Abstract

Algorithmic meta theorems are algorithmic results that apply to whole families of combinatorial problems, instead of just specific problems. These families are usually defined in terms of logic and graph theory. An archetypal algorithmic meta theorem is Courcelle’s Theorem [9], which states that all graph properties definable in monadic second-order logic can be decided in linear time on graphs of bounded tree width.

This article is an introduction into the theory underlying such meta theorems and a survey of the most important results in this area.

1 Introduction

In 1990, Courcelle [9] proved a fundamental theorem stating that graph properties definable in monadic second-order logic can be decided in linear time on graphs of bounded tree width. This is the first in a series of algorithmic meta theorems. More recent examples of such meta theorems state that all first-order definable properties of planar graphs can be decided in linear time [41] and that all first-order definable optimisation problems on classes of graphs with excluded minors can be approximated in polynomial time to any given approximation ratio [19]. The term “meta theorem” refers to the fact that these results do not describe algorithms for specific problems, but for whole families of problems, whose definition typically has a logical and a structural (usually graph theoretical) component. For example, Courcelle’s Theorem is about monadic second-order logic on graphs of bounded tree width.

This article is an introductory survey on algorithmic meta theorems. Why should we care about such theorems? First of all, they often provide a quick way to prove that a problem is solvable efficiently. For example, to show that the 3-colourability problem can be solved in linear time on graphs of bounded tree width, we observe that 3-colourability is a property of graphs definable in monadic second-order logic and apply Courcelle’s theorem. Secondly, and more substantially, algorithmic meta theorems yield a better understanding of the scope of general algorithmic techniques and, in some sense, the limits of tractability. In particular, they clarify the interactions between logic and combinatorial structure, which is fundamental for computational complexity.

The general form of algorithmic meta theorems is:

All problems definable in a certain logic on a certain class of structures can be solved efficiently.

Problems may be of different types, for example, they may be optimisation or counting problems, but in this article we mainly consider decision problems. We briefly discuss other types of problems in Section 7.2. Efficient solvability may mean, for example, polynomial time solvability, linear or quadratic time solvability, or fixed-parameter tractability. We will discuss this in detail in Section 2.3. Let us now focus on the two main ingredients of the meta theorems, logic and structure.
The two logics that, so far, have been considered almost exclusively for meta theorems are first-order logic and monadic second-order logic. Techniques from logic underlying the theorems are Feferman-Vaught style composition lemmas, automata theoretic techniques, and locality results such as Hanf’s Theorem and Gaifman’s Theorem.

The structures in algorithmic meta theorems are usually defined by graph theoretic properties. Actually, to ease the presentation, the only structures we will consider in this survey are graphs. Many of the meta theorems are tightly linked with graph minor theory. This deep theory, mainly developed by Robertson and Seymour in a long series of papers, describes the structure of graphs with excluded minors. It culminates in the graph minor theorem [69], which states that every class of closed graphs under taking minors can be characterised by a finite set of excluded minors. The theory also has significant algorithmic consequences. Robertson and Seymour [67] proved that every class of graphs that is closed under taking minors can be recognised in cubic time. More recently, results from graph minor theory have been combined with algorithmic techniques that had originally been developed for planar graphs to obtain polynomial time approximation schemes and fixed parameter tractable algorithms for many standard optimisation problems on families of graphs with excluded minors. The methods developed in this context are also underlying the more advanced algorithmic meta theorems.

There are some obvious similarities between algorithmic meta theorems and results from descriptive complexity theory, in particular such results from descriptive complexity theory that also involve restricted classes of graphs. As an example, consider the theorem stating that fixed-point logic with counting captures polynomial time on graphs of bounded tree width [46], that is, a property of graphs of bounded tree width is definable in fixed-point logic with counting if and only if it is decidable in polynomial time. Compare this to Courcelle’s Theorem. Despite the similarity, there are two crucial differences: On the one hand, Courcelle’s Theorem is weaker as it makes no completeness claim, that is, it does not state that all properties of graphs of bounded tree width that are decidable in linear time are definable in monadic second-order logic. On the other hand, Courcelle’s Theorem is stronger in its algorithmic content. Whereas it is very easy to show that all properties of graphs (not only graphs of bounded tree width) definable in fixed-point logic with counting are decidable in polynomial time, the proof of Courcelle’s theorem relies on substantial algorithmic ideas like the translation of monadic second-order logic over trees into tree automata [74] and a linear time algorithm for computing tree decompositions [5]. In general, algorithmic meta theorems involve nontrivial algorithms, but do not state completeness, whereas in typical results from descriptive complexity, the algorithmic content is limited, and the nontrivial part is completeness. But there is no clear dividing line. Consider, for example, Papadimitriou and Yannakakis’s [60] well known result that all optimisation problems in the logically defined class MAXSNP have a constant factor approximation algorithm. This theorem does not state completeness, but technically it is much closer to Fagin’s Theorem [35], a central result of descriptive complexity theory, than to the algorithmic meta theorems considered here. In any case, both algorithmic meta theorems and descriptive complexity theory are branches of finite model theory, and there is no need to draw a line between them.

The reader I had in mind when I wrote this article has a solid background in logic, but maybe knows less about graph theory. Consequently, we will be spending much more time on graph theoretic concepts and results than on logical ones. In particular, there is a long section devoted to graph minor theory. It was my goal to cover the developments up to the most recent and strongest results, which are concerned with monadic second-order logic on graphs of bounded rank width and with first-order logic on graphs with excluded minors. The proofs of most theorems are at least sketched, so that hopefully the reader will not only get an impression of the results, but also of the techniques involved in their proofs.

2 The basics

\( \mathbb{R} \), \( \mathbb{Q} \), \( \mathbb{Z} \), and \( \mathbb{N} \) denote the sets of real numbers, rational numbers, integers, and natural numbers (that is, positive integers), respectively. For a set \( S \subseteq \mathbb{R} \), by \( S_{\geq 0} \) we denote the set of nonnegative numbers in \( S \). For integers \( m, n \), by \( [m,n] \) we denote the interval \( \{m,m+1,\ldots,n\} \), which is empty if \( n < m \). Furthermore, we let \([n] = [1,n]\). The power set of a set \( S \) is denoted by \( 2^S \), and the set of all \( k \)-element subsets of \( S \) by \( \binom{S}{k} \).
2.1 Graphs

A graph $G$ is a pair $(V(G),E(G))$, where $V(G)$ is a finite set whose elements are called vertices and $E(G) \subseteq \binom{V(G)}{2}$ is a set of unordered pairs of vertices, which are called edges. Hence graphs in this paper are always finite, undirected, and simple, where simple means that there are no self-loops or parallel edges.

If $e = \{u,v\}$ is an edge, we say that the vertices $u$ and $v$ are adjacent, and that both $u$ and $v$ are incident with $e$. A graph $H$ is a subgraph of a graph $G$ (we write $G \subseteq H$) if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

If $E(H) = E(G) \cap \binom{V(G)}{2}$, then $H$ is an induced subgraph of $G$. For a set $W \subseteq V(G)$, we write $G[W]$ to denote the induced subgraph $(W,E(G) \cap \binom{W}{2})$ and $G \setminus W$ to denote $G[V(G) \setminus W]$. For a set $F \subseteq E$, we let $G[F]$ be the subgraph $(\cup F, F)$. Here \( \cup F \) denote the union of all edges in $F$, which is the set of all vertices incident with at least one edge in $F$. We call $G[F]$ the subgraph of $G$ generated by $F$; note that it is not necessarily an induced subgraph of $G$. The union of two graphs $G$ and $H$ is the graph $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$, and the intersection $G \cap H$ is defined similarly. The complement of a graph $G = (V,E)$ is the graph $\overline{G} = (V, \binom{V}{2} \setminus E)$. There is a unique empty graph $(\emptyset, \emptyset)$. For $n \geq 1$, we let $K_n$ be the complete graph with $n$ vertices. To be precise, let us say $K_n = \{[n], [n]\}$. Similarly, we let $K_{n,m}$ be the complete bipartite graph with parts of size $m,n$, respectively.

Occasionally, we consider (vertex) labelled graphs. A labelled graph is a tuple $G = (V(G),E(G),P_1(G),\ldots,P_l(G))$, where $P_i(G) \subseteq V(G)$ for all $i \in [l]$. The symbols $P_i$ are called labels, and if $v \in P_i(G)$ we say that $v$ is labelled by $P_i$. Subgraphs, union, and intersection extend to labelled graphs in a straightforward manner. The underlying graph of a labelled graph $G$ is $(V(G),E(G))$. Whenever we apply graph theoretic notions such as connectivity to labelled graphs, we refer to the underlying graph.

The order $|G|$ of a graph $G$ is the number of vertices of $G$. We usually use the letter $n$ to denote the order of a graph. The size of $G$ is the number $|E(G)| = |E|$. Up to a constant factor, this is the size of the adjacency list representation of $G$ under a uniform cost model.

\( \mathcal{G} \) denotes the class of all graphs. For every class \( \mathcal{G} \) of graphs, we let \( \mathcal{G}_{lb} \) be the class of all labelled graphs whose underlying graph is in \( \mathcal{G} \). A graph invariant is a mapping defined on the class \( \mathcal{G} \) of all graphs that is invariant under isomorphisms. All graph invariants considered in this paper are integer valued. For a graph invariant $f : \mathcal{G} \to \mathbb{Z}$ and a class \( \mathcal{G} \) of graphs, we say that \( \mathcal{G} \) has bounded $f$ if there is a $k \in \mathbb{Z}$ such that $f(G) \leq k$ for all $G \in \mathcal{G}$.

Let $G = (V,E)$ be a graph. The degree $\deg^G(v)$ of a vertex $v \in V$ is the number of edges incident with $v$. We omit the superscript $G$ if $G$ is clear from the context. The (maximum) degree of $G$ is the number

$$\Delta(G) = \max \{\deg(v) : v \in V\}.$$  

The minimum degree $\delta(G)$ is defined analogously, and the average degree $d(G)$ is $2|E(G)|/|V(G)|$. Observe that $|G| = O(d(G) \cdot |G|)$. Hence if a class \( \mathcal{G} \) of graphs has bounded average degree, then the size of the graphs in \( \mathcal{G} \) is linearly bounded in the order. In the following, “degree” of a graph, without qualifications, always means “maximum degree”.

A path in $G = (V,E)$ of length $n \geq 0$ from a vertex $v_0$ to a vertex $v_n$ is a sequence $v_0, \ldots, v_n$ of distinct vertices such that $\{v_{i-1}, v_i\} \in E$ for all $i \in [n]$. Note that the length of a path is the number of edges on the path. Two paths are disjoint if they have no vertex in common. $G$ is connected if it is nonempty and for all $v, w \in V$ there is a path from $v$ to $w$. A connected component of $G$ is a maximal (with respect to $\subseteq$) connected subgraph. Graph $G$ is $k$-connected, for some $k \geq 1$, if $|V| > k$ and for every $W \subseteq V$ with $|W| < k$ the graph $G \setminus W$ is connected.

A cycle in a graph $G = (V,E)$ of length $n \geq 3$ is a sequence $v_1, \ldots, v_n$ of distinct vertices such that $\{v_i, v_{i+1}\} \in E$ and $\{v_{n-1}, v_1\} \in E$ for all $i \in [2,n]$. Graph $G$ is acyclic, or a forest, if it has no cycle. $G$ is a tree if it is acyclic and connected. It will be a useful convention to call the vertices of trees nodes. A node of degree at most 1 is called a leaf. The set of all leaves of a tree $T$ is denoted by $L(T)$. Nodes that are not leaves are called inner nodes. A rooted tree is a triple $T = (V(T),E(T),r(T))$, where $(V(T),E(T))$ is a tree and $r(T) \in V(T)$ is a distinguished node called the root. A node $t$ of a rooted tree $T$ is the parent of a node $u$, and $u$ is a child of $t$, if $t$ is the predecessor of $u$ on the unique path from the root $r(T)$ to $u$. Two nodes that are children of the same parent are called siblings. A binary tree is a rooted tree $T$ in which every inner node has exactly two children.
2.2 Logic

I assume that the reader has a solid background in logic. To simplify matters, we only consider logics over (labelled) graphs, even though most results mentioned in this survey extend to more general structures. Let us briefly review the syntax and semantics of first-order logic FO and monadic second-order logic MSO. We assume that we have an infinite supply of individual variables, usually denoted by the lowercase letters \(x, y, z\), and an infinite supply of set variables, usually denoted by uppercase letters \(X, Y, Z\). First-order formulas in the language of graphs are built up from atomic formulas \(E(x, y)\) and \(x = y\) by using the usual Boolean connectives \(\neg\) (negation), \(\land\) (conjunction), \(\lor\) (disjunction), \(\rightarrow\) (implication), and \(\leftrightarrow\) (bi-implication) and existential quantification \(\exists x\) and universal quantification \(\forall x\) over individual variables. Individual variables range over vertices of a graph. The atomic formula \(\neg\phi\) means that \(\phi\) is false in a labelled graph, even though most results mentioned in this survey extend to more general structures. I assume that the reader has a solid background in logic. To simplify matters, we only consider logics over labelled graphs, even though most results mentioned in this survey extend to more general structures.

Example 2.1. A dominating set in a graph \(G = (V, E)\) is a set \(S \subseteq V\) such that for every \(v \in V\), either \(v\) is in \(S\) or \(v\) is adjacent to a vertex in \(S\).

The following first-order sentence \(\text{dom}_k\) says that a graph has a dominating set of size \(k\):

\[
\text{dom}_k = \exists x_1 \ldots \exists x_k \left( \bigwedge_{1 \leq i < j \leq k} x_i \neq x_j \land \forall y \bigvee_{i=1}^{k} (y = x_i \lor E(y, x_i)) \right).
\]

The following formula \(\text{dom}(X)\) says that \(X\) is a dominating set. More precisely, for every graph \(G\) and every subset \(S \subseteq V(G)\) it holds that \(G \models \text{dom}(S)\) if and only if \(S\) is a dominating set of \(G\).

\[
\text{dom}(X) = \forall y \left( X(y) \lor \exists z \left( X(z) \land E(z, y) \right) \right).
\]

Occasionally, we consider monadic second-order formulas that contain no second-order quantifiers, but have free set variables. We view such formulas as first-order formulas, because free set variables are essentially the same as labels (unary relation symbols). An example of such a formula is the formula \(\text{dom}(X)\) in the previous example. We say that a formula \(\phi(X)\) is positive in \(X\) if \(X\) only occurs in the scope of an even number of negation symbols. It is negative in \(X\) if \(X\) only occurs in the scope of an odd number of relation symbols.

Example 2.2. The following monadic second-order sentences \(\text{conn}\) and \(\text{acyc}\) say that a graph is connected and acyclic, respectively:

\[
\text{conn} = \exists xx = x \land \forall X \left( (\exists x X(x) \land \forall x \forall y \left( (X(x) \land E(x, y)) \rightarrow X(y) \right) \right)
\]

\[
\rightarrow \forall x X(x)) \right).
\]

\[
\text{acyc} = \neg \exists X \left( \exists x X(x) \land \forall X (X(x)
\]

\[
\rightarrow \exists y_1 \exists y_2 (y_1 \neq y_2 \land E(x, y_1) \land E(x, y_2) \land X(y_1) \land X(y_2)) \right).
\]
The sentence *acyc* is based on the simple fact that a graph has a cycle if and only if there is an induced subgraph in which every vertex has degree at least 2. Then the sentence *tree = conn & acyc* says that a graph is a tree.

The *quantifier rank* of a first-order or monadic second-order formula $\varphi$ is the nesting depth of quantifiers in $\varphi$. Let $G$ be a graph and $\bar{v} = (v_1, \ldots, v_k) \in V(G)^k$, for some nonnegative integer $k$. For every $q \geq 0$, the *first-order q-type* of $\bar{v}$ in $G$ is the set $tp_{q}^{FO}(G, \bar{v})$ of all first-order formulas $\varphi(x_1, \ldots, x_k)$ of quantifier rank at most $q$ such that $G \models \varphi(v_1, \ldots, v_k)$. The *monadic second-order q-type* of $\bar{v}$ in $G$, $tp_{q}^{MSO}(G, \bar{v})$ is defined analogously. As such, types are infinite sets, but we can syntactically *normalise* formulas in such a way that there are only finitely many normalised formulas of fixed quantifier rank and with a fixed set of free variables, and that every formula can effectively be transformed into an equivalent normalised formula. We represent a type by the set of normalised formulas it contains. There is a fine line separating decidable and undecidable properties of types and formulas. For example, it is decidable whether a formula is contained in a type: We just normalise the formula and test if it is equal to one of the normalised formulas in the type. It is undecidable whether a set of normalised formulas actually is (more precisely: represents) a type. To see this, remember that types are satisfiable by definition.

For a tuple $\bar{v} = (v_1, \ldots, v_k)$, we sloppily write $\{\bar{v}\}$ to denote the set $\{v_1, \ldots, v_k\}$. It will always be clear from the context whether $\{\bar{v}\}$ refers to the set $\{v_1, \ldots, v_k\}$ or the 1-element set $\{(v_1, \ldots, v_k)\}$. For tuples $\bar{v} = (v_1, \ldots, v_k)$ and $\bar{w} = (w_1, \ldots, w_l)$, we write $\bar{v} \bar{w}$ to denote the tuple $(v_1, \ldots, v_k, w_1, \ldots, w_l)$. We shall heavily use the following “Feferman–Vaught style” composition lemma.

**Lemma 2.3.** Let $tp$ be one of $tp^{FO}$, $tp^{MSO}$. Let $G, H$ be labelled graphs and $\bar{u} \in V(G)^l$, $\bar{v} \in V(G)^k$, $\bar{w} \in V(H)^m$ such that $V(G) \cap V(H) = \{\bar{u}\}$ (cf. Figure 2.1). Then for all $q \geq 0$, $tp_{q}(G \cup H, \bar{uvw})$ is determined by $tp_{q}(G, \bar{uv})$ and $tp_{q}(H, \bar{uw})$. Furthermore, there is an algorithm that computes $tp_{q}(G \cup H, \bar{uvw})$ from $tp_{q}(G, \bar{uv})$ and $tp_{q}(H, \bar{uw})$.

Let me sketch a proof of this lemma for first-order types. The version for monadic second-order types can be proved similarly, but is more complicated (see, for example, [53]).

**Proof sketch.** Let $G, H$ be labelled graphs and $\bar{u} \in V(G)^l$ such that $V(G) \cap V(H) = \{\bar{u}\}$. By induction on $\varphi$, we prove the following claim:

**Claim:** Let $\varphi(\bar{x}, \bar{y}, \bar{z})$ be a first-order formula of quantifier rank $q$, where $\bar{x}$ is a $k$-tuple and $\bar{y}, \bar{z}$ are tuples of arbitrary length. Then there is a Boolean combination $\Phi(\bar{x}, \bar{y}, \bar{z})$ of expressions $G \models \psi(\bar{x}, \bar{y})$ and $H \models \chi(\bar{z})$ for formulas $\psi, \chi$ of quantifier rank at most $q$, such that for all tuples $\bar{v}$ of vertices of $G$ and $\bar{w}$ of vertices of $H$ of the appropriate lengths it holds that

$$G \cup H \models \varphi(\bar{u}, \bar{v}, \bar{w}) \iff \Phi(\bar{u}, \bar{v}, \bar{w}).$$

Here $\Phi(\bar{u}, \bar{v}, \bar{w})$ denotes the statement obtained from $\Phi(\bar{x}, \bar{y}, \bar{z})$ by substituting $\bar{u}$ for $\bar{x}$, $\bar{v}$ for $\bar{y}$, and $\bar{w}$ for $\bar{z}$.

Furthermore, the construction of $\Phi$ from $\varphi$ is effective.

The claim holds for atomic formulas, because there are no edges from $G \setminus V(H)$ to $H \setminus V(G)$ in $G \cup H$. It obviously extends to Boolean combinations of formulas. So suppose that $\varphi(\bar{x}, \bar{y}, \bar{z}) = \exists \varphi_0(\bar{x}, \bar{y}, \bar{z})$. Let $\bar{v}, \bar{w}$ be tuples in $G, H$ of the appropriate lengths. By the induction hypothesis, there are $\Psi(\bar{x}, \bar{y}, \bar{z})$
and $\Psi_2(\bar{x}, \bar{y}, \bar{z}_0)$ such that

$$G \cup H \models \varphi(\bar{u}, \bar{v}, \bar{w})$$

$$\iff \exists v_0 \in V(G) \Psi_1(\bar{u}, \bar{v}, \bar{w}) \text{ or } \exists w_0 \in V(H) \Psi_2(\bar{u}, \bar{v}, \bar{w}).$$

We may assume that $\Psi_1$ is of the form

$$\bigvee_{i=1}^m (G \models \chi_i(\bar{x}, \bar{y}_0) \land H \models \xi_i(\bar{x}, \bar{z})).$$

Hence $\exists v_0 \in V(G) \Psi_1(\bar{u}, \bar{v}, \bar{w})$ is equivalent to

$$\bigvee_{i=1}^m (\exists v_0 \in V(G) G \models \chi_i(\bar{u}, \bar{v}, \bar{w}) \land H \models \xi_i(\bar{u}, \bar{w})).$$

We let $\Phi_1 = \bigvee_{i=1}^m (G \models \exists y_0 \chi_i(\bar{x}, \bar{y}_0) \land H \models \xi_i(\bar{x}, \bar{z})).$ Similarly, we define a $\Phi_2$ from $\Psi_2$, and then we let $\Phi = \Phi_1 \lor \Phi_2$. Clearly, the claim implies the statements of the lemma.

### 2.3 Algorithms and complexity

I assume that the reader is familiar with the basics of the design and analysis of algorithms. We will make extensive use of the Oh-notation. For example, we will denote the class of all polynomially bounded functions of one variable $n$ by $n^{O(1)}$. I also assume that the reader is familiar with standard complexity classes such as PTIME, NP, and PSPACE and with concepts such as reducibility between problems and hardness and completeness for complexity classes. If not specified otherwise, reductions are always polynomial time many-one reductions. The following example illustrates our notation for introducing algorithmic problems.

**Example 2.4.** The **dominating set problem** is defined as follows:

**DOMINATING-SET**

*Instance:* A graph $G$ and a natural number $k$.

*Problem:* Decide if $G$ has a dominating set of size $k$.

It is well-known that DOMINATING-SET is NP-complete.

We are mainly interested in algorithms for and the complexity of model checking problems. For every logic $L$ and every class $\mathcal{C}$ of graphs, we let:

**MC($L$, $\mathcal{C}$)**

*Instance:* A sentence $\varphi$ of $L$ and a graph $G \in \mathcal{C}$.

*Problem:* Decide if $G \models \varphi$.

If $\mathcal{C}$ is the class of all graphs, we just write MC($L$).

**Example 2.5.** Example 2.1 shows that DOMINATING-SET is reducible to MC(FO). Hence MC(FO) is NP-hard. As MC(FO) is trivially reducible to MC(MSO), the latter is also NP-hard.

**Fact 2.6 (Vardi [75]).** MC(FO) and MC(MSO) are PSPACE-complete.

This fact is often phrased as: “The combined complexity of FO resp. MSO is PSPACE-complete.” Combined complexity refers to both the sentence and the graph being part of the input of the model checking problem. Two principal ways of dealing with the hardness of model checking problems are restrictions of the logics and restrictions of the classes of input graphs. In this survey, we shall only consider restrictions of the classes of input graphs. As for restrictions of the logics, let me just mention that the model checking problem remains NP-hard even for the fragment of first-order logic whose formulas are the positive
primitive formulas, that is, existentially quantified conjunctions of atomic formulas. On the other hand, the model checking problem is in polynomial time for the bounded variable fragments of first-order logic [76].

Unfortunately, restricting the class of input graphs does not seem to improve the complexity, because the hardness result in Fact 2.6 can be strengthened: Let \( G \) be any graph with at least two vertices. Then it is PSPACE-hard to decide whether a given FO-sentence \( \varphi \) holds in the fixed graph \( G \). Of course this implies the corresponding hardness result for MSO. Hence not only the combined complexity, but also the expression complexity of FO and MSO is PSPACE-complete. The reason for this hardness result is that in graphs with at least two vertices we can take atoms of the form \( x = y \) to represent Boolean variables and use this to reduce the PSPACE-complete satisfiability problem for quantified Boolean formulas to the model checking problem. Let us explicitly state the following consequence of this hardness result, where we call a class of graphs nontrivial if it contains at least one graph with at least two vertices.

**Fact 2.7.** For every nontrivial class \( \mathcal{C} \) of graphs, the problems \( MC(\text{FO}, \mathcal{C}) \) and \( MC(\text{MSO}, \mathcal{C}) \) are PSPACE-hard.

So what can we possibly gain by restricting the class of input graphs of our model checking problems? As there are no polynomial time algorithms (unless \( \text{PTIME} = \text{PSPACE} \)) even for very simple classes \( \mathcal{C} \) of input graphs, we have to relax our notion of “tractability”. A drastic way of doing this is to consider data complexity instead of combined complexity, that is, consider the complexity of evaluating a fixed sentence of the logic in a given graph. The following fact implies that the data complexity of FO is in PTIME:

**Fact 2.8.** There is an algorithm that solves \( MC(\text{FO}) \) in time \( O(k^2 \cdot n^3) \), where \( n \) denotes the number of vertices of the input graph \( G \) and \( k \) the length of the input sentence \( \varphi \).

Even though FO and MSO have the same combined complexity and the same expression complexity, the following example shows that the two logics differ in their data complexity:

**Example 2.9.** It is easy to see that there is an MSO-formula \( 3\text{-col} \) saying that a graph is 3-colourable. As the 3-colourability problem is NP-complete, this shows that the data complexity of MSO is NP-hard.

There are, however, nontrivial classes \( \mathcal{C} \) of graphs such that the data complexity of MSO restricted to \( \mathcal{C} \) is in PTIME. As we shall see later, an example of such a class is the class of all trees. Thus things are starting to get interesting.

Still, while we have seen that polynomial combined complexity is too restrictive, polynomial data complexity may be too weak as a notion of tractability. Recall from the introduction that this survey is about algorithmic meta theorems, that is, uniform tractability results for classes of algorithmic problems defined in terms of logic. Fact 2.8 implies such a meta theorem: Every graph property definable in first-order logic can be decided in polynomial time. A serious drawback of this result is that it does not bound the degrees of the polynomial running times of algorithms deciding first-order properties. An important justification for PTIME being a reasonable mathematical model of the class of “tractable” (that is, efficiently solvable) problems is that most problems solvable in polynomial time are actually solvable by algorithms whose running time is bounded by polynomials of low degree, usually not more than three. However, this is not the case for parameterized families of polynomial time definable problems such as the family of first-order definable graph properties, for which the degree of the polynomials is unbounded. Or more plainly, even for a property that is defined by a fairly short first-order sentence, say, of length \( k = 10 \), an algorithm deciding this property in time \( O(n^{10}) \) hardly qualifies as efficient. A much more useful meta theorem would state that first-order definable graph properties can be decided “uniformly” in polynomial time, that is, in time bounded by polynomials of a fixed degree. Unfortunately, such a theorem does not seem to hold, at least not for first-order definable properties of the class of all graphs.

The appropriate framework for studying such questions is that of parameterized complexity theory [26, 38, 57]. A parameterized problem is a pair \((P, \kappa)\), where \( P \) is a decision problem in the usual sense and \( \kappa \) is a polynomial time computable mapping that associates a natural number, called the parameter, with each instance of \( P \).\(^1\)

\(^1\)At some places in this paper (the first time in Remark 3.19) we are dealing with “parameterized problems” where the parameterization is not polynomial time computable. Whenever this appears here, the parameterization is computable by an fpt algorithm (see below), and this is good enough for our purposes. The same issue is also discussed in Section 11.4 of [38].
Here we are mainly interested in model checking problems parameterized by the length of the input formula. For a logic $L$ and a class $\mathcal{C}$ of graphs, we let:

\begin{align*}
\text{p-MC}(L, \mathcal{C}) \\
\text{Instance:} & \quad \text{A sentence } \varphi \text{ of } L \text{ and a graph } G \in \mathcal{C}.
\text{Parameter:} & \quad |\varphi|.
\text{Problem:} & \quad \text{Decide if } G \models \varphi.
\end{align*}

A parameterized problem $(P, \kappa)$ is fixed-parameter tractable if there is an algorithm deciding whether an instance $x$ is in $P$ in time

$$f(\kappa(x)) \cdot |x|^c,$$

for some computable function $f$ and some constant $c$. We call an algorithm that achieves such a running time an \textit{fpt algorithm}. Slightly imprecisely, we call $f$ the \textit{parameter dependence} of the algorithm and $c$ the \textit{exponent}. An fpt algorithm with exponent 1 is called a \textit{linear fpt algorithm}. Similarly, fpt algorithms with exponents 2 and 3 are called \textit{quadratic} and \textit{cubic}. FPT denotes the class of all parameterized problems that are fixed-parameter tractable.

Hence a parameterized model checking problem is fixed-parameter tractable if and only if it is “uniformly” in polynomial time, in the sense discussed above. (By requiring the function $f$ bounding the running time to be computable, we impose a slightly stronger uniformity condition than above. This is inessential, but technically convenient.) Parameterized complexity theory is mainly concerned with the distinction between running times like $O(2^k \cdot n)$ (fpt) and $O(n^k)$ (not fpt). Running times of the latter type yield the parameterized complexity class XP. Intuitively, a problem is in XP if it can be solved by an algorithm whose running time is polynomial for fixed parameter values. Formally, XP is the class of all parameterized problems that can be decided in time

$$O(|x|^{f(\kappa(x))}),$$

for some computable function $f$. Hence essentially, the parameterized model checking problem for a logic is in XP if and only if the data complexity of the logic is polynomial time. The class XP strictly contains FPT; this is an easy consequence of the time hierarchy theorem.

There is an appropriate notion of \textit{fpt reduction} and a wide range of parameterized complexity classes between FPT and XP.

\textbf{Example 2.10.} A \textit{clique} in a graph is the vertex set of a complete subgraph. The parameterized clique problem is defined as follows:

\begin{align*}
p-\text{CLIQUE} \\
\text{Instance:} & \quad \text{A graph } G \text{ and a natural number } k.
\text{Parameter:} & \quad k.
\text{Problem:} & \quad \text{Decide if } G \text{ has a clique of size } k.
\end{align*}

It is easy to see that $p-\text{CLIQUE} \in \text{XP}$. It can be proved that $p-\text{CLIQUE}$ is complete for the parameterized complexity class $W[1]$ under fpt reductions.

\textbf{Example 2.11.} The parameterized dominating set problem is defined as follows:

\begin{align*}
p-\text{DOMINATING-SET} \\
\text{Instance:} & \quad \text{A graph } G \text{ and a natural number } k.
\text{Parameter:} & \quad k.
\text{Problem:} & \quad \text{Decide if } G \text{ has a dominating set of size } k.
\end{align*}

It is easy to see that $p-\text{DOMINATING-SET} \in \text{XP}$. It can be proved that $p-\text{DOMINATING-SET}$ is complete for the parameterized complexity class $W[2]$ under fpt reductions.
The parameterized complexity classes \( W[1] \) and \( W[2] \) form the first two levels of the so-called \( W\)-hierarchy of classes between \( \text{FPT} \) and \( \text{XP} \). Yet another parameterized complexity class, located between the \( W\)-hierarchy and \( \text{XP} \), is called \( \text{AW}[\ast] \). Thus we have

\[
\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq W[3] \subseteq \cdots \subseteq \text{AW}[\ast] \subseteq \text{XP}.
\]

It is conjectured that all containments between the classes are strict.

**Fact 2.12 (Downey, Fellows, and Taylor [27]).** \( p\text{-MC}(\text{FO}) \) is \( \text{AW}[\ast]\text{-complete under fpt reductions} \).

Observe that by Example 2.9, \( p\text{-MC}(\text{MSO}) \) is not even in \( \text{XP} \) unless \( \text{PTIME} = \text{NP} \).

This concludes our brief introduction to parameterized complexity theory. For proofs of all results mentioned in this section, I refer the reader to [38].

## 3 Monadic second-order logic on tree-like classes of graphs

The model checking problem for monadic second-order logic turns out to be tractable on trees and graph classes that are sufficiently similar to trees. A well-known measure for the similarity of a graph with a tree is **tree width**. In this article, however, we will work with **branch width** instead. The tree width and branch width of a graph are the same up to a factor of \( 3/2 \), so the results are essentially the same. Some of the results, including Courcelle’s theorem, may sound unfamiliar this way, but the reader can substitute “tree” for “branch” almost everywhere, and the results will remain true (up to constant factors, which we usually disregard anyway). Using branch width instead of tree width may make this article a bit more interesting for those who do not want to read the definition of tree width for the 100th time. However, the main reason for working with branch width is that it combines nicely with the other graph invariant that we shall study in this section, **rank width**. Indeed, both branch width and rank width of a graph are instances of the same abstract notion of branch width of a set function.

### 3.1 Trees

Let \( \mathcal{T} \) denote the class of all trees. Recall that then \( \mathcal{T}_{lb} \) denotes the class of labelled trees.

**Theorem 3.1 (Folklore).** \( p\text{-MC}(\text{MSO}, \mathcal{T}_{lb}) \) is solvable by a linear \( \text{fpt} \) algorithm.

We sketch two proofs of this theorem. Even though one may view them as “essentially the same”, the first is more natural from an algorithmic point of view, while the second will be easier to generalise later.

**First proof sketch.** Using a standard encoding of arbitrary trees in binary trees via the “first-child/next-sibling” representation, we can reduce the model checking problem for monadic second-order logic on arbitrary labelled trees to the model checking problem for monadic second-order logic on labelled binary trees. By a well-known theorem due to Thatcher and Wright [74], we can effectively associate a (deterministic) bottom-up tree automaton \( A_\psi \) with every MSO-sentence \( \psi \) over binary trees such that a binary tree \( T \) satisfies \( \psi \) if and only if the automaton \( A_\psi \) accepts \( T \). By simulating the run of \( A_\psi \) on \( T \), it can be checked in linear time whether \( A_\psi \) accepts a tree \( T \).

**Second proof sketch.** Again, we first reduce the model checking problem to binary trees. Let \( T \) be a labelled binary tree, and let \( \psi \) be a monadic second-order sentence, say, of quantifier rank \( q \). For every \( t \in V(T) \), let \( T_t \) be the subtree of \( T \) rooted in \( t \). Starting from the leaves, our algorithm computes \( tp^\text{MSO}_q(T_t, t) \) for every \( t \in T \), using Lemma 2.3. Then it decides if \( \psi \in tp^\text{MSO}_q(T, r) \) for the root \( r \) of \( T \).

The \( \text{fpt} \) algorithms described in the two proofs of Theorem 3.1 are linear in the size of the input trees. Clearly, this is optimal in terms of \( n \) (up to a constant factor). But what about the parameter dependence, that is, the function \( f \) in an \( \text{fpt} \) running time \( f(k) \cdot n \)? Recall that a function \( f : \mathbb{N}^q \to \mathbb{N} \) is **elementary** if it can be formed from the successor function, addition, subtraction, and multiplication using composition, projections, bounded addition and bounded multiplication of the form \( \sum_{\ell \leq m} g(n_1, \ldots, n_k, \ell) \) and \( \prod_{\ell \leq m} g(n_1, \ldots, n_k, \ell) \), respectively. It is easy to see that a function \( f : \mathbb{N}^q \to \mathbb{N} \) is elementary if and only if there is an \( h \geq 0 \) such that \( f(n) \leq \exp^{(h)}(n) \) for all \( n \in \mathbb{N} \). Here \( \exp^{(h)} \) denotes the \( h\)-fold exponentiation
defined by \( \exp^{(0)}(n) = n \) and \( \exp^{(h)}(n) = 2^\exp^{(h-1)}(n) \) for all \( n, h \in \mathbb{N} \). It is well known that there is no elementary function \( f \) such that the number of states of the smallest automaton \( A_\varphi \) equivalent to an MSO-formula \( \varphi \) of length \( k \) is at most \( f(k) \). It follows that the parameter dependence of our automata based \( \text{fpt} \) algorithm for \( p\text{-MC}(\text{MSO}, \mathcal{T}) \) is non-elementary. Similarly, the number of monadic second-order \( q \)-types is nonelementary in terms of \( q \), and hence the type based \( \text{fpt} \) algorithm also has a nonelementary parameter dependence. Of course this does not rule out the existence of other \( \text{fpt} \) algorithms with a better parameter dependence. The following theorem shows that, under reasonable complexity theoretic assumptions, no such algorithms exist, not even for first-order model checking:

**Theorem 3.2 (Frick and Grohe [42]).**

(1) Unless \( \text{PTIME} = \text{NP} \), there is no \( \text{fpt} \) algorithm for \( p\text{-MC}(\text{MSO}, \mathcal{T}) \) with an elementary parameter dependence.

(2) Unless \( \text{FPT} = \text{W}[1] \), there is no \( \text{fpt} \) algorithm for \( p\text{-MC}(\text{FO}, \mathcal{T}) \) with an elementary parameter dependence.

As almost all classes \( \mathcal{C} \) of graphs we shall consider in the following contain the class \( \mathcal{T} \) of trees, we have corresponding lower bounds for the model checking problems on these classes \( \mathcal{C} \). The only exception are classes of graphs of bounded degree, but even for such classes, we have a triply exponential lower bound [42] (cf. Remark 4.11).

### 3.2 Branch decompositions

We first introduce branch decompositions in an abstract setting and then specialise them to graphs in two different ways.

#### 3.2.1 Abstract branch decompositions

Let \( A \) be a nonempty finite set and \( \kappa : 2^A \to \mathbb{R} \). In this context, the function \( \kappa \) is often called a *connectivity function*. A branch decomposition of \( (A, \kappa) \) is a pair \((T, \beta)\) consisting of a binary tree \( T \) and a bijection \( \beta : V(T) \to A \). (Recall that \( L(T) \) denote the set of leaves of a tree \( T \).) We inductively define a mapping \( \beta : V(T) \to 2^A \) by letting

\[
\beta(t) = \begin{cases} 
\{\beta(t)\} & \text{if } t \text{ is a leaf}, \\
\beta(t_1) \cup \beta(t_2) & \text{if } t \text{ is an inner node with children } t_1, t_2.
\end{cases}
\]

The *width* of the branch decomposition \((T, \kappa)\) is defined to be the number

\[
\text{width}(T, \kappa) = \max \{ \kappa(\beta(t)) \mid t \in V(T) \},
\]

and the *branch width* of \((A, \kappa)\), denoted by \( bw(A, \kappa) \), is defined to be the minimum of the widths of all branch decompositions of \((A, \kappa)\). We extend the definition of branch width to empty ground sets \( A \) by letting \( bw(\emptyset, \kappa) = \kappa(\emptyset) \) for all \( \kappa : \{\emptyset\} \to \mathbb{R} \). Note that \((\emptyset, \kappa)\) does not have a branch decomposition, because the empty graph, not being connected, is not a tree.

Usually, the connectivity functions \( \kappa \) considered for branch decompositions are integer-valued, symmetric, and submodular. A function \( \kappa : 2^A \to \mathbb{R} \) is symmetrical if \( \kappa(B) = \kappa(A \setminus B) \) for all \( B \subseteq A \). The function \( \kappa \) is submodular if

\[
\kappa(B) + \kappa(C) \geq \kappa(B \cup C) + \kappa(B \cap C)
\]

for all \( B, C \subseteq A \).

**Example 3.3.** Let \( A \subseteq \mathbb{R}^n \) be finite. For every \( B \subseteq A \), let \( r(B) \) be the dimension of the linear subspace of \( \mathbb{R}^n \) generated by \( B \), or equivalently, the rank of the matrix with column vectors \( B \) (defined to be 0 if \( B = \emptyset \)). Define \( \kappa_{\text{lin}} : 2^A \to \mathbb{Z} \) by

\[
\kappa_{\text{lin}}(B) = r(B) + r(A \setminus B) - r(A).
\]
Example 3.4. Again, let the connectivity function be defined by $\kappa_t$ lin $\{1\}$. It is not hard to prove that $\kappa_t$ lin $\{1\}$ has a leaf $t \in L(T)$ with $\beta(t) = (1, 1, 1)^T$, and we have $\kappa_t$ lin $\{1\} = 1$.

Figure 3.1 shows two branch decompositions of $(A, \kappa_t)$ from Example 3.3

$\kappa_t$ lin measures the dimension of the intersection of the subspace generated by $B$ and the subspace generated by $A \setminus B$. It is easy to see that $\kappa_t$ lin is symmetric and submodular.

For example, let

$$A = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\} \subseteq \mathbb{R}^4.$$  

Figure 3.1 shows two branch decompositions of $(A, \kappa_t)$ lin $\{1\}$. I leave it as an exercise for the reader to verify that the first decomposition has width 1 and the second has width 2. Observe that $bw(A, \kappa_t)$ lin $\{1\} = 1$, because every branch decomposition $(T, \beta)$ of $(A, \kappa_t)$ lin $\{1\}$ has a leaf $t \in L(T)$ with $\beta(t) = (1, 1, 1)^T$, and we have $\kappa_t$ lin $\{1\} = 1$.

Example 3.4. Again, let $A \subseteq \mathbb{R}^n$. Now, for $B \subseteq A$ let $d(B)$ be the dimension of the affine subspace of $\mathbb{R}^n$ spanned by $B$ (defined to be $-1$ if $B = \emptyset$), and let

$$\kappa_{\text{aff}}(B) = d(B) + d(A \setminus B) - d(A).$$

It is not hard to prove that $\kappa_{\text{aff}}$ is also symmetric and submodular.

Figure 3.2 shows an example of a set $A = \{a, b, c, d, e, f, g, h\} \subseteq \mathbb{R}^2$ and a branch decomposition of $(A, \kappa_{\text{aff}})$ of width 1.

Example 3.5. The previous two examples have a common generalisation, which is known as the branch width of matroids.\(^2\) Let $M$ be a matroid with base set $A$ and rank function $r_M$. Then the function $\kappa : 2^A \rightarrow \mathbb{Z}$ defined by

$$\kappa_M(B) = r_M(B) + r_M(A \setminus B) - r_M(A)$$

is known as the connectivity function of the matroid.\(^3\) Obviously, $\kappa_M$ is symmetric, and as the rank function $r_M$ is submodular, $\kappa_M$ is also submodular.

\(^2\) Readers who do not know anything about matroids should not worry. This example is the only place in this survey where they appear.

\(^3\) Often, the connectivity function is defined by $\kappa_M(B) = r_M(B) + r_M(A \setminus B) - r_M(A) + 1$, but this difference is inessential here.
Before we return to graphs, let us state a very general algorithmic result, which shows that approximately optimal branch decompositions can be computed by an fpt algorithm. The proof of this theorem is beyond the scope of this survey. It is based on a deep algorithm for minimizing submodular functions due to Iwata, Fleischer and Fujishige [48].

When talking about algorithms for branch decompositions, we have to think about how the input of these algorithms is specified. Let $\mathcal{A}$ be a class of pairs $(A, \kappa)$, where $\kappa : 2^A \rightarrow \mathbb{Z}$ is symmetric and submodular and takes only nonnegative values. We call $\mathcal{A}$ a tractable class of connectivity functions, if we have a representation of the pairs $(A, \kappa) \in \mathcal{A}$ such that, given the representation of $(A, \kappa)$, the ground set $A$ can be computed in polynomial time, and for every $B \subseteq A$, the value $\kappa(B)$ can be computed in polynomial time.

For example, if $\mathcal{A}$ is the class of pairs $(A, \kappa_{\text{lin}})$, where $A$ is a finite set of vectors over some finite field or the field of rationals and $\kappa_{\text{lin}}$ is the linear connectivity function, then we can represent a pair $(A, \kappa_{\text{lin}})$ simply by a matrix whose columns are the vectors in $A$. For the graph based examples that we shall describe next, the underlying graph is always used as a representation.

**Theorem 3.6 (Oum and Seymour [59]).** Let $\mathcal{A}$ be a tractable class of connectivity functions. Then there is an fpt algorithm that, given $(A, \kappa) \in \mathcal{A}$ and a parameter $k \in \mathbb{N}$, computes a branch decomposition of $(A, \kappa)$ of width at most $3k$ if $bw(A, \kappa) \leq k$. If $bw(A, \kappa) > k$, the algorithm may still compute a branch decomposition of $(A, \kappa)$ of width at most $3k$, or it may simply halt without an output.\(^4\)

### 3.2.2 Branch decompositions of graphs

Let $G = (V, E)$ be a graph. For a set $F \subseteq E$, we define the *boundary* of $F$ to be the set $\partial F$ of all vertices of $G$ incident both with an edge in $F$ and with an edge in $E \setminus F$. We define a function $\kappa_G : 2^E \rightarrow \mathbb{Z}$ by letting $\kappa_G(F) = |\partial F|$ for every $F \subseteq E$. It is not hard to verify that $\kappa_G$ is symmetric and submodular. A *branch decomposition* of $G$ is a branch decomposition of $(E, \kappa_G)$, and the *branch width* $bw(G)$ of $G$ is the branch width of $(E, \kappa_G)$.

**Example 3.7.** Figure 3.3 shows an example of a graph and branch decomposition of this graph of width 5.

**Example 3.8 ([66]).** For every $n \geq 3$, the complete graph $K_n$ on $n$-vertices has branch width $\lceil 2n/3 \rceil$.

We omit the proof of the lower bound. For the upper bound, we partition the vertex set of $K_n$ into three parts $V_1, V_2, V_3$ of size $\lceil n/3 \rceil$ or $\lfloor n/3 \rfloor$, and we partition the edge set into three sets $E_{12}, E_{23}, E_{13}$ such that

\(^4\)An fpt algorithm of this type is known as an *fpt approximation algorithm* [7]
Figure 3.3. A graph with a branch decomposition of width 5. The numbers at the nodes indicate the size of the boundary of the edges in the subtree below that node.
edges in $E_{ij}$ are only incident with vertices in $V_i \cup V_j$. Then we take arbitrary branch decompositions of the three subgraphs $G_{ij} = (V_i \cup V_j, E_{ij})$ and join them together as indicated in Figure 3.4. 

Note that the construction of the previous example actually shows that every $n$-vertex graph has branch width at most $\lceil \frac{2n}{3} \rceil$. 

**Example 3.9 ([66]).** A graph has branch width 0 if and only if it has maximum degree at most 1. A graph has branch width 1 if and only if it has at least one vertex of degree greater than 1, and every connected component has at most one vertex of degree greater than 1.

Let me suggest it as an exercise for the reader to prove these simple facts. 

**Example 3.10 ([66]).** For all $n \geq 2$, the $n \times n$-grid 

$$G_{n \times n} = \left( [n] \times [n], \{ (i_1, j_1), (i_2, j_2) \mid |i_1 - i_2| + |j_1 - j_2| = 1 \} \right)$$

(cf. Figure 3.5) has branch width $n$. 

Branch width is closely related to the more familiar tree width. In fact, it is not very hard to prove the following inequalities for all graphs $G$ [66]:

$$bw(G) \leq tw(G) + 1 \leq \max \{ (3/2) \cdot bw(G), 2 \},$$

where $bw(G)$ denotes the tree width of $G$.

As the connectivity functions $\kappa_G$ are symmetric and submodular, approximately optimal branch decompositions can be computed by the general purpose algorithm of Theorem 3.6. However, for the special case of branch decompositions of graphs, better algorithms are known:

**Theorem 3.11 (Bodlaender and Thilikos [6]).** There is an algorithm that, given a graph $G$ and a $k \in \mathbb{N}$, decides if $bw(G) \leq k$ and computes a branch decomposition of $G$ of width at most $k$ if this is the case in time 

$$f(k) \cdot n,$$

where $n = |V(G)|$, for some computable function $f$. 

14
3.2.3 Rank decompositions of graphs

Whereas branch width is based on decompositions of the edge set of a graph, for rank width we decompose its vertex set. For a graph $G = (V,E)$ and subsets $U, W \subseteq V$ of its vertex set, we let $M = M_G(U,W)$ be the $|U| \times |W|$-matrix with entries $m_{uw}$ for $u \in U, w \in W$, where

$$m_{uw} = \begin{cases} 1 & \text{if } \{u,w\} \in E, \\ 0 & \text{otherwise}. \end{cases}$$

Hence $M_G(V,V)$ is just the adjacency matrix of $G$. We view $M$ as a matrix over the field $\text{GF}(2)$ and let $\text{rk}(M)$ be its row rank over $\text{GF}(2)$. Now we define a connectivity function $\rho_G : 2^V \to \mathbb{N}$ by

$$\rho_G(U) = \text{rk} \left( M(U,V \setminus U) \right)$$

for all $U \subseteq V$. Since the row rank and column rank of a matrix coincide, the function $\rho_G$ is symmetric, and it is not hard to prove that it is submodular. A rank decomposition of $G$ is a branch decomposition of $(V, \rho_G)$, and the rank width $\text{rw}(G)$ of $G$ is the rank width of $(V, \rho_G)$.

Example 3.12. Figure 3.6 shows an example of a graph and a rank decomposition of this graph of width 1.

It is easy to prove that rank width can be bounded in terms of branch width. The following theorem, which gives a tight bound, is not so obvious:

Theorem 3.13 (Oum [58]). For every graph $G$ holds that $\text{rw}(G) \leq \max\{1, \text{bw}(G)\}$.

The following example shows that the rank width of a graph can be substantially smaller than the branch width, and that it can also be the same.

Example 3.14. It is easy to see that every rank decomposition of a complete graph has width 1. Combined with Example 3.8, this shows that the branch width and rank width of a graph can differ by a factor $\Omega(n)$, where $n$ denotes the number of vertices.

Let $I(K_n)$ be the graph obtained from the complete $n$-vertex graph $K_n$ by subdividing all edges once, that is, by replacing every edge by a path of length 2. $I(K_n)$ is the incidence graph of $K_n$. Then if $n \geq 3$ and $n \equiv 0, 1 \mod 3$ we have $\text{rw}(I(K_n)) = \text{bw}(I(K_n)) = \lceil (2/3) \cdot n \rceil$ [58].

Example 3.15. It can be shown that the rank width of an $(n \times n)$-grid is at least $\lceil n/2 - 2 \rceil$ (follows from [58]). Hence grids have both large branch width and large rank width.

As for the branch width of graphs, there is an algorithm for computing rank width that is more efficient than the general purpose algorithm of Theorem 3.6.
Theorem 3.16 (Hlineny and Oum [31]). There is an algorithm that, given a graph $G$ and a $k \in \mathbb{N}$, decides if $rw(G) \leq k$ and computes a rank decomposition of $G$ of width at most $k$ if this is the case in time

$$f(k) \cdot n^3,$$

where $n = |V(G)|$, for some computable function $f$.

Rank width is related to the graph invariant clique width [17], which is defined in terms of a graph algebra: The clique width $cw(G)$ of a graph $G$ is the least number of constant symbols required in a term in this algebra that describes the graph $G$. Oum and Seymour [59] proved that for every graph $G$ it holds that

$$rw(G) \leq cw(G) \leq 2^{rw(G)+1} - 1.$$

In particular, this implies that a class of graphs has bounded rank width if and only if it has bounded clique width.

3.3 Courcelle’s Theorems

For every $k \geq 1$, let $\mathcal{B}_k$ be the class of all graphs of branch width at most $k$ and $\mathcal{R}_k$ the class of all graphs of rank width at most $k$. The following theorem is usually formulated in terms of tree width, but by (3.2) the following “branch width version” is equivalent.

**Courcelle’s Theorem ([9]).** For every $k$, the problem $p$-MC(MSO, $\mathcal{B}_k$) is solvable by a linear FPT algorithm.

As for Theorem 3.1, we sketch two proofs. The first is a reduction to Theorem 3.1, whereas the second is a generalisation of the second proof of Theorem 3.1.

**First proof sketch.** Let us fix $k \geq 1$. We reduce the model checking problem on the class $\mathcal{B}_k$ to that on labelled trees and then apply Theorem 3.1. We associate with each graph $G \in \mathcal{B}_k$ a labelled tree $T^+$ and with each MSO-sentence $\phi$ over graphs a sentence $\phi^+$ over labelled trees such that $G \models \phi$ if and only if $T^+ \models \phi^+$. We will do this in such a way that $T^+$ is computable from $G$ in linear time and that $\phi^+$ is computable from $\phi$. Then our model checking algorithm proceeds as follows: Given $G \in \mathcal{B}_k$ and $\phi \in$ MSO, it computes $T^+$ and $\phi^+$ and then tests if $T^+$ satisfies $\phi^+$ using the algorithm of Theorem 3.1.

The mapping $G \mapsto T^+$ will not be canonical, i.e., isomorphic graphs $G$ will not necessarily yield isomorphic trees $T^+$; the tree $T^+$ will depend on the specific representation of the input graph $G$ and on the algorithm we use to compute a branch decomposition of this input graph. Note that this does not affect the correctness of our algorithm.

We construct $T^+$ from $G$ as follows: Without loss of generality we assume that $G$ has no isolated vertices. We first compute a branch decomposition $(T, \beta)$ of $G$ of width at most $k$, which can be done in linear time by Theorem 3.11. Then we define a labelling of $T$ that allows us to reconstruct $G$ from the labelled tree $T^+$ within MSO. Formally, we define the labelling in such a way that $G$ is MSO-interpretable in $T^+$. Then we can construct $\phi^+$ from $\phi$ using the method of syntactic interpretations (see [30, 12]).

We assume that $T$ is an ordered binary tree, that is, each inner node has a left and a right child. Recall that, for a node $t$ of $T$, $\beta(t)$ is the set of all edges $e$ of $G$ such that $e = \beta(u)$ for some leaf $u$ of $T$ that appears in the subtree rooted at $t$. Let $B_t = \partial \beta(t)$ be the boundary of $\beta(t)$, that is, the set of all vertices incident with an edge in $\beta(t)$ and with an edge in $E(G) \setminus \beta(t)$. Since the width of $(T, \beta)$ is at most $k$ we have $|B_t| \leq k$ for all nodes $t$. The labelling of the tree $T^+$ encodes for every inner node $t$ with left child $t_1$ and right child $t_2$ how $B_t$ intersects the sets $B_{t_1}$ and $B_{t_2}$. We assume some linear order of the vertices of $G$. Then there will be labels $P_{ij}$ for $i, j \in [k]$, indicating that the $i$th vertex in $B_{t_1}$ is equal to the $j$th vertex in $B_{t_2}$, and similarly labels $P_{2ji}$ for $t_2$. Note that $B_t \subseteq B_{t_1} \cup B_{t_2}$, so these labels “determine” $B_t$. We do not label the leaves.

For each leaf $t$, the set $B_t$ consists of the two endpoints of the edge $\beta(t)$ (unless one or both endpoints have degree 1). It is easy to write down four MSO-sentences $eq_{ij}(x,y)$, for $i, j \in \{0, 1\}$, such that for all leaves $u, t$ of $T$ we have $T^+ \models eq_{ij}(u,v)$ if and only if the $i$th vertex in $B_u$ is equal to the $j$th vertex in $B_t$. Recalling our assumptions that $G$ has no isolated vertices, it is now easy to reconstruct $G$ from $T^+$ within MSO. This completes our proof sketch. □
Second proof sketch. Let $G$ be a graph of branch width $k$, and let $\varphi$ be an MSO-sentence, say, of quantifier rank $q$. We compute a branch decomposition $(T, \beta)$ of $G$ of width $k$. We fix some linear order on the vertices of $G$. For every $t \in V(T)$ we let $\bar{b}_t$ be the ordered tuple of the elements of $\partial \beta(t)$. Recall that for a subset $B \subseteq E(G)$, by $G[B]$ we denote the subgraph $(\bigcup B, B)$ generated by $B$.

Starting from the leaves we inductively compute $t_\varphi(G[\bar{\beta}(t)], \bar{b}_t)$ for all $t \in V(T)$, applying Lemma 2.3 at every node. For this to work it is important that for all nodes $t$ with children $t_1$ and $t_2$ it holds that

$$V(G[\bar{\beta}(t_1)] \cap G[\bar{\beta}(t_2)]) \subseteq \partial \bar{\beta}(t_1) \cup \partial \bar{\beta}(t_2)$$

and $\partial \bar{\beta}(t) \subseteq \partial \bar{\beta}(t_1) \cup \partial \bar{\beta}(t_2)$.

Finally, we check if $\varphi \in t_\varphi(G[\bar{\beta}(r)], \bar{b}_r)$ for the root $r$. (Note that $\bar{b}_r$ is actually the empty tuple, but this does not matter.)

The following theorem was first proved by Courcelle [8, 11] in a version phrased in terms of certain graph grammars. Later, a version for clique width was proved by Courcelle, Makowsky, and Rotics [14], and finally the relation between clique width and rank width was established by Oum and Seymour [59].

**Theorem 3.17 ([8, 11, 14, 59]).** For every $k$, the problem $p$-$MC(\mathcal{R}_k)$ is solvable by a cubic fpt algorithm.

**Proof sketch.** The proof follows the same strategy as the first proof of Courcelle’s Theorem: We fix $k$. For every graph $G \in \mathcal{R}_k$ we construct a labelled tree $T^*$ such that $G$ can be reconstructed from $T^*$ within MSO. Then using the method of syntactic interpretations, for every MSO-sentence $\varphi$ over graphs we obtain an MSO-sentence $\varphi^*$ over labelled trees such that $G \models \varphi \iff T^* \models \varphi^*$.

$T^*$ is obtained by suitably labelling the tree $T$ of a rank decomposition $(T, \beta)$ of $G$ of width $k$. The difficulty here is to encode $G$ in a labelling of $T$ that uses only finitely many labels. Let $t$ be an inner node of $T$ with children $t_1$ and $t_2$. For $i = 1, 2$, let $U_i = \bar{\beta}(t_i)$. Furthermore, let $U = U_1 \cup U_2$ and $W = V \setminus U$. Then $\bar{\beta}(t) = U$, and the matrices at the nodes $t_1, t_2, t$ can be written as

$$M(U_1, V \setminus U_1) = (M(U_1, U_2) \ M(U_1, W)),$n$$

$$M(U_2, V \setminus U_2) = (M(U_2, U_1) \ M(U_2, W)),$n$$

$$M(U, V \setminus U) = \begin{pmatrix} M(U_1, W) \\ M(U_2, W) \end{pmatrix}.$$n

Note that $M(U_2, U_1)$ is the transpose of $M(U_1, U_2)$.

For every node $t \in V(T)$ we compute a set $B_t$ of at most $k$ vertices of $G$ such that the rows corresponding to the vertices in $B_t$ form a basis of the row space of the matrix $M(U, V \setminus U)$, where $U = \bar{\beta}(t)$. We define a labelling of the (inner) nodes of $T$ as follows: Let $t$ be an inner node with children $t_1$ and $t_2$ and $U_1 = \bar{\beta}(t_1)$, $U_2 = \bar{\beta}(t_2)$, $U = U_1 \cup U_2 = \bar{\beta}(t)$. Then at $t$ the labelling encodes

- the matrix $M(B_{t_1}, B_{t_2})$,

- for $i = 1, 2$ and each $v \in B_i$, a representation of the row of $M(U, V \setminus U)$ corresponding to $v$ as a linear combination of vectors of the basis corresponding to $B_i$ over the field $GF(2)$.

Note that this amounts to at most $3k^2$ bits of information: The matrix requires at most $k^2$ bits, and a linear combination of $k$ vectors over $GF(2)$ requires $k$ bits.

We now describe how the graph $G$ can be reconstructed from the labelled tree $T^*$. The vertices of $G$ correspond to the leaves of $T^*$. To find out whether there is an edge between a vertex $v_1$, say, with $v_1 = \beta(u_1)$ and a vertex $v_2$, say with $v_2 = \beta(u_2)$, we proceed as follows: Let $t$ be the first common ancestor of $u_1$ and $u_2$, and let $t_1$ and $t_2$ be the children of $t$ such that $u_i$ is a descendant of $t_i$ for $i = 1, 2$. Let $U_i = \bar{\beta}(t_i)$ and $U = U_1 \cup U_2 = \bar{\beta}(t)$. Then $v_i \in U_i$. Note that $B_{t_1} = \{v_1\}$, because the matrices at the leaves only have one row. Hence, using the labelling, we can recursively find a representation of the row of the matrix $M(U_1, V \setminus U_i)$ corresponding to $v_1$ as a linear combination of the rows corresponding to $B_{t_i}$. Then we can use the matrix $M(B_{t_1}, B_{t_2})$, which is also part of the labelling, to compute the entry $m_{v_1v_2}$ of the matrix $M(U_1, U_2)$, and this entry tells us whether there is an edge between $v_1$ and $v_2$. The following example illustrates this construction. 

\[\square\]
Example 3.18. Consider the graph $G$ and branch decomposition displayed in Figure 3.6. We define the "bases" as follows:

<table>
<thead>
<tr>
<th>$t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_i$</td>
<td>{1}</td>
<td>{2}</td>
<td>{3}</td>
<td>{4}</td>
<td>{5}</td>
<td>{6}</td>
<td>{1}</td>
<td>{1}</td>
<td>{4}</td>
<td>{1}</td>
<td>{5}</td>
</tr>
</tbody>
</table>

Then for example, at node $b$ the following information is stored: The matrix

$$M(\{1\}, \{2\}) = (1),$$

and a representation of the rows $r_1 = (1 1 0 0 0 0)$ and $r_2 = (0 1 0 0 0 0)$ of the matrix $M(\{1,2,3\}, \{4,5,6\})$ in terms of the row $r_1$:

$$r_1 = 1 \cdot r_1, \quad r_2 = 0 \cdot r_1.$$

To determine whether there is an edge, say, between $v_1 = 3$ and $v_2 = 5$ we take the least common ancestor of the two leaves, $a$ with its two children $b$ and $d$. The representation of row $r_3 = (1 1 1)$ of $M(\{1,2,3\}, \{4,5,6\})$ with respect to $B_b = \{1\}$ is $r_3 = 1 \cdot r_1$, and the representation of row $r_5 = (1 0 1)$ of $M(\{4,5,6\}, \{1,2,3\})$ with respect to $B_d = \{4\}$ is $r_5 = 1 \cdot r_1$. Hence $m_{35} = 1 \cdot 1 \cdot m_{14} = 1$, that is, there is an edge between 3 and 5.

It follows from Theorem 3.2 that the parameter dependence of the fpt algorithms in the previous two theorems has to be nonelementary.

We close this section with two remarks about strengthenings of the two theorems:

Remark 3.19. Our proofs yield stronger theorems than stated: Not only is the MSO model checking problem fixed-parameter tractable on every class of graphs whose branch width is bounded, but actually the following doubly parameterized model checking problem is fixed-parameter tractable:

- **Instance:** A sentence $\varphi \in \text{MSO}$ and a graph $G$.
- **Parameter:** $|\varphi| + \text{bw}(G)$.
- **Problem:** Decide if $G \models \varphi$.

The same is true for rank width.

Remark 3.20. It is easy to see that both theorems can be extended to labelled graphs.

Courcelle’s Theorem even holds for a stronger monadic second order logic, denoted by $\text{MSO}_2$, that admits quantification not only over sets of vertices of a graph, but also over sets of edges. This stronger result can easily be derived from the (labelled) version of Courcelle’s Theorem. Define the incidence graph $I(G)$ of a graph $G$ to be the graph $(V_l, E_l)$, where $V_l = V(G) \cup E(G)$ and $E_l = \{ \{v, e\} \mid v \in e \}$. It is not hard to see that for every graph $G$ of branch width at least 2 it holds that $\text{bw}(G) = \text{bw}(I(G))$. Furthermore, every $\text{MSO}_2$-formula over $G$ can be translated to an MSO-formula over the labelled incidence graph $(I(G), P)$, where $P = E(G)$ (The labelling is not really needed, but convenient.) Hence it follows from Courcelle’s Theorem that $\text{p-MC}(\text{MSO}_2, \mathcal{F}_k)$ has a linear fpt algorithm for every $k \geq 1$.

This does not work for rank width, because the rank width of the incidence graph can be much larger than that of the original graph. Surprisingly, the rank width of the incidence graph of a graph is closely related to the branch width of the original graph. Oum [58] proved that

$$\text{bw}(G) - 1 \leq \text{rw}(I(G)) \leq \text{bw}(G)$$

for every graph $G$ with at least one vertex of degree 2.

### 4 First-order logic on locally tree-like classes of graphs

There is not much hope for extending the tractability of monadic second-order model checking to further natural classes of graphs such as planar graphs or graphs of bounded degree. Indeed, the $\text{MSO}$-definable 3-colourability problem is NP-complete even when restricted to planar graphs of degree 4. For first-order logic, however, the model checking problem is tractable on much larger classes of graphs. Seese [71] showed that first-order model checking admits a linear fpt algorithm on all classes of bounded degree. Later Frick and Grohe [41] proved the same for planar graphs, essentially by the general approach that we shall describe in this section. The crucial property of first-order logic that we exploit is its **locality**.
4.1 Locality

Let $G = (V, E)$ be a graph. The distance $\text{dist}^G(v,w)$ between two vertices $v,w \in V$ is the length of the shortest path from $v$ to $w$. For every $v \in V$ and $r \in \mathbb{N}$, the $r$-neighbourhood of $v$ in $G$ is the set

$$N^G_r(v) = \{ w \in V \mid \text{dist}^G(v,w) \leq r \}$$

of all vertices of distance at most $r$ from $v$. For a set $W \subseteq V$, we let $N^G_r(W) = \bigcup_{w \in W} N^G_r(w)$. We omit the superscript $G$ if $G$ is clear from the context. The radius of a connected graph $G$ is the least $r$ for which there is a vertex $v \in V(G)$ such that $V(G) \subseteq N_r(v)$. The radius of a disconnected graph is $\infty$.

Observe that distance is definable in first-order logic, that is, for every $x,y \in V$, $\text{dist}^G(x,y)$ is a vertex.

A first-order formula $\varphi(x)$ is $r$-local, if for all graphs $G,G'$ and all $v \in V(G),v' \in V(G')$ such that $G[N_r(v)]$ is isomorphic to $G'[N_r(v')]$ we have

$$G \models \varphi(v) \iff G' \models \varphi(v').$$

This means that it only depends on the $r$-neighbourhood of a vertex whether an $r$-local formula holds at this vertex.

A basic local sentence is a first-order sentence of the form

$$\exists x_1 \ldots \exists x_k \left( \bigwedge_{1 \leq i < j \leq k} \text{dist}(x_i,x_j) > 2r \land \bigwedge_{i=1}^k \varphi(x_i) \right),$$

where $\varphi(x)$ is $r$-local.

Gaifman’s Locality Theorem ([43]). Every first-order sentence is equivalent to a Boolean combination of basic local sentences.

Furthermore, there is an algorithm that computes a Boolean combination of basic local sentences equivalent to a given first-order sentence.

It has recently been proved [20] that the translation of a first-order sentence into a Boolean combination of basic local sentences may involve a nonelementary blow-up in the size of the sentence.

4.2 Localisations of graph invariants

Recall that $\mathcal{G}$ denotes the class of all graphs. For every graph invariant $f : \mathcal{G} \rightarrow \mathbb{N}$ we can define its localisation $\ell_f : \mathcal{G} \times \mathbb{N} \rightarrow \mathbb{N}$ by

$$\ell_f(G,r) = \max \left\{ f(G[N_r(v)]) \mid v \in V(G) \right\}.$$}

Hence to compute $\ell_f(G,r)$, we apply $f$ to every $r$-neighbourhood in $G$ and then take the maximum. We say that a class $\mathcal{C}$ of graphs has locally bounded $f$ if there is a computable\footnote{It would be more precise to call this notion “effectively locally bounded $f$”, but this would make the terminology even more awkward.} function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that $\ell_f(G,r) \leq g(r)$ for all $G \in \mathcal{C}$ and all $r \in \mathbb{N}$.

Example 4.1. One of the simplest graph invariants is the order of a graph. Observe that a class of graphs has locally bounded order if and only if it has bounded degree.

Moreover, if a class $\mathcal{C}$ has bounded degree then it has locally bounded $f$ for every computable graph invariant $f$. \hfill \qed
In this section, we are mainly interested in the localisation of branch width. Maybe surprisingly, there are several natural classes of graphs of locally bounded branch width. We start with two trivial examples and then move on to more interesting ones:

**Example 4.2.** Every class of graphs of bounded branch width has locally bounded branch width.

**Example 4.3.** Every class of graphs of bounded degree has locally bounded branch width. This follows immediately from Example 4.1.

**Example 4.4 ([64, 33]).** The class of planar graphs has locally bounded branch width. More precisely, a planar graph of radius \( r \) has branch width at most \( 2r+1 \).

Let me sketch the proof. Let \( G \) be a planar graph of radius \( r \), and let \( v_0 \) be a vertex such that \( V(G) \subseteq N_r(v_0) \). We show how to recursively partition the edge set of \( G \) in such a way that at each stage, the boundary of each part has cardinality at most \( 2r+1 \). This will give us a branch decomposition of width at most \( 2r+1 \).

Without loss of generality we may assume that \( G \) is 2-connected; if it is not, we first decompose it into its 2-connected blocks. Figure 4.1 illustrates the following steps. We fix a planar embedding of \( G \), and let \( C \) be the exterior cycle. We pick two vertices \( v, w \) on \( C \) and shortest paths \( P, Q \) from \( v_0 \) to \( v, w \), respectively. Then we cut along \( P \) and \( Q \). This gives us a partition of \( E(G) \) into two parts whose boundary is contained in \( V(P \cup Q) \). We can add the edges in \( E(P \cup Q) \) arbitrarily to either of the two parts. Now we consider each of the parts separately. The boundary cycle consists of \( P, Q \), and a piece of the cycle \( C \). If this piece of \( C \) is just one edge, we can split it off and then further decompose the rest. Otherwise, we pick a vertex \( x \) on the piece of \( C \) and a shortest path \( R \) from \( v_0 \) to \( x \). We obtain two new parts with boundaries \( V(P \cup R) \) and \( V(Q \cup R) \). We partition these new parts recursively until they only consist of their boundaries, and then we partition the rest arbitrarily. Of course this proof sketch omits many details and special cases. For example, the vertex \( v_0 \) could be on the boundary cycle to begin with. I leave it to the reader to work out these details.

The branch decomposition in Figure 3.3 was obtained by this method. Note that the graph has radius 2, with centre \( v_0 \) being the vertex incident with the edges \( m \) and \( j \). The initial paths \( P \) and \( Q \) have edge sets \( E(P) = \{s, m\} \) and \( E(Q) = \{j\} \). The right part consists of the edges \( a, b, c, k, d, e, f, l, o, n, u, t, w, p, v, q \). The edges of \( P \cup Q \) were added to the left part. In the next step, the right part was split along the path \( R \) with \( E(R) = \{k, e\} \). The right part of this split consists of the edges \( f, l, o, n, u, t, w, p, v, q \). The edge \( f \) immediately can be split off, and the new boundary cycle is \( w, q, l, k, m, s \). The new splitting path consists of the edge \( o \), et cetera.

**Example 4.5 ([33]).** The genus of a graph is the minimum genus of an orientable or nonorientable surface the graph can be embedded into. For every \( k \), the class of all graphs of genus at most \( k \) has locally bounded branch width. Moreover, for every \( k \) the class of all graphs of crossing number at most \( k \) has locally bounded branch width.

In the next example, we shall construct an artificial class of graphs of locally bounded branch width. It serves as an illustration that the global structure of graphs of locally bounded branch width can be quite complicated. In particular, this example shows that there are classes of graphs of locally bounded branch width and of unbounded average degree. Recall that if a class \( \mathcal{C} \) of graphs has unbounded average degree then the size of the graphs in \( \mathcal{C} \) is superlinear in the order. The graph classes in all previous examples have bounded average degree and thus size linear in the order. For planar graphs and graphs of bounded genus, this follows from Euler's formula.

**Example 4.6 ([41]).** Recall that the girth of a graph is the length of its shortest cycle, and the chromatic number is the least number of colours needed to colour the graph in such a way that no two adjacent vertices receive the same colour. We shall use the well-known fact, due to Erdős ([34]), that for all \( g, k \geq 1 \) there exist graphs of girth greater than \( g \) and chromatic number greater than \( k \). The proof of this fact (see [2]) shows that we can effectively, though maybe not efficiently, construct such a graph \( G_{g,k} \) for given \( g \) and \( k \).

Then for every \( k \geq 1 \), every graph \( G_{k,1} \) must have a subgraph \( H_k \) of minimum degree at least \( k \); otherwise we could properly colour \( G \) with \( k \) colours by a straightforward greedy algorithm (see [25], Corollary 5.2.3). Let \( H_k \subseteq G_{k,k} \) be such a subgraph. As a subgraph of \( G_{k,k} \) the graph \( H_k \) still has girth greater than \( k \).

Let \( \mathcal{C} = \{H_k \mid k \geq 1\} \). Then \( \mathcal{C} \) has unbounded minimum degree and hence unbounded average degree. Nevertheless, \( \mathcal{C} \) has locally bounded branch width. To see this, simply observe that for \( k \geq 2r+1 \), the
Figure 4.1. Schematic branch decomposition of a planar graph
Theorem 4.7. Let \( f \) be a graph invariant such that the following parameterization of the model checking problem for first-order logic is fixed-parameter tractable:

\[
\text{p-MC(FO,} f\text{)}
\]

*Instance:* A sentence \( \varphi \in \text{FO} \) and a labelled graph \( G \).

*Parameter:* \( |\varphi| + f(G) \).

*Problem:* Decide if \( G \models \varphi \).

Then for every class \( \mathcal{C} \) of graphs of locally bounded \( f \), the problem p-MC(FO,\( \mathcal{C} \)) is fixed-parameter tractable.

The proof of the theorem relies on Gaifman’s Locality Theorem and the following lemma:

Lemma 4.8 ([41]). Let \( f \) and \( \mathcal{C} \) be as in Theorem 4.7. Then the following problem is fixed-parameter tractable:

*Instance:* A labelled graph \( G = (V, E, P) \in \mathcal{C}_{lb} \) and \( k, r \in \mathbb{N} \).

*Parameter:* \( k + r \).

*Problem:* Decide if there are vertices \( v_1, \ldots, v_k \in P \) such that \( \text{dist}(v_i, v_j) > 2r \) for \( 1 \leq i < j \leq k \).

For simplicity, we only prove the lemma for graph invariants \( f \) that are induced-subgraph-monotone, that is, for all graphs \( G \) and induced subgraphs \( H \subseteq G \) we have \( f(H) \leq f(G) \). Note that both branch width and rank width are induced-subgraph-monotone.

Proof sketch of Lemma 4.8. Given \( G = (V, E, P) \) and \( k, r \in \mathbb{N} \), we first compute a maximal (with respect to inclusion) set \( S \subseteq P \) of vertices of pairwise distance greater than \( 2r \). If \( |S| \geq k \), then we are done.

Otherwise, we know that \( P \subseteq N_{2r}(S) \). Let \( H \) be the induced subgraph of \( G \) with vertex set \( N_{2r}(S) \). As \( |S| < k \), the radius of each connected component of \( H \) is at most \((3r+1) \cdot k\). Hence, because \( f \) is induced-subgraph-monotone,

\[
f(H) \leq \ell_f(G, (3r+1) \cdot k) \leq g((3r+1) \cdot k),
\]

where \( g \) is a function witnessing that \( \mathcal{C} \) has locally bounded \( f \).

Since \( P \subseteq N_{2r}(S) \) and \( V(H) = N_{3r}(S) \), for all vertices \( v, w \in P \) it holds that \( \text{dist}^G(v, w) > 2r \) if and only if \( \text{dist}^H(v, w) > 2r \). Hence it remains to check whether \( H \) contains \( k \) vertices labelled \( P \) of pairwise distance greater than \( 2r \). This is equivalent to saying that \( H \) satisfies the first-order sentence

\[
\exists x_1 \ldots \exists x_k \left( \bigwedge_{1 \leq i < j \leq k} \text{dist}(x_i, x_j) > 2r \land \bigwedge_{i=1}^k P(x_i) \right).
\]

We can use an fpt algorithm for p-MC(FO,\( f \)) to check this.

Proof sketch of Theorem 4.7. Let \( G = (V, E) \in \mathcal{C} \) and \( \varphi \in \text{FO} \). We first transform \( \varphi \) into an equivalent Boolean combination of basic local sentences. Then we check separately for each basic local sentence in this Boolean combination whether it is satisfied by \( G \) and use the results to determine whether \( \varphi \) holds.
Corollary 4.10. Fixed-parameter tractable.

Corollary 4.11. p-algorithm in the order (and thus an algorithm that is linear in the order cannot even read the whole input graph). It is an open question whether for every class $C$ we first look at the exponent of the fpt algorithms. An analysis of the algorithms described above shows that for every class $\mathcal{C}$ of locally bounded branch width, this yields a quadratic fpt algorithm, and for classes of locally bounded rank width, it yields an fpt algorithm with exponent four.

For classes $\mathcal{C}$ of locally bounded branch width, the exponent can be brought arbitrarily close to 1; more precisely, for every $\varepsilon > 0$ there is an fpt algorithm for $p$-$\text{MC}(\text{FO}, \mathcal{C})$ with running time of $f(k) \cdot |G|^{1+\varepsilon}$ [41]. Note that we cannot hope to find an fpt algorithm that is linear in the order for general classes of locally bounded branch width, because by Example 4.6 there are classes $\mathcal{C}$ of locally bounded branch width and unbounded average degree, which implies that the size of the graphs in $\mathcal{C}$ is not linearly bounded in the order (and thus an algorithm that is linear in the order cannot even read the whole input graph). It is an open question whether for every class $\mathcal{C}$ of graphs of locally bounded branch width there is an fpt algorithm $p$-$\text{MC}(\text{FO}, \mathcal{C})$ that is linear in the size $|G|$ of the input graph.

For specific classes $\mathcal{C}$, such as the class of planar graphs and classes of bounded genus or bounded degree, it is known that there are fpt algorithms that are linear in the order [41, 71].

Finally, let us look at the parameter dependence of the fpt algorithms. In general, it is again nonelementary by Theorem 3.2, because our classes contain the class of all trees. However, classes of graphs of bounded degree do not contain all trees, and it turns out that for such classes there are fpt algorithms with an elementary parameter dependence. For the class $\mathcal{D}_k$ of graphs of degree at most $k \geq 3$, there is a linear fpt algorithm for $p$-$\text{MC}(\text{FO}, \mathcal{D}_k)$ with a triply exponential parameter dependence, and there is a matching lower bound, which even holds on labelled binary trees [42].

5 Digression: Graph minor theory

A graph $H$ is a minor of a graph $G$ if $H$ can be obtained from $G$ by deleting vertices, deleting edges, and contracting edges. Contracting an edge means removing the edge, identifying its two end vertices, and possibly removing the resulting parallel edges. Figure 5.1 illustrates this. We write $H \preceq G$ if $H$ is

So let us consider a basic local sentence

$$\psi = \exists x_1 \ldots \exists x_k \left( \bigwedge_{1 \leq i < j \leq k} \text{dist}(x_i, x_j) > 2r \wedge \bigwedge_{i=1}^{k} \chi(x_i) \right),$$

where $\chi(x)$ is $r$-local. For each vertex $v$ of $G$ we check whether $G[N_r(v)]$ satisfies $\chi(v)$ using an fpt algorithm for $p$-$\text{MC}(\text{FO}, f)$. We can do this within the desired time bounds because $f(G[N_r(v)]) \leq \ell_f(G, r)$. If $G[N_r(v)]$ satisfies $\chi(v)$, we label $v$ by $P$. To determine whether $G$ satisfies $\psi$, we have to check whether the labelled graph $(V, E, P)$ has $k$ vertices in $P$ of pairwise distance greater than $2r$. By Lemma 4.8, this can be done by an fpt algorithm.

Corollary 4.9 ([41]). For every class $\mathcal{C}$ of graphs of locally bounded branch width, $p$-$\text{MC}(\text{FO}, \mathcal{C})$ is fixed-parameter tractable.

Corollary 4.10. For every class $\mathcal{C}$ of graphs of locally bounded rank width, $p$-$\text{MC}(\text{FO}, \mathcal{C})$ is fixed-parameter tractable.

Let me close this section with a few remarks on the running time of the model checking algorithms.

Remark 4.11. We first look at the exponent of the fpt algorithms. An analysis of the algorithms described above shows that for every class $\mathcal{C}$ of locally bounded $f$ we obtain an fpt algorithm for $p$-$\text{MC}(\text{FO}, \mathcal{C})$ with exponent $c + 1$, where $c$ is the exponent of an fpt algorithm for $p$-$\text{MC}(\text{FO}, f)$. Hence for classes of locally bounded branch width, this yields a quadratic fpt algorithm, and for classes of locally bounded rank width, it yields an fpt algorithm with exponent four.

For classes $\mathcal{C}$ of locally bounded branch width, the exponent can be brought arbitrarily close to 1; more precisely, for every $\varepsilon > 0$ there is an fpt algorithm for $p$-$\text{MC}(\text{FO}, \mathcal{C})$ with running time of $f(k) \cdot |G|^{1+\varepsilon}$ [41]. Note that we cannot hope to find an fpt algorithm that is linear in the order for general classes of locally bounded branch width, because by Example 4.6 there are classes $\mathcal{C}$ of locally bounded branch width and unbounded average degree, which implies that the size of the graphs in $\mathcal{C}$ is not linearly bounded in the order (and thus an algorithm that is linear in the order cannot even read the whole input graph). It is an open question whether for every class $\mathcal{C}$ of graphs of locally bounded branch width there is an fpt algorithm $p$-$\text{MC}(\text{FO}, \mathcal{C})$ that is linear in the size $|G|$ of the input graph.

For specific classes $\mathcal{C}$, such as the class of planar graphs and classes of bounded genus or bounded degree, it is known that there are fpt algorithms that are linear in the order [41, 71].

Finally, let us look at the parameter dependence of the fpt algorithms. In general, it is again nonelementary by Theorem 3.2, because our classes contain the class of all trees. However, classes of graphs of bounded degree do not contain all trees, and it turns out that for such classes there are fpt algorithms with an elementary parameter dependence. For the class $\mathcal{D}_k$ of graphs of degree at most $k \geq 3$, there is a linear fpt algorithm for $p$-$\text{MC}(\text{FO}, \mathcal{D}_k)$ with a triply exponential parameter dependence, and there is a matching lower bound, which even holds on labelled binary trees [42].
isomorphic to a minor of $G$. A minor mapping from $H$ to $G$ is a mapping $\mu$ that associates with each $v \in V(H)$ a connected subgraph $\mu(v) \subseteq G$ and with each $e \in E(H)$ an edge $\mu(e) \in E(G)$ such that:

- for all $v \neq w$, the graphs $\mu(v)$ and $\mu(w)$ are vertex disjoint;
- for all $e = \{v, w\} \in E(H)$, the edge $\mu(e)$ is incident to a vertex $v' \in V(\mu(v))$ and a vertex $w' \in V(\mu(w))$.

It is easy to see that $H \preceq G$ if and only if there is a minor mapping from $H$ to $G$. Observe that the graphs $\mu(v)$ of a minor mapping $\mu$ can be chosen to be trees. If $\mu$ is a minor mapping from $H$ to $G$, we call the subgraph $\mu(H) = \bigcup_{v \in V(H)} V(\mu(v)) \cup \bigcup_{e \in E(H)} E(\mu(e)) \cup \{\mu(e) \mid e \in E(H)\}$.

of $G$ an image of $H$ in $G$. Figure 5.2 shows an example.

For every graph $H$, we let $X(H) = \{G \mid H \not\preceq G\}$. We say that a class $\mathcal{C}$ of graphs excludes $H$ if $\mathcal{C} \subseteq X(H)$. For a class $\mathcal{H}$ of graphs, we let

$$X(\mathcal{H}) = \bigcap_{H \in \mathcal{H}} X(H) = \{G \mid H \not\preceq G \text{ for all } H \in \mathcal{H}\}.$$ 

A class $\mathcal{C}$ of graphs is minor-closed if for every graph $G \in \mathcal{C}$ and every $H \preceq G$ it holds that $H \in \mathcal{C}$. Observe that a class $\mathcal{C}$ of graphs is minor-closed if and only if it can be defined by excluding minors, that is, there is a class $\mathcal{H}$ such that $\mathcal{C} = X(\mathcal{H})$. (Just take $\mathcal{H} = \mathcal{I} \setminus \mathcal{C}$.) Robertson and Seymour proved that every minor-closed class of graphs can actually be defined by excluding finitely many minors:

**Graph Minor Theorem (Robertson and Seymour [69]).** For every minor-closed class $\mathcal{C}$ of graphs there is a finite class $\mathcal{F}$ of graphs such that $\mathcal{C} = X(\mathcal{F})$.

Many natural classes of graphs are minor-closed:

**Example 5.1.** Every cycle can be contracted to a triangle $K_3$. Hence the class of forests (acyclic graphs) is precisely $X(K_3)$.

**Example 5.2.** For every $k \geq 1$, the class $\mathcal{B}_k$ of all graphs of branch width $k$ is minor-closed. Let me suggest it as an exercise for the reader to prove this. Furthermore, it holds that $\mathcal{B}_2 = X(K_4)$ [66].

**Example 5.3.** Series-parallel graphs and outerplanar graphs exclude $K_4$. It can be shown that $X(K_4)$ is precisely the class of all graphs that are subgraphs of series-parallel graphs (see [25], Exercise 7.32). $X(\{K_4, K_{2,3}\})$ is the class of outerplanar graphs (see [25], Exercise 4.20).

6In the literature, the term "model" is used instead of "image". We prefer "image" here to avoid confusion with "models" in the logical sense.
Example 5.4. By Kuratowski’s well-known theorem [50] (or, more precisely, by a variant due to Wagner [77]), the class of planar graphs is $\mathcal{X}(\{K_5, K_{3,3}\})$.

Example 5.5. For every $k \geq 0$, the class of all graphs of genus $k$ is minor-closed.

Note that all previous examples of minor-closed classes also have locally bounded branch width. But this is an coincidence, as the following example shows.

Example 5.6. A graph $G$ is an apex graph if there is a vertex $v \in V(G)$ such that $G \setminus \{v\}$ is planar. The class of all apex graphs is minor-closed.

The class of apex graphs does not have locally bounded branch width. To see this, consider the “pyramid graphs” $P_n$ obtained from the $(n \times n)$-grid $G_{n \times n}$ by adding a new vertex and connecting it to all vertices of the grid. Obviously, the pyramid graphs are apex graphs, and for every $n \geq 1$ we have

$$\ell_{bw}(P_n, 1) \geq bw(G_{n \times n}) \geq n,$$

where the second inequality holds by Example 3.10.

Example 5.7. A graph is knot free if it can be embedded into $\mathbb{R}^3$ in such a way that no cycle of the graph is knotted in a nontrivial way. It is easy to see that the class of all knot free graphs is minor-closed.

Similarly, the class of all graphs that can be embedded into $\mathbb{R}^3$ in such a way that no pair of cycles is linked is minor-closed.

Let me also mention a “non-example”: The class of all graphs of crossing number $k \geq 1$ is not minor-closed.

5.1 Structure theory

The proof of the graph minor theorem relies on a deep structure theory for classes of graphs with excluded minors. While it is far beyond the scope of this survey to describe this theory in adequate detail, or even give a precise statement of the main structural result, I would like to give the reader a glimpse of the theory, because the model checking algorithms for graphs with excluded minors heavily rely on it. Let me start with a disclaimer: The following intuitive remarks may make a nice story, but they not always reflect the actual proofs and thus should be taken with some care.

Suppose we have a class $\mathcal{C}$ with excluded minors. Then $\mathcal{C} \subseteq \mathcal{X}(K_k)$ for some $k$, because every graph is a minor of some complete graph. We fix $\mathcal{C}$ and $k$ for the rest of this section. We want to describe the structure of the graphs in $\mathcal{C}$ by “decomposing” them into “simple” building blocks. We shall define later what exactly we mean by “decomposing” a graph. For now, let us just remark that if a graph has bounded branch width, then we can decompose it into pieces of bounded size. Thus we are mainly interested in classes $\mathcal{C}$ of unbounded branch width. The following theorem, which is one of the fundamental results of the whole theory, gives us a handle on the structure of graphs of unbounded branch width:

Excluded Grid Theorem (Robertson and Seymour [65]). There is a computable function $f$ such that for every $k \geq 1$ and every graph $G$, if $bw(G) \geq f(k)$ then $G_{k\times k} \not\subseteq G$.

A proof of this theorem can be found in [25].

The Excluded Grid Theorem tells us that if our class $\mathcal{C}$ has unbounded branch width, then the graphs in $\mathcal{C}$ contain large grids. Now we can try to use these large grids as “coordinate systems” and describe the structure of the graphs relative to the grids. So suppose we have a graph $G \in \mathcal{C}$ with a large grid minor, and let $H \subseteq G$ be the image of a large grid. Let us further assume that $G$ is highly connected; if it is not we first decompose it into highly connected parts and then consider each of them separately. We come back to this decomposition process later. We think of the grid as embedded into the plane and the rest of $G$ being glued onto $H$. It can be proved now that $G \setminus H$ must be glued onto $H$ in a fairly “orderly” way: If there are many pairwise far apart “crossings” in the interior of $G$ then we can find a $K_k$-minor in $G$, which is impossible because $G \in \mathcal{C} \subseteq \mathcal{X}(K_k)$. Here a crossing consists of two pairwise disjoint paths with endpoints $v_1, v_3$ and $v_2, v_4$ respectively, such that $v_1, v_2, v_3, v_4$ occur in this clockwise order on some cycle of the grid. Figure 5.3 shows a grid with two crossings. This leaves us with the following structure: There
is a bounded number of vertices, called apices, that are connected to the grid in an arbitrary fashion. After removing the apices, there still may be many crossings, but they must be grouped together into a bounded number of small regions, called vortices. Apart from the apices and the vortices, the rest of $G$ must fit nicely into the planar structure of the grid, that is, the components of $G \setminus H$ are planar pieces, each of which can be embedded into a “square” of the grid. However, so far we have only talked about the interior of the grid. There may be connections between different parts of the boundary, but they also cannot be too wild, because otherwise we could find a large clique minor again. We can subdivide the boundary into a bounded number of segments and stick some of these together. This gives us a graph that can be embedded into a surface of bounded genus (recall that every surface can be obtained by gluing together edges of a convex polygon in the plane). Thus after removing a bounded number of apices and vortices, $G$ can be embedded into a surface of bounded genus. We say that $G$ has almost bounded genus. We assumed that $G$ is highly connected; if it is not then we can decompose it into pieces with this property. This is Robertson and Seymour’s main structure theorem [68]: For every class $\mathcal{C}$ of graphs with an excluded minor, the graphs in $\mathcal{C}$ can be decomposed into graphs that have almost bounded genus.

Let us now make it precise what we mean by “decomposing” a graph. Intuitively, we want to recursively split the graph along small separators until there no longer are small separators and the graph is highly connected. But if we do this, we lose too much structure in the decomposition process, because two vertices that are far apart on one side of the partition may be close together on the other side and hence in the original graph. Thus “locality”, and similarly “connectivity”, may be destroyed in the decomposition process, and this is something we would like to avoid. We take a very drastic approach: Whenever we separate a graph, on both sides we add edges between all vertices in the separator.

We call a graph $G$ a clique sum of graphs $G_1$ and $G_2$ (and write $G = G_1 \oplus G_2$) if $G_1 \cap G_2$ is a complete graph, $V(G) = V(G_1) \cup V(G_2)$, $E(G) \subseteq E(G_1) \cup E(G_2)$, and $E(G_1) \setminus E(G) \subseteq E(G_2)$, $E(G_2) \setminus E(G) \subseteq E(G_1)$. Thus $G$ is a subgraph of $G_1 \cup G_2$ obtained by possibly deleting some of the edges in $G_1 \cap G_2$. Figure 5.4 illustrates this. Note that we are slightly abusing notation here because there may be several non-isomorphic graphs $G$ such that $G = G_1 \oplus G_2$.

A clique sum decomposition of a graph $G$ is a pair $(T, \gamma)$ consisting of a binary tree $T$ and a mapping $\gamma$ that associates a graph $\gamma(t)$ with every node $t \in V(T)$ such that $\gamma(r) = G$ for the root $r$ of $T$ and $\gamma(t) = \gamma(t_1) \oplus \gamma(t_2)$ for all nodes $t$ with children $t_1, t_2$. Figure 5.5 shows an example of a clique sum decomposition.
Figure 5.5. A clique sum decomposition

of a graph. The decomposition in Figure 5.5 is complete in the sense that all leaves are cliques and hence do not permit any nontrivial clique decompositions. In general, the clique decompositions we are interested in are not necessarily complete.

We call the graphs $\gamma(t)$ in a clique sum decomposition $(T, \gamma)$ the parts of the decomposition and the parts $\gamma(t)$ for the leaves $t$ the atomic parts, or just atoms. $(T, \gamma)$ is a clique sum decomposition over a class $\mathcal{A}$ of graphs if all atoms of $(T, \gamma)$ belong to $\mathcal{A}$. We call a graph decomposable over $\mathcal{A}$ if it has a clique sum decomposition over $\mathcal{A}$ and denote the class of all graphs that are decomposable over $\mathcal{A}$ by $\mathcal{D}(\mathcal{A})$.

Example 5.8. Let $k \geq 1$, and let $\Theta_k$ be the class of all graphs of order at most $k$. If a graph $G$ is decomposable over $\Theta_k$, then $bw(G) \leq \max\{k, 2\}$. Let me suggest it as an exercise for the reader to verify this simple fact.

Conversely, it is not too hard to prove that if a graph has branch width at most $k$, then it is decomposable over $\Theta_{\lceil (3/2)k \rceil}$. Let me remark that a graph has tree width $k$ if and only if it is decomposable over $\Theta_{k+1}$. This follows from the fact that a graph has tree width at most $k$ if and only if it is a subgraph of a chordal graph of clique number $k+1$ (see Corollary 12.3.12 of [25]). The result for branch width then follows by (3.2).

It is an instructive exercise to prove the following simple lemma:

Lemma 5.9. If a class $\mathcal{A}$ of graphs is minor-closed, then the class $\mathcal{D}(\mathcal{A})$ is also minor-closed.

Robertson and Seymour’s structure theorem for classes of graphs with excluded minors can now be stated slightly more precisely as follows: For every class $\mathcal{C}$ of graphs with an excluded minor there is a class $\mathcal{A}$ of graphs that have almost bounded genus such that $\mathcal{C} \subseteq \mathcal{D}(\mathcal{A})$. Of course this still leaves it open what exactly is meant by “almost bounded genus”. We refer the curious reader to the last chapter of Diestel’s book [25] for a more comprehensive introduction to the theory, or to Robertson and Seymour’s original article [68].

We close this section by stating a simplified version of a Robertson and Seymour’s structure theorem that will be sufficient for our purposes. Recall that $\ell_{bw}$ denotes the localization of branch width. Minor-closed classes of locally bounded branch width are particularly well behaved. Eppstein [32, 33] proved that a minor closed class $\mathcal{C}$ has locally bounded branch width if and only if it does not contain all apex graphs (recall the definition of apex graphs from Example 5.6). Demaine and Hajiaghayi [22] proved that if a class of graphs has locally bounded branch width, then there actually is a linear bound on the local
branch width, that is, there is a $\lambda \geq 1$ such that for all $G \in \mathcal{G}$ and for all $r \geq 1$ it holds that $\ell_{bw}(G,r) \leq \lambda \cdot r$. This motivates the definition of the following classes of graphs, for every $\lambda \geq 1$:

$$\mathcal{L}_\lambda = \{ G \mid \ell_{bw}(H,r) \leq \lambda \cdot r \text{ for all } H \preceq G \}. $$

For every $\mu \geq 0$, we define a class of graphs that are “$\mu$-close” to $\mathcal{L}_\lambda$:

$$\mathcal{L}_{\lambda,\mu} = \{ G \mid \exists X \subseteq V(G) : |X| \leq \mu \text{ and } G \setminus X \in \mathcal{L}_\lambda \}.$$  

**Theorem 5.10 (Grohe [45]).** For every class $\mathcal{G}$ with excluded minors, there exist nonnegative integers $\lambda, \mu$ such that $\mathcal{G} \subseteq \mathcal{D}(\mathcal{L}_{\lambda,\mu})$.

To obtain this result from Robertson and Seymour’s structure theorem, one only has to prove that graphs of almost bounded genus are in $\mathcal{L}_{\lambda,\mu}$ for suitable $\lambda, \mu$. This is not very difficult.

### 5.2 Algorithms

Before we get back to model checking problems, let me briefly describe some other algorithmic applications of graph minor theory. Consider the following two parameterized problems:

<table>
<thead>
<tr>
<th><strong>p-DISJOINT-PATHS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> A graph $G$ and vertices $s_1,t_1,\ldots,s_k,t_k \in V(G)$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $k$.</td>
</tr>
<tr>
<td><strong>Problem:</strong> Decide if there are pairwise disjoint paths $P_i$, for $i \in [k]$, from $s_i$ to $t_i$ in $G$.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>p-MINOR</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> Graph $G,H$.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $</td>
</tr>
<tr>
<td><strong>Problem:</strong> Decide if $H \preceq G$.</td>
</tr>
</tbody>
</table>

For neither of the two problems, it is even obvious that they belong to the class XP, that is, can be solved in polynomial time for fixed $k$. For DISJOINT-PATHS, this was a long standing open problem posed by Garey and Johnson [44]. Robertson and Seymour proved that both problems are fixed-parameter tractable:

**Theorem 5.11 (Robertson and Seymour [67]).**

$p$-DISJOINT-PATHS and $p$-MINOR have cubic fpt algorithms.

The reader may wonder why we combine both problems in one theorem. The reason is that they are both special cases of the more general rooted minor problem. A rooted graph is a tuple $(G,v_1,\ldots,v_k)$, where $G$ is a graph and $v_1,\ldots,v_k \in V(G)$, and a rooted graph $(H,w_1,\ldots,w_k)$ is a rooted minor of a rooted graph $(G,v_1,\ldots,v_k)$ if there is a minor map $\mu$ from $H$ into $G$ such that $v_i \in V(\mu(w_i))$ for all $i \in [k]$. The parameterized problem $p$-ROOTED-MINOR is defined as $p$-MINOR, but for rooted graphs. I leave it to the reader to reduce $p$-DISJOINT-PATHS to $p$-ROOTED-MINOR. Robertson and Seymour proved that $p$-ROOTED-MINOR has a cubic fpt algorithm.

To get an idea of the proof it is easiest to look at the disjoint paths problem. Suppose we are given a graph $G$ and $s_1,t_1,\ldots,s_k,t_k \in V(G)$. Let us further assume, to simplify the presentation, that $G$ is 2k-connected. If $K_{3k} \preceq G$, then we know that there are disjoint paths from the $s_i$s to the $t_i$s: As the graph is 2k-connected, by Menger’s theorem we can find disjoint paths from $s_1,t_1,\ldots,s_k,t_k$ to an image of $K_{3k}$. Then in the image of $K_{3k}$, we can connect the pieces in the right way because all connections are there. This is not entirely trivial, because we only have an image of $K_{3k}$ and not a subgraph, but it can be done. So now we can assume that $K_{3k} \not\preceq G$; and we can apply the structure theory for graphs with excluded $K_{3k}$. If the branch width of $G$ is bounded, we can solve the disjoint paths problem easily, for example, by applying Courcelle’s theorem. If the branch width is large, then by the Excluded Grid Theorem, we can
find a large grid in \( G \). By the arguments described above, we can now find a small set of vertices such that after removing these vertices, the whole graph \( G \) fits nicely into the planar structure of the grid. Passing to a smaller grid if necessary, we may assume that all the \( s_i \) and \( t_i \) are outside the grid. Now it can be proved that if there are disjoint paths from \( s_i \) to \( t_i \) for all \( i \in [k] \), then there are such paths that avoid the middle vertex of the grid (say, the grid has odd order). Intuitively, it is plausible that if we have a very large grid and \( k \) disjoint paths traversing the grid, then we can always re-route them to avoid the middle vertex. Proving this formally turns out to be the most difficult part of the whole proof [62, 63]. It builds on the full structure theory described in the previous section. However, once this is done, we know that we can delete \( G \) for all \( i \in \mathbb{Z} \), testing membership in the minor-closed class \( C \). As \( C \) is closed under taking subgraphs, we may actually assume that this simplicial decomposition is complete.

Remarkably, Corollary 5.12 just claims the existence of an algorithm, without actually giving us the algorithm. For example, by Example 5.7 it implies the existence of a cubic time algorithm for deciding whether a graph is knot free. But we do not know such an algorithm! The reason is that we do not know a finite family of excluded minors defining the class of knot free graphs. Corollary 5.12 is constructive in the sense that if we are given a finite family of excluded minors that defines the class \( C \), then we can construct a cubic time algorithm deciding \( C \). However, for many minor-closed classes we do not know such a finite family.

In recent years, there has been a substantial body of work on algorithms for graph problems restricted to graph classes with excluded minors or even generalisations of such classes [1, 21, 23, 24, 45, 49]. The algorithmic meta theorems presented in the following section should be seen in this context as an attempt to get a more global view on the potentials of algorithmic graph minor theory.

We close this section with a lemma that we will need in the next section. Note that, in particular, it implies an algorithmic version of Theorem 5.10: For every class \( C \) with excluded minors there is a polynomial time algorithm that, given a graph in \( C \), computes a clique sum decomposition of \( G \) over \( \mathcal{L}(\lambda, \mu) \).

**Lemma 5.13.** For every minor-closed class \( \mathcal{A} \) of graphs there is an algorithm that, given a graph \( G \in \mathcal{P}(\mathcal{A}) \), computes a clique sum decomposition of \( G \) over \( \mathcal{A} \) in time \( O(n^3) \).

**Proof sketch.** Recall that if we write \( G = G_1 \oplus G_2 \), this implies that \( V(G_1 \cap G_2) \) induces a clique in both \( G_1 \) and \( G_2 \), but not necessarily in \( G \). If it also induces a clique in \( G \), and hence \( G = G_1 \cup G_2 \), we call the clique sum simplicial. We call a clique sum decomposition \((T, \gamma)\) a simplicial decomposition if the clique sums at all nodes of \( T \) are simplicial. We call a simplicial decomposition complete if its atoms can not be decomposed any further. Simplicial decompositions are much easier to handle than clique sum decompositions. Tarjan and Yannakakis [72, 73] showed that a complete simplicial decomposition of a given graph can be found in linear time.

Observe that if a graph \( G \) has a clique sum decomposition over \( \mathcal{A} \), then some supergraph \( G' \supseteq G \) with the same vertex set has a simplicial decomposition over \( \mathcal{A} \). As \( \mathcal{A} \) is closed under taking subgraphs, we may actually assume that this simplicial decomposition is complete.

To compute a clique sum decomposition of a graph \( G \) over \( \mathcal{A} \), we proceed as follows: We add as many edges as possible to \( G \) so that the resulting graph \( G' \) is still in the class \( \mathcal{P}(\mathcal{A}) \). We can do this in time \( O(n^3) \), testing membership in the minor-closed class \( \mathcal{P}(\mathcal{A}) \) in cubic time for every potential edge. Then we use the Tarjan-Yannakakis algorithm to find a complete simplicial decomposition of the graph \( G' \). This also gives us a clique sum decomposition of \( G \). \( \square \)

### 6 First-order logic on graph classes with excluded minors

Let \( C \) be a class of graphs with excluded minors. Our goal is to design an fpt algorithm for the first-order model checking problem on \( C \). Recall that by Theorem 5.10, the graphs in \( C \) are decomposable into
graphs that “almost” have locally bounded branch width, where almost means “after removing a bounded number of vertices”. We know how to deal with graphs of locally bounded branch width, and it is not hard to extend this to graphs of almost locally bounded branch width. Moreover, we know how to deal with tree structured graphs. By combining these things, so it seems, it should not be too hard to obtain the desired result. This is true, but there are technical difficulties to overcome.

We say that a tuple \( v \) of vertices of a graph \( G \) induces a clique in \( G \) if \( G[\{v\}] \) is a complete graph. We write \( G = G' \oplus_H H \) to denote that \( G \) is a clique sum of graphs \( G' \) and \( H \) with \( V(G') \cap V(H) = \{v\} \). For tuples \( v_1, \ldots, v_m \) of vertices in \( G' \) and graphs \( H_1, \ldots, H_m \), we may write \( G' \oplus_{v_1} H_1 \oplus_{v_2} \cdots \oplus_{v_m} H_m \); the order of the summation of the \( H_s \) does not matter. In the following, types are always first-order types, and we write \( tp \) instead of \( tp^{\Theta_0} \). Recall Lemma 2.3.

**Lemma 6.1.** The following problem is fixed parameter tractable:

<table>
<thead>
<tr>
<th>Instance: A labelled graph ( G' ) of branch width ( k ), tuples ( \bar{v}_i \in V(G')^k ) for ( i \in [0, m] ) that induce cliques in ( G' ), and ( q )-types ( \Theta_1, \ldots, \Theta_m ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter: ( q ).</td>
</tr>
<tr>
<td>Problem: Compute the type ( tp_q(G, \bar{v}<em>0) ) for some graph ( G = G' \oplus</em>{v_1} H_1 \oplus_{v_2} \cdots \oplus_{v_m} H_m ), where the ( H_i ) are graphs with ( tp_q(H_i, \bar{v}_i) = \Theta_i ) for all ( i \in [m] ).</td>
</tr>
</tbody>
</table>

Note that by Lemma 2.3, if \( tp_q(G, \bar{v}_0) = \Theta_0 \) for some graph \( G = G' \oplus_{v_1} H_1 \oplus_{v_2} \cdots \oplus_{v_m} H_m \), where the \( H_i \) are graphs with \( tp_q(H_i, \bar{v}_i) = \Theta_i \) for all such graphs \( G \).

**Proof sketch of Lemma 6.1.** Recall the second proof we sketched for Courcelle’s theorem. Observe that this proof would also work if the function \( \beta \) in a branch decomposition was not injective, that is, if some of the edges of \( G \) appeared at several leaves of \( T \). Let us call a pair \( (T, \beta) \), where \( T \) is a binary tree and \( \beta \) is a surjective, but not necessarily injective function from the leaves of \( T \) onto \( E(G) \), a pseudo branch decomposition. The extension \( \hat{\beta} \) of \( \beta \) to all nodes of the tree can be defined as for branch decompositions. If for some node \( t_1 \) with sibling \( t_2 \) an edge appears both in \( \hat{\beta}(t_1) \) and \( \hat{\beta}(t_2) \), then both endpoints of this edge are part of the boundary at \( t_1 \) and \( t_2 \) and hence contribute to the width of the decomposition.

Now let us prove the statement of the lemma. As all the \( \bar{v}_i \) induce cliques in \( G' \) and \( bw(K_n) = \lceil (2/3) \cdot n \rceil \), we have \( k_i \leq (3/2) \cdot k \) for all \( i \). It it easy to transform a given branch decomposition \( (T, \beta) \) of \( G' \) of width \( k \) into a pseudo branch decomposition \( (T', \beta') \) of \( G' \) of width at most \( (3/2) \cdot k \) such that for all \( i \in [m] \) there exists a node \( s_i \in V(T') \) such that \( \beta'(s_i) = E(G'[\{\bar{v}_i\}]) = \left\lceil \frac{\|v_i\|}{2} \right\rceil \). For every node \( t \in V(T') \), we let \( A_t = \{v_0\} \cap \bigcup \beta'(t) \) be the set of all elements of the tuple \( \bar{v}_0 \) that are incident with an edge in \( \beta'(t) \), and we let \( \bar{b}_t \) be a tuple with \( \{\bar{b}_t\} = \partial \beta'(t) \cup A_t \). Note that \( \{\bar{b}_r\} = \{v_0\} \) for the root \( r \) of \( T' \). Starting at the leaves, we now compute the types \( tp_q(G[\beta'(t)]_{t}, \bar{b}_t) \) for all nodes \( t \), using Lemma 2.3 at every node, except that at the nodes \( s_i \) we use the types \( \Theta_i \) instead of \( tp_q(G[\beta'(s_i)]_{s_i}, \bar{b}_{s_i}) \). \( \square \)

**Lemma 6.2.** For all \( \lambda, \mu \), the following problem is fixed-parameter tractable:

<table>
<thead>
<tr>
<th>Instance: A labelled graph ( G' \in L_{\lambda, \mu} ), tuples ( \bar{v}_i \in V(G')^k ) for ( i \in [0, m] ) that induce cliques in ( G' ), and ( q )-types ( \Theta_1, \ldots, \Theta_m ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter: ( q ).</td>
</tr>
<tr>
<td>Problem: Compute ( tp_q(G, \bar{v}<em>0) ) for some graph ( G = G' \oplus</em>{v_1} H_1 \oplus_{v_2} \cdots \oplus_{v_m} H_m ), where the ( H_i ) are graphs with ( tp_q(H_i, \bar{v}_i) = \Theta_i ) for all ( i \in [m] ).</td>
</tr>
</tbody>
</table>

**Proof sketch.** We prove the statement by induction on \( \mu \). For \( \mu = 0 \), that is, graphs in \( L_{\lambda, 0} \), it can be proved similarly as Theorem 4.7 (using Lemma 6.1 locally).

So let \( \mu > 0 \). Suppose we are given an instance of the problem. We observe that the graph \( G' \) contains a vertex \( w \) such that \( G' \setminus \{w\} \in L_{\lambda, \mu-1} \). As \( L_{\lambda, \mu-1} \) is minor-closed and hence decidable in cubic time by Corollary 5.12, we can find such a vertex in time \( O(n^3) \). We define a new labelled graph \( G'' \) by deleting the vertex \( w \) and labelling all vertices adjacent to \( w \) in \( G' \) with a new label \( P \). We then translate every formula \( \psi \) of quantifier rank at most \( q \) into a formula \( \psi^* \) such that \( G \models \psi(v_0) \iff G'' \models \psi^*(v_0) \). As \( G'' \in L_{\lambda, \mu-1} \),
we can apply the induction hypothesis to check if $G^* = \emptyset^*(\bar{v}_0)$, and this way we can compute the type of $\bar{v}_0$ in $G$.

\begin{theorem}[Flum and Grohe [37]]
For every class $\mathcal{C}$ of graphs with an excluded minor, the problem \( p\text{-MC}(\mathcal{FO}, \mathcal{C}) \) is fixed-parameter tractable.
\end{theorem}

\begin{proof}[Proof sketch]
Let $G \in \mathcal{C}$ and $\phi \in \mathcal{FO}$, say, of quantifier rank $q$.

Let $\lambda, \mu \geq 0$ such that $\mathcal{C} \subseteq \mathcal{B}(\mathcal{L}_{\lambda, \mu})$. Using Lemma 5.13, we compute a clique sum decomposition $(T, \gamma)$ of $G$ over $\mathcal{L}_{\lambda, \mu}$.

Now the obvious idea is to compute the $q$-types of the “boundary tuples” for the parts $\gamma(t)$ in the decomposition in a bottom-up fashion, similarly to the second proof of Courcelle’s Theorem. Unfortunately, this simple idea does not work, because a clique sum decomposition is not as well-behaved as a branch decomposition, and the boundaries of the parts may have unbounded size. It may even happen that an atom of the decomposition (corresponding to a leaf of the tree) intersects all other atoms. Figure 6.1 illustrates this.

Observe that a graph in $\mathcal{L}_{\lambda, \mu}$ cannot contain a clique with more than $k = \lceil (3/2) \cdot \lambda + \mu \rceil$ vertices. Hence for all nodes $t$ of $T$ with children $t_1, t_2$, we must have $V(\gamma(t_1) \cap \gamma(t_2)) \leq k$, because $V(\gamma(t_1) \cap \gamma(t_2))$ is a clique in the $\gamma(t_i)$, and this clique will appear in some atom of the decomposition. Let us fix some order of the vertices of $G$. For every inner node $t$ with children $t_1, t_2$, we let $\bar{c}_t$ be the ordered tuple that contains the elements of $V(\gamma(t_1) \cap \gamma(t_2))$.

Our algorithm proceeds recursively, that is, “top-down”, instead of “bottom up” as the algorithm in the proof of Courcelle’s Theorem, to compute the types of the tuples $\bar{c}_t$. Let us start at the root $r$ of $T$. Our goal is to compute the $q$-type of the empty tuple in $G$. Suppose that the clique sum at $r$ is $G = G_1 \oplus G_2$. We now want to compute the $q$-type of the tuple $\bar{c}_r$ in both $G_1$ and $G_2$: from that we easily get the $q$-type of the empty tuple in $G$ using Lemma 2.3. So let us continue by computing the $q$-type of $\bar{c}_t$ in $G_1$. Suppose the children of $t_1$ are $t_{11}$ and $t_{12}$. Let $\bar{c}_{t_1} = \bar{c}_{t_1}$. Now we have a problem: To determine the $q$-type of $\bar{c}_t$ in $G_1$, it does not suffice to compute the $q$-types of $\bar{c}_t$ in $G_{11}$ and $G_{12}$, because $\bar{c}_t$ and $\bar{c}_r$ may be disjoint tuples. It seems that we have to compute the $q$-type of the longer tuple $\bar{c}_t \bar{c}_r$ in both graphs. But clearly we cannot afford the tuples to get longer at every recursion level. Now recall that $\{\bar{c}_r\}$ is a clique in $G_1$. Hence it is either contained in $\{\bar{c}_{t_1}\} = V(G_{11}) \cap V(G_{12})$, in which case we have no problem anyway, or it is contained in precisely one of the two graphs $G_{11}, G_{12}$. Suppose $\bar{c}_r$ is contained in $G_{12}$. Then we first compute the $q$-type $\Theta$ of the tuple $\bar{c}_t \bar{c}_r$ in $G_{12}$. Now we have to compute the type of $\bar{c}_r$ in the graph $G_1 = G_{12} \oplus G_{11}$. That is, we are in the situation where we have to compute the type of a tuple $\bar{v}$ of vertices of a graph $G'$ in a graph $G' \oplus H$ for some (and hence all) graph(s) $H$ with $\text{tp}_q(H, \bar{v}) = \Theta$. Furthermore, we know that $\bar{v}, \bar{v}'$ induce cliques in $G'$. The general problem we have to solve recursively at all nodes of the decomposition tree is the following:

Compute the $q$-type of a tuple $\bar{v}_0$ of vertices of a graph $G'$ in a graph $G' \oplus H_1 \oplus \ldots \oplus H_m$ for some (and hence all) graph(s) $H_i$ with $\text{tp}_q(H_i, \bar{v}_i) = \Theta_i$. Here all the tuples $\bar{v}_i$ have length at most $k$, and they induce cliques in $G'$.

\begin{figure}[ht]
\centering
\begin{tikzpicture}
    \node (G) at (0,0) {$G'$};
    \node (H1) at (-2,2) {$H_1$};
    \node (H2) at (-2,-2) {$H_2$};
    \node (H3) at (2,2) {$H_3$};
    \node (H4) at (2,-2) {$H_4$};
    \node (I) at (0,4) {$I$};
    \node (G1) at (-3,0) {$G' \oplus H_1 \oplus H_2 \oplus I$};
    \node (G2) at (-3,-4) {$G' \oplus H_1 \oplus H_2$};
    \node (G3) at (3,0) {$G' \oplus H_1 \oplus I$};
    \node (G4) at (3,-4) {$G'$};
    \node (H5) at (5,0) {$H_5$};
    \node (H6) at (5,-4) {$H_6$};
    \node (H7) at (7,0) {$H_7$};
    \node (H8) at (7,-4) {$H_8$};

    \draw (G) -- (H1);
    \draw (G) -- (H2);
    \draw (G) -- (H3);
    \draw (G) -- (H4);
    \draw (G) -- (I);
    \draw (G1) -- (H5);
    \draw (G1) -- (H6);
    \draw (G1) -- (H7);
    \draw (G1) -- (H8);
    \draw (G2) -- (H5);
    \draw (G2) -- (H6);
    \draw (G2) -- (H7);
    \draw (G2) -- (H8);
    \draw (G3) -- (H5);
    \draw (G3) -- (H6);
    \draw (G3) -- (H7);
    \draw (G3) -- (H8);
    \draw (G4) -- (H5);
    \draw (G4) -- (H6);
    \draw (G4) -- (H7);
    \draw (G4) -- (H8);

\end{tikzpicture}
\caption{The left hand side shows a graph and the right hand side a clique sum decomposition of this graph where the atom $G'$ intersects four other atoms and the atom $H_5$ intersects two other atoms.}
\end{figure}
At the leaves we can use Lemma 6.2 to do this. At the inner nodes, we proceed as described for the node \( t_1 \) above.

The proof of the theorem actually shows that for all classes \( \mathcal{C} \) with excluded minors, \( p\text{-MC}(\text{FO}, \mathcal{C}) \) has an fpt algorithm with exponent at most 5. Hence, the exponent is independent of the class \( \mathcal{C} \). Thus we have “almost” proved that there is an fpt algorithm for the model checking problem parameterized both by formula size and the size of the excluded minor. With considerable additional effort, we can get rid of the “almost” in this statement. Let me explain where the difficulties are and, in very general terms, how they are resolved.

Let us first make the statement precise. We define a new graph invariant \textit{excluded minor order (emo)} by letting

\[
\text{emo}(G) = \min\{|H| \mid H \not\preceq G\}
\]

for every graph \( G \). Note that \( \text{emo}(G) = \min\{n \mid K_n \not\preceq G\} \) and that a class \( \mathcal{C} \) excludes a minor if and only if it has bounded excluded minor order. Our goal is to prove that the following problem is fixed-parameter tractable:

\[
\begin{align*}
\text{Instance:} & \quad \text{A graph } G \text{ and a sentence } \varphi \in \text{FO}. \\
\text{Parameter:} & \quad |\varphi| + \text{emo}(G). \\
\text{Problem:} & \quad \text{Decide if } G \models \varphi.
\end{align*}
\]

We have already proved that for every \( k \) there is an fpt algorithm \( A_k \) with exponent 5 for the first-order model checking problem on the class of all graphs of excluded minor order at most \( k \). The problem is that the family \( A_k \) of algorithms is nonuniform, that is, we have a different algorithm for every \( k \). To prove that \( p\text{-MC}(\text{FO}, \text{emo}) \) is fixed-parameter tractable, we need a uniform family \( A \), or equivalently, a single algorithm \( A \) that takes \( k \) as an additional input. The family of algorithms we construct in the proof is nonuniform because we use Corollary 5.12 to get decision algorithms for the minor-closed classes \( \mathcal{L}_{\lambda, \mu} \) (in the proof of Lemma 6.2) and \( \mathcal{D}(\mathcal{L}_{\lambda, \mu}) \) (in the proof of Lemma 5.13) for parameters \( \lambda, \mu \) that depend on the excluded minor order of the input graph. If we could compute finite families of excluded minors characterising the classes \( \mathcal{L}_{\lambda, \mu} \) and \( \mathcal{D}(\mathcal{L}_{\lambda, \mu}) \) from the parameters \( \lambda, \mu \), then we would be fine, but we currently do not know how to do this. Fortunately, there is an alternative approach that avoids Corollary 5.12 entirely. The application of Corollary 5.12 in the proof of Lemma 5.13 yielded an algorithm for computing a clique sum decomposition of a graph over \( \mathcal{D}(\mathcal{L}_{\lambda, \mu}) \). While we do not know how to compute such a decomposition uniformly in \( \lambda \) and \( \mu \), in [18] we found a way to compute, uniformly in \( \lambda, \mu \), a decomposition that is a sufficiently good approximation of the desired clique sum decomposition. The algorithm recursively splits the input graph along small separators that are sufficiently “balanced”. The application of Corollary 5.12 in the proof of Lemma 6.2 was needed to find a set of at most \( \mu \) vertices in a graph in \( \mathcal{L}_{\lambda, \mu} \) whose removal left a graph in \( \mathcal{L}_\lambda \). In [18], we found an fpt algorithm that, given a graph \( G \in \mathcal{L}_{\lambda, \mu} \), computes a set \( W \subseteq V(G) \) of at most \( \mu \) vertices such that \( G \setminus W \in \mathcal{L}_{\lambda'} \) for some \( \lambda' \) that is effectively bounded in terms of \( \lambda \). This is good enough for our purposes. Putting everything together, we obtain the following result:

**Theorem 6.4 (Dawar, Grohe, and Kreutzer [18]).** \( p\text{-MC}(\text{FO}, \text{emo}) \) is fixed-parameter tractable.

We say that a class \textit{locally excludes a minor} if it has locally bounded excluded minor order. Then combining Theorems 6.4 and 4.7, we get:

**Corollary 6.5 ([18]).** For every class \( \mathcal{C} \) locally excluding a minor, the problem \( p\text{-MC}(\text{FO}, \mathcal{C}) \) is fixed-parameter tractable.

### 7 Other logics and other problems

In this section, we briefly discuss some extensions of the main results mentioned in this survey to more powerful logics, and also to variants of the basic model checking problem.
7.1 Other logics

It is really not much that is known about algorithmic meta theorems for logics other than first-order and monadic second-order logic. Courcelle’s Theorem and its variant for graphs of bounded rank width can be extended to the extension of monadic second order logic by modulo counting quantifiers [10, 12] (also see [53]), and clearly not to full binary second order logic.

As for the results for first-order logic, let us consider potential extensions of the model-checking results to monadic transitive closure logic and monadic least fixed-point logic. Both transitive closure logic and least fixed-point logic have been extensively studied in finite model theory [29, 51]. Their monadic fragments are strictly contained in monadic second-order logic, and they strictly contain first-order logic. (When we say that a logic contains another logic, we mean semantic containment, that is, L₁ contains L₂ if every formula of L₂ is logically equivalent to a formula of L₁. We say that L₁ strictly contains L₂ if L₁ contains L₂, but L₂ does not contain L₁.) Monadic transitive closure logic and monadic least fixed-point logic seem to mark the boundary of the range of logics to which the tractability results for first-order model checking can be extended.

Monadic transitive closure logic TC₁ is the extension of first-order logic by formulas of the form [TC₁,x,y]ϕ(x,y), where ϕ is a formula with free variables among {x,y}. The free variables of the formula [TC₁,x,y]ϕ(x,y) are x and y. It is allowed to nest TC-operators arbitrarily and interleave them with first-order quantifiers and connectives. However, we do not allow any other free variables than x and y in the formula ϕ in [TC₁,x,y]ϕ(x,y). The semantics is defined as follows: If G is a (labelled) graph and v,w ∈ V(G), then G |= [TC₁,x,y]ϕ(v,w) if and only if there is an m ≥ 1 and vertices v₁,...,vₘ ∈ V(G) such that v = v₁, w = vₘ, and G |= ϕ(vᵢ,vᵢ₊₁) for all i ∈ [m-1].

Example 7.1. The following TC₁-sentence states that a graph is connected:

∀x∀y[TC₁,x,y]E(x,y)](x,y).

It is known that there is no sentence of first-order logic defining connectivity (see, e.g., [29, 30, 51]).

Example 7.2. The following TC₁-sentence states that a graph has no cyclic walk of odd length and hence is bipartite

¬∃x∃y[(TC₁,x,y)E(x,z)∧E(z,y)](x,y)∧E(y,x).

Again, it known that there is no sentence of first-order logic defining bipartiteness.

The logic TC₁ trivially contains FO, and it is strictly contained in MSO. As opposed to MSO, its data complexity is still in polynomial time (actually, in nondeterministic logarithmic space).

Theorem 7.3. Let C be a class of graphs that contains all planar graphs of degree at most 3. Then p-MC(TC₁,C) is hard for the parameterized complexity class AW[∗].

Proof sketch. We reduce the model checking problem for first-order logic on arbitrary graphs, which is known to be AW[∗]-complete (by Theorem 2.12), to p-MC(TC₁,C). Let G be a graph and ϕ a first-order sentence.

We start with constructing a drawing of G in the plane, which of course may involve edge crossings. We can find a drawing with at most polynomially many (in the number of vertices of G) crossings such that in each point of the plane at most 2 edges cross. We introduce five new labels P₁,P₂,Q₁,Q₂,R. We define a new labelled graph G₁ by labelling each vertex of the original graph G with P₁ and replacing each edge crossing in the drawing of G by a little gadget, as shown in Figure 7.1. Observe that the edge relation of the graph G can be defined in G₁ by a TC₁-formula (but not by an FO-formula, because an edge may cross many other edges). G₁ is planar, but may have degree greater than 3. We define a graph G₂ by replacing every vertex v of G₁ of degree d by a binary tree with exactly d leaves. With each leaf we associate one vertex w adjacent to v in G₁. We connect the leaf of the v-tree associated with v with the leaf of the w-tree associated with v. Then we identify v with the root of its tree, label it P₁, and label all other vertices of the tree P₂. Then the edge relation of G₂ is also definable in G₂ by a TC₁-formula. We can use this formula to translate the formula ϕ into a TC₁-formula ϕ₂ such that

G |= ϕ ⇐⇒ G₂ |= ϕ₂.
$G_2$ is a planar graph of degree at most 3, and it clearly can be computed from $G$ in polynomial time. This gives us the desired reduction.

Monadic least-fixed-point logic $\text{LFP}^1$ (see, e.g., [47, 70]) is the extension of first-order logic by formulas of the form $[\text{LFP}_{x,X} \varphi](x)$, where $\varphi$ is a first-order formula such that $X$ only occurs positively in $\varphi$ and $\varphi$ has no free individual variables other than $x$. (It may have free set variables other than $X$.) The free variables of $[\text{LFP}_{x,X} \varphi](x)$ are $x$ and all free set variables of $\varphi$ except $X$. To define the semantics, let $\varphi = \varphi(x,X,Y_1,\ldots,Y_m)$. Let $G$ be a labelled graph and $W_1,\ldots,W_m \subseteq V(G)$, $v \in V(G)$. Then $G \models [\text{LFP}_{x,X} \varphi(x,X,W_1,\ldots,W_m)](v)$ if and only if $v$ is in the least fixed point of the monotone operator $U \mapsto \{u \mid G \models \varphi(u,U,W_1,\ldots,W_m)\}$ on $V(G)$. We call a formula in $\text{LFP}^1$ restricted if for every subformula of the form $[\text{LFP}_{x,X} \varphi](x)$, the formula $\varphi$ has no free set variables other than $X$. By $\text{LFP}^1_r$ we denote the fragment of $\text{LFP}^1$ consisting of all restricted formulas.

The reason for requiring that a formula $\varphi$ in the scope of a fixed-point operator $[\text{LFP}_{x,X} \varphi](x)$ contains no free individual variables other than $x$ is that otherwise even the restricted fragment of the logic would contain $\text{TC}^1$. It can be shown that $\text{LFP}^1_r$ (as defined here) does not contain $\text{TC}^1$ and that, conversely, $\text{TC}^1$ does not contain $\text{LFP}^1_r$. I was unable to come up with convincing examples of properties of plain graphs that are definable in $\text{LFP}^1_r$ or $\text{LFP}^1$, but not in first-order logic. However, this changes when we admit more general structures. For example, on Kripke structures, that is, labelled directed graphs with one distinguished element, $\text{LFP}^1_r$ contains the modal $\mu$-calculus. Here is another example:

Example 7.4. We can describe Boolean circuits as labelled directed acyclic graphs, and assignments to the input gates by an additional label. It is easy to see that there is an $\text{LFP}^1_r$-formula stating that an assignment satisfies a circuit. This is not definable in first-order logic.

As we mentioned earlier, almost all results presented in this survey extend to arbitrary structures. In this context, the following tractability result is more interesting than it may seem in a purely graph theoretical context.

Theorem 7.5. Let $\mathcal{C}$ be a class of graphs such that $\text{p-MC}(\text{FO}, \mathcal{C}_{lb})$ is fixed-parameter tractable. Then $\text{p-MC}(\text{LFP}^1_r, \mathcal{C}_{lb})$ is fixed-parameter tractable.

Proof sketch. To evaluate a formula of the form $[\text{LFP}_{x,X} \varphi](x)$, where $\varphi = \varphi(x,X)$ is first-order, in a graph $G$, we proceed as follows: We introduce a new label $P$. Initially, we set $P(G) = \emptyset$. Then we repeatedly compute the set of all $v \in V(G)$ such that $G \models \varphi(v,P(G))$ using an fpt algorithm for $\text{p-MC}(\text{FO}, \mathcal{C}_{lb})$ and set $P(G)$ to be the set of all these vertices. After at most $n = |G|$ steps, the computation reaches a fixed point, which consists precisely of all $v$ such that $G \models [\text{LFP}_{x,X} \varphi](v)$. Using this algorithm as a subroutine, we can easily model-check arbitrary sentences in $\text{LFP}^1_r$.

Lindell [52] proved that for the classes $\mathcal{D}_k$ of graphs of degree at most $k$, the problem $\text{p-MC}(\text{LFP}^1_r, \mathcal{D}_k)$ even has a linear time fpt algorithm.
7.2 Generalised model checking problems

For a formula \( \varphi(x_1, \ldots, x_k) \) and a graph \( G \), by \( \varphi(G) \) we denote the set of all tuples \( (v_1, \ldots, v_k) \in V(G)^k \) such that \( G \models \varphi(v_1, \ldots, v_k) \). For every logic \( L \) and class \( \mathcal{C} \) of graphs, we may consider the following variants of the model checking problem \( p\text{-MC}(L, \mathcal{C}) \): The input always consists of a graph \( G \in \mathcal{C} \) and a formula \( \varphi \in L \), possibly with free variables. The parameter is \( |\varphi| \). The decision problem simply asks if \( \varphi(G) \) is nonempty. For logics closed under existential quantification, this problem is equivalent to the model checking problem \( p\text{-MC}(L, \mathcal{C}) \). Therefore, we will not consider it here anymore. The construction problem asks for a solution \( \bar{v} \in \varphi(G) \) if there exists one. The evaluation (or listing) problem asks for all solutions, that is, for the whole set \( \varphi(G) \). Finally, the counting (or enumeration) problem asks for the number \( |\varphi(G)| \) of solutions. All these problems have natural applications.

The results on monadic second-order model checking on graphs of bounded branch width and bounded rank width (Theorems 3.3 and 3.17) can be extended to the corresponding construction and counting problems \([3, 15, 36, 39]\). For the evaluation problem, the situation is a bit more complicated because the size of the answer \( \varphi(G) \) may be much larger than the size of the input (\( n^k \) for a graph of order \( n \) and a formula with \( k \) free variables), hence we cannot expect an algorithm that is fixed-parameter tractable. However, it has been proved that there is a linear time fpt algorithm for this problem if the running time is measured in terms of the input size plus the output size \([16, 36]\). Recently, it has been shown that there even is such an algorithm that does a linear (in terms of the input size) pre-computation and then produces solutions with delay bounded in terms of the parameter \([4, 13]\).

Frick \([39, 40]\) proved that the construction problem and counting problem for many classes of graphs of locally bounded branch width, including planar graphs and graphs of bounded degree, has a linear fpt algorithm. This is a nontrivial extension of the model checking results. Even for a simple first-order definable counting problem like the parameterized independent set counting problem ("Count the number of independent sets of size \( k \) in a graph."), say, on a class of graphs of bounded degree, it is not obvious how to solve it by an fpt algorithm. For the evaluation problem, again there are linear time fpt algorithms if the running time is measured in terms of the input size plus the output size \([39]\). For classes of graphs of bounded degree, Durand and Grandjean \([28]\) proved that there is an fpt algorithm for the first-order evaluation problem that does a linear pre-computation and then produces solutions with delay bounded in terms of the parameter.

Finally, let us take a brief look at optimisation problems, which play a central role in complexity theory, but have not been studied very systematically in the context of meta theorems. Consider a first-order definable counting problem that does a linear pre-computation and then produces solutions with delay bounded in terms of the parameter. This results on monadic second-order model checking on graphs of bounded branch width and bounded rank width (Theorems 3.3 and 3.17) can be extended to the corresponding construction and counting problems \([3, 15, 36, 39]\). For the evaluation problem, the situation is a bit more complicated because the size of the answer \( \varphi(G) \) may be much larger than the size of the input (\( n^k \) for a graph of order \( n \) and a formula with \( k \) free variables), hence we cannot expect an algorithm that is fixed-parameter tractable. However, it has been proved that there is a linear time fpt algorithm for this problem if the running time is measured in terms of the input size plus the output size \([16, 36]\). Recently, it has been shown that there even is such an algorithm that does a linear (in terms of the input size) pre-computation and then produces solutions with delay bounded in terms of the parameter.

8 Concluding remarks and open questions

Figure 8.1 gives an overview of the classes of graphs we have studied in this survey. Let me conclude by mentioning four directions for further research where I believe the most interesting open questions are:

8.1 Further tractable classes

Many of the classes of graphs considered in this survey, including all classes excluding a minor, have bounded average degree. It may be tempting to conjecture that first-order model checking is tractable on all classes of graphs of bounded average degree, but it is easy to see that this is not the case. As Stephan Kreutzer observed, it is not even the case for classes of bounded maximum average degree, where the maximum average degree of a graph \( G \) is the maximum of the average degrees of all subgraphs of \( G \). To see this, just observe that model-checking on a graph \( G \) can be reduced to model-checking on its incidence
Figure 8.1. Classes of graphs with a tractable first-order model checking problems. Double-lined ellipses contain families of classes. Classes below the dashed line have a tractable monadic second-order model checking problem.

graph (i.e., the graph obtained from $G$ by subdividing each edge once), and that every incidence graph has maximum average degree at most 4.

Nešetřil and Ossona de Mendez [55] introduced a property of graph classes that may be viewed as a refinement of maximum average degree and that avoids such problems. Let $G$ be a graph. The radius of a connected vertex set $X \subseteq V(G)$ is the minimum $r$ such that $X$ is contained in $N_r(v)$ for some $v \in V(G)$. The radius of a minor mapping $\mu$ from a graph $H$ to $G$ is the minimum of the radii of the sets $\mu(v)$, for $v \in V(H)$. We write $H \leq_r G$ if there is a minor mapping of radius at most $r$ from $H$ to $G$. Note that $H \leq_0 G$ if and only if $H$ is a subgraph of $G$. The greatest reduced average density (grad) of rank $r$ of $G$ is the number

$$\nabla_r(G) = \max \left\{ \frac{|E(H)|}{|V(H)|} \mid H \leq_r G \right\}.$$

Note that $\nabla_0(G)$ is half the maximum average degree of $G$. Now a class $\mathcal{C}$ of graphs has bounded expansion if there is some function $f$ such that $\nabla_r(G) \leq f(r)$ for all $G \in \mathcal{C}$ and $r \geq 0$. Nešetřil and Ossona de Mendez observed that every class of graphs excluding a minor has bounded expansion. It can be shown that there are classes of bounded expansion that do not exclude a minor, not even locally. Conversely, there are classes of bounded local tree width and hence classes locally excluding a minor that do not have bounded expansion. This follows from Example 4.6 and the fact that classes of bounded expansion have bounded average degree. I refer the reader to [54, 55, 56] for an introduction to classes of bounded expansion and an overview of their nice algorithmic properties.

Conjecture 8.1. $\text{p-MC(FO, } \mathcal{C})$ is fixed-parameter tractable for every class $\mathcal{C}$ of graphs of bounded expansion.

8.2 Necessary conditions for tractability

The main results presented in this survey may be viewed as giving sufficient conditions for classes of graphs to have tractable first-order or monadic second-order model checking problems. What are necessary conditions for tractability, and which classes have hard model checking problems? Note that it is not easy to come up with structural conditions for hardness, because we can “cheat” and, for example, pad graphs that have a structure presumably making model checking difficult with a large number of isolated vertices.
This makes the model checking problem “easier” just because it gives us more time to solve it. Thus we probably want to impose closure conditions on the classes of graphs we consider, such as being closed under taking subgraphs.

It follows from the Excluded Grid Theorem that for minor-closed classes \( \mathcal{C} \) of graphs, \( p\text{-MC}(\text{MSO}, \mathcal{C}) \) is fixed-parameter tractable if and only if \( \mathcal{C} \) has bounded branch width. Actually, this can be slightly strengthened to classes closed under taking topological minors. I do not know of any results beyond that. To stimulate research in this direction, let me state a conjecture:

**Conjecture 8.2.** Let \( \mathcal{C} \) be a class of graphs that is closed under taking subgraphs. Suppose that the branch width of \( \mathcal{C} \) is not poly-logarithmically bounded, that is, there is no constant \( c \) such that \( \text{bw}(G) \leq \log^c |G| \) for every \( G \in \mathcal{C} \).

Then \( p\text{-MC}(\text{MSO}, \mathcal{C}) \) is not fixed parameter tractable.

Of course, with current techniques we can only hope to prove this conjecture under some complexity theoretic assumption.

For first-order logic, I have much less of an intuition. Clearly, the present results are very far from optimal. Just as an illustration, observe that if a class \( \mathcal{C} \) of graphs has a tractable first-order model checking problem, then so has the closure of \( \mathcal{C} \) under complementation. (Recall that the complement of a graph \( G = (V, E) \) is the graph \( \bar{G} = (V, (V \setminus E)) \).) However, most of the classes we considered here are not closed under complementation.

### 8.3 Structures of bounded rank width

Most of the results of this survey can easily be extended from classes \( \mathcal{C} \) of graphs to the classes \( \mathcal{C}_{\text{str}} \) of arbitrary relational structures whose underlying graphs (Gaifman graphs) are in \( \mathcal{C} \). However, this is not true for the results that involve rank width. It is not at all obvious what an appropriate notion of rank width for arbitrary structures could look like, and I think it is a challenging open problem to find such a notion.

### 8.4 Model checking for monadic least fixed-point logic

**Conjecture 8.3.** Let \( \mathcal{C} \) be a class of graphs such that \( p\text{-MC}(\text{FO}, \mathcal{C}_{\text{lb}}) \) is fixed-parameter tractable. Then \( p\text{-MC}(\text{LFP}^1, \mathcal{C}_{\text{lb}}) \) is fixed-parameter tractable.

It will be difficult to prove this conjecture, because it is related to the notoriously open problem of whether the model checking problem for the modal \( \mu \)-calculus is in polynomial time. But maybe the conjecture is wrong; refuting it might be more feasible.

### Acknowledgements

I would like to thank Bruno Courcelle, Arnaud Durand, Sang-Il Oum, Stéphan Thomassé for patiently answering various questions I had while writing this survey. Thanks to Isolde Adler, Yijia Chen, Reinhard Diestel, Jörg Flum, Magdalena Grüber, Stephan Kreutzer, Nicole Schweikardt for valuable comments on earlier drafts of the survey.

### References


