma \in \mathcal{T}$, Understand-Unbr-ALG can call Solve-Unbr-ALG with $\gamma \oplus \beta$ as input, and it lets $\mathsf{ans}(\gamma,\beta)$ denote the result.

[6] Here, Understand-Unbr-ALG is not required to verify whether the input is an $(s-r,c)$-unbreakable boundaried structure.

Finally, in the third phase, for every $\beta \in \mathcal{C}$, the algorithm performs the following computation. It checks whether for every $\gamma \in \mathcal{T}$ it holds that $\mathsf{ans}(\alpha,\gamma) = \mathsf{ans}(\beta,\gamma)$, and if the answer is positive, then it outputs β. Since $\alpha \in \mathcal{A}^p_{\subseteq I}$, there exists $\beta' \in \mathcal{C}$ such that $E_\alpha = E_{\beta'}$, and therefore, at the latest, when $\beta = \beta'$, the algorithm terminates. Thus, the algorithm is well defined, and it is clear that it runs in time $\mathcal{O}(n^d)$.

To conclude that the algorithm is correct, it remains to show that for all $\beta \in \mathcal{C} \setminus \{\beta'\}$, there exists $\gamma \in \mathcal{T}$ such that $\mathsf{ans}(\alpha,\gamma) \neq \mathsf{ans}(\beta,\gamma)$, as this would imply that the algorithm necessarily outputs β'. For this purpose, suppose by way of contradiction that there exists $\beta \in \mathcal{C} \setminus \{\beta'\}$ such that for all $\gamma \in \mathcal{T}$ it holds that $\mathsf{ans}(\alpha,\gamma) = \mathsf{ans}(\beta,\gamma)$. We now argue that $E_\beta = E_{\beta'}$ which leads to a contradiction since each boundaried structure in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ belongs to a different equivalence class.

For all $\gamma \in \mathcal{T}$, since it holds that $\mathsf{ans}(\alpha,\gamma) = \mathsf{ans}(\beta,\gamma)$, it also holds that $\mathsf{ans}(\beta',\gamma) = \mathsf{ans}(\beta,\gamma)$. This implies that $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$. Consider some boundaried structure γ (not necessarily in $\mathcal{T}$) that is compatible with β' (and thus also with β). We claim that $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$. Indeed, let γ' be the (unique) boundaried structure in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ such that $E_{\gamma'} = E_\gamma$. Then, $\sigma_\psi(\beta' \oplus \gamma') = \sigma_\psi(\beta' \oplus \gamma)$ and $\sigma_\psi(\beta \oplus \gamma') = \sigma_\psi(\beta \oplus \gamma)$. Note that since γ' is compatible with β', it is also compatible with α, and hence $\gamma' \in \mathcal{T}$. Therefore, $\sigma_\psi(\beta' \oplus \gamma') = \sigma_\psi(\beta \oplus \gamma')$. Overall, we obtain that indeed $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$.

Note that $\beta \equiv_c \beta'$, and thus, since we have shown that for every boundaried structure γ compatible with β' it holds that $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$, we derive that $E_\beta = E_{\beta'}$. However, each boundaried structure in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ belongs to a different equivalence class, and thus we have reached the desired contradiction. ◀

3.3 Understand$[\psi]$ on General Structures

The Algorithm Understand-ALG. We start by describing an algorithm called Understand-ALG, which is based on recursion. Given an input to UNDERSTAND$[\psi]$ on general boundaried structures, which is a boundaried structure α, the algorithm works as follows. First, it calls Break-ALG (given by Lemma 5) with G_α as input to either obtain an $(\frac{s-r}{2^c}, c)$-witnessing separation (X,Y) or correctly conclude that G_α is $(s-r,c)$-unbreakable. In the second case or if $n < 2(s-r)$, it calls Understand-Unbr-ALG (given by Lemma 26), and returns its output. Next, suppose that Understand-ALG obtained an $(\frac{s-r}{2^c}, c)$-witnessing separation (X,Y) and that $n \geq 2(s-r)$. Without loss of generality, assume that $|X \cap \delta(G_\alpha)| \leq |Y \cap \delta(G_\alpha)|$. Denote $\Delta = \{v \in X \cap Y : v \notin \delta(G_\alpha)\}$.

Now, we define a boundaried structure, $\beta \in \mathcal{A}^p_{\subseteq I}$, which can serve as an instance of UNDERSTAND$[\psi]$. First, we let the graph G_β be $G_\alpha[X]$, and we define $\delta(G_\beta) = (X \cap \delta(G_\alpha)) \cup \Delta$. Now, for all $v \in X \cap \delta(G_\alpha)$, we define $\lambda_{G_\beta}(v) = \lambda_{G_\alpha}(v)$. Since $|X \cap \delta(G_\alpha)| \leq |Y \cap \delta(G_\alpha)|$, $\alpha \in \mathcal{A}^p_{\subseteq I}$ and $|X \cap Y| \leq c$, we have that $|(X \cap \delta(G_\alpha)) \cup \Delta| \leq 2c$. Thus, to each $v \in \Delta$, we can let $\lambda_{G_\beta}(v)$ assign some unique integer from $I \setminus \lambda_{G_\alpha}(X \cap \delta(G_\alpha))$. Hence, $G_\beta \in \mathcal{F}_{\subseteq I}$. Now, for all $i \in \{2,\ldots,p\}$, we set $\beta[i]$ as follows. If $\mathbf{type}(\alpha)[i] \in \{\mathsf{vertex},\mathsf{edge}\}$: If $\alpha[i] \in V(G_\beta) \cup E(G_\beta)$, then $\beta[i] = \alpha[i]$, and otherwise $\beta[i] = \star$. Else: $\beta[i] = \alpha[i] \cap (V(G_\beta) \cup E(G_\beta))$.

Understand-ALG proceeds by calling itself recursively with β as input, and it lets β' be the output of this call. Now, we define another boundaried structure, $\gamma \in \mathcal{A}^p_{\subseteq I}$, which can serve as an instance of UNDERSTAND$[\psi]$. First, we define the boundaried graph G_γ as follows. Let H be the disjoint union of $G_{\beta'}$ and $G[Y]$, where both $G_{\beta'}$ and $G[Y]$ are treated as not-boundaried graphs. For all $v \in X \cap Y$, identify (in H) the vertex v of $G[Y]$ with the vertex u of $G_{\beta'}$ that satisfies $\lambda_{G_{\beta'}}(u) = \lambda_{G_\beta}(v)$, and for the sake of simplicity, let v and u also denote the identity of the resulting (unified) vertex. The graph G_γ is the result of this process. Moreover, let Δ' denote the set of vertices in $G_{\beta'}$ whose labels belong to $G_\beta(\Delta)$. Next, set $\delta(G_\gamma) = (Y \cap \delta(G_\alpha)) \cup (\delta(G_{\beta'}) \setminus \Delta')$. Now, for all $v \in Y \cap \delta(G_\alpha)$, we define $\lambda_{G_\gamma}(v) = \lambda_{G_\alpha}(v)$, and for all $v \in \delta(G_{\beta'}) \setminus \Delta'$, we define $\lambda_{G_\gamma}(v) = \lambda_{G_{\beta'}}(v)$ (note that if a vertex belongs to both $Y \cap \delta(G_\alpha)$ and $\delta(G_{\beta'}) \setminus \Delta'$, we still assign it the same label). Hence, $G_\gamma \in \mathcal{F}_{\subseteq I}$. For the sake of simplicity, if two vertices have the same label (one in G_α and the other in G_γ), we let the identity of one of them also refer to the other and vice versa. For all $i \in \{2, \ldots, p\}$, we set $\gamma[i]$ to have the same type as $\alpha[i]$, and define it as follows. If $\mathbf{type}(\alpha)[i] \in \{\mathsf{vertex}, \mathsf{edge}\}$: If $\alpha[i] \in V(G_\gamma) \cup E(G_\gamma)$, then $\gamma[i] = \alpha[i]$, and otherwise $\gamma[i] = \star$. Else: $\gamma[i] = \alpha[i] \cap (V(G_\gamma) \cup E(G_\gamma))$. Finally, Understand-ALG calls itself recursively with γ as input, and it returns γ', the output of this call.

Correctness and running time. Finally, we prove the following two results. Along with Lemma 27, these complete the proof of Lemma 25.

▶ **Lemma 27.** *If* Understand-ALG *terminates, then it correctly solves* UNDERSTAND$[\psi]$ *on general boundaried structures.*

▶ **Lemma 28.** Understand-ALG *runs in time* $\mathcal{O}(n^d)$.

4 Applications

In this section, we first show how Theorem 23 can be easily deployed to show the fixed parameter tractability of a wide range of problems of the following kind. The input is a graph G and the task is to find a connected induced subgraph of G of bounded treewidth such that "few" vertices outside this subgraph have neighbors inside the subgraph, and additionally the subgraph has a CMSO-definable property. Then, we show that technical problem-specific ingredients of a powerful method for designing parameterized algorithms called recursive understanding, can be replaced by a black-box invocation of Theorem 23.

4.1 "Pendant" Subgraphs with CMSO-Definable Properties

Formally, given a CMSO sentence ψ and a non-negative integer t, the t-PENDANT$[\psi]$ problem is defined as follows. The input of t-PENDANT$[\psi]$ is a graph G and a parameter k, and the objective is to determine whether there exists $U \subseteq V(G)$ such that $G[U]$ is a connected graph of treewidth at most t, $|N(U)| \leq k$ and $\sigma_\psi(G[U])$ is true.

It is straightforward to define a CMSO formula φ with free variable S such that the t-PENDANT$[\psi]$ problem is equivalent to MIN-CMSO$[\varphi]$ as follows.

▶ **Observation 4.1.** *Let G be a graph, and let k be a parameter. Then, (G, k) is a* YES-*instance of t-*PENDANT$[\psi]$ *if and only if $((G), k)$ is a* YES-*instance of* MIN-CMSO$[\varphi]$.

Next, we solve t-PENDANT$[\psi]$ on unbreakable graphs with the appropriate parameters. Define $c : \mathbb{N}_0 \to \mathbb{N}_0$ as follows. For all $k \in \mathbb{N}_0$, let $\widehat{c}(k) = k + t$. Let $s : \mathbb{N}_0 \to \mathbb{N}_0$ be the function $\widehat{s}$ in Theorem 23 with $\widehat{\psi} = \varphi$ and $\widehat{c} = c$. We first prove the following lemma.

▶ **Lemma 29.** *Let (G, k) be a* YES-*instance of t-*PENDANT$[\psi]$ *parameterized by k on $(s(k), k+t)$-unbreakable graphs. Then, there exists $U \subseteq V(G)$ such that $G[U]$ is a connected graph of treewidth at most t, $|N(U)| \leq k$, $\sigma_\psi(G[U])$ is* true *and $|U| < 3(s(k) + t)$.*

Proof. Since (G, k) is a YES-instance, there exists $U \subseteq V(G)$ such that $G[U]$ is a connected graph of treewidth at most t, $|N(U)| \leq k$ and $\sigma_\psi(G[U])$ is true. Moreover, since the treewidth of $G[U]$ is at most t, it is easy to see that there exists a separation (X, Y) of order at most t of $G[U]$ such that $|X|, |Y| \geq |U|/3$ (see, e.g., [11]). Then, set $X' = X \cup N(U)$ and $Y' = (V(G)\setminus X)\cup(X\cap Y)$. Note that (X', Y') is a separation of order $|X\cap Y|+|N(U)| \leq k+t$. Moreover, $X \setminus Y \subseteq X' \setminus Y'$ and $Y \setminus X \subseteq Y' \setminus X'$. Thus, (X', Y') is a $(|U|/3 - t, k + t)$-witnessing separation. Since G is $(s(k), k+t)$-unbreakable graph, we have that $|U|/3-t < s(k)$. Therefore, $|U| < 3(s(k) + t)$, which concludes the correctness of the lemma. ◀

▶ **Lemma 30.** *t-Pendant$[\psi]$ parameterized by k is* FPT *on $(s(k), k + t)$-unbreakable graphs.*

Finally, by Theorem 23, Observation 4.1 and Lemma 30, we derive the following result.

▶ **Theorem 31.** *t-Pendant$[\psi]$ parameterized by k is* FPT *on general graphs.*

References

1 Karl Abrahamson and Michael Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph structure theory (Seattle, WA, 1991)*, volume 147 of *Contemp. Math.*, pages 539–563, Providence, RI, 1993. Amer. Math. Soc. doi:10.1090/conm/147/01199.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. URL: http://dl.acm.org/citation.cfm?id=2973749, doi:10.1145/2973749.

4 Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.

5 Simone Bova, Robert Ganian, and Stefan Szeider. Model checking existential logic on partially ordered sets. *ACM Trans. Comput. Log.*, 17(2):10:1–10:35, 2016. doi:10.1145/2814937.

6 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

7 B. Courcelle. The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Théor. Appl.*, 26(3):257–286, 1992.

8 Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.

9 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of graph grammars and computing by graph transformation, Vol. 1*, pages 313–400. World Sci. Publ, River Edge, NJ, 1997.

10 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach.* Cambridge University Press, 2012.

11 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

12 A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *LICS'07*, pages 270–279. IEEE Computer Society, 2007.

13 Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

14 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, Berlin, 1998.

15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

16 Zdenek Dvorak, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

17 Michael R. Fellows and Michael A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 520–525. IEEE, 1989.

18 J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

19 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

20 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, and Sebastian Ordyniak. Faster existential FO model checking on posets. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:8)2015.

21 Robert Ganian, Petr Hlinený, Daniel Král, Jan Obdrzálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:11)2015.

22 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011. doi:10.1145/1993636.1993700.

23 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

24 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169, 2011. doi:10.1109/FOCS.2011.53.

25 Eun Jung Kim, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Parameterized algorithms for min-max multiway cut and list digraph homomorphism. *J. Comput. Syst. Sci.*, 86:191–206, 2017. doi:10.1016/j.jcss.2017.01.003.

26 Ashutosh Rai and M. S. Ramanujan. Strong parameterized deletion: Bipartite graphs. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 21:1–21:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.21.

27 Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. A parameterized algorithm for mixed-cut. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 672–685, 2016. doi:10.1007/978-3-662-49529-2_50.

28 D. Seese. Linear time computable problems and first-order descriptions. *Math. Structures Comput. Sci.*, 6(6):505–526, 1996.