Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 251 (2009) 65–81
www.elsevier.com/locate/entcs

A Practical Approach to Courcelle's Theorem[1]

Joachim Kneis[2] Alexander Langer[3]

Dept. of Computer Science
RWTH Aachen University
Germany

Abstract

In 1990, Courcelle showed that every problem definable in Monadic Second-Order Logic (MSO) can be solved in linear time on graphs with bounded treewidth. This powerful and important theorem is amongst others the foundation for several fixed parameter tractability results. The standard proof of Courcelle's Theorem is to construct a finite bottom-up tree automaton that recognizes a tree decomposition of the graph. However, the size of the automaton, which is usually hidden as a constant in the Landau-notation, can become extremely large and cannot be bounded by any elemental function unless P = NP (Frick and Grohe, 2004). This makes the problem hard to tackle in practice, because it is just impossible to construct the tree automata.
Aiming for a practical implementation, we give a proof of Courcelle's Theorem restricted to Extended MSO formulas of the form $\mathrm{opt}\, U \subseteq V\ \varphi(U)$, where φ is a first-order formula with vocabulary (adj, U) and $\mathrm{opt} \in \{\min, \max\}$. Note that many optimization problems such as Minimum Vertex Cover, Minimum Dominating Set, and Maximum Independent Set can be expressed by such formulas. The proof uses a new technique based on using Hintikka game properties in dynamic programming. To demonstrate the usability of this approach, we present an implementation that solves such formulas on graphs with small pathwidth. It turns out that the large constants can be circumvented on graphs that are not too complex.

Keywords: Exact Algorithms, Parameterized Algorithms, Treewidth, Model-Checking, Monadic Second-Order Logic, Courcelle's Theorem

1 Introduction

There is a large number of NP-hard problems in graph theory, for which exact solutions can be computed efficiently on trees. For example, it is long-known [4,12,30,31] that all problems definable in Monadic Second-Order Logic (MSO) can be solved in linear time on the class of trees. MSO is an extension of First-Order Logic (FO), which allows quantification not only over objects, but also over sets of objects, as in, e.g., $\exists D\, \varphi(D)$, where $\varphi(D)$ is an MSO-formula with free set variable D. The classic approach for MSO model checking on trees is to construct a deterministic bottom-up tree automaton that in linear time recognizes the language defined by

[1] Supported by the DFG under grant RO 927/8
[2] Email: kneis@cs.rwth-aachen.de
[3] Email: langer@cs.rwth-aachen.de

doi:10.1016/j.entcs.2009.08.028

the formula. For finite structures (such as graphs), there are a couple of optimized implementations available, for example the MONA tool [22] developed at the University of Aarhus.

Many hard problems can even be solved efficiently on graphs that might not be trees, but are—in some sense—still sufficiently tree*like*. A formal parameter that is widely accepted to measure this likeliness is the *treewidth* of a graph. For example, trees and forests have treewidth 1, but the complete graph on n nodes has treewidth $n-1$. Interweaved with the definition of treewidth is the one of a *tree decomposition* of a graph G. Such a tree decomposition is basically a tree T annotated with graph separators of G such that certain properties hold. Treewidth and tree decompositions were introduced by Robertson and Seymour in a series of papers on graph minors [27,28,29], but turned out to be very useful in other areas as well, for example for solving hard problems on graphs. The reasons is, that one can often use the tree structure of the tree decomposition to solve the problem on G. Since the aforementioned graph separators have size bounded by the treewidth plus one, this approach can lead to algorithms with a run time polynomially bounded in $|G|$ if G has bounded treewidth. This is one of the reasons why treewidth has found many applications in parameterized complexity theory.

Parameterized complexity theory is an approach to explore whether hard problems can be solved exactly with a run time that comes close to polynomial time on "well-behaved" instances. For example, one could argue that graphs with low treewidth are "well-behaved" for a problem, if the problem is efficiently solvable on treelike graphs. Formally, a parameterized problem L is a set of pairs (I,k) where I is an *instance* and k the *parameter*. A parameterized problem L is called *fixed parameter tractable* and belongs to the complexity class FPT if there is an algorithm that decides membership of L in time $f(k)poly(|I|)$, where f is an arbitrary function. If the parameter is small, such an algorithm can be quite efficient in spite of the NP-hardness of the problem—in particular if f is a moderately exponential function. For more information on parameterized complexity, we refer the reader to introductionary texts such as [9,15,26].

In 1990, Courcelle [6] showed that each problem definable in Counting MSO, an extension of MSO with additional predicates evaluating the size of sets, can be solved in linear time on graphs with bounded treewidth. In other words, every such problem is in FPT when parameterized in the graph's treewidth. This result was generalized by Arnborg, Lagergren, and Seese [1] to Extended MSO (EMSO), which furthermore introduces relational evaluation over the size of sets. Among others, one may for example take the minimum or the maximum size among all sets satisfying a formula and evaluate this integer further.

This general and strong result, commonly referred to as "Courcelle's Theorem", allows a broad application in the field of parameterized complexity theory, sometimes even when the original parameterization is not the treewidth. For example, it is used in recent work on the parameterized complexity of the Crossing Number problem [19,21], for which no direct algorithm is known.

The standard proof of Courcelle's Theorem is again the automata-theoretic ap-

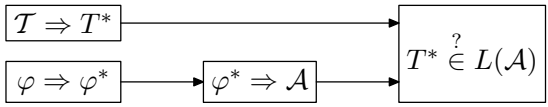

Fig. 1. The classical automata-theoretic approach to Courcelle's Theorem: Given a formula φ and a tree decomposition $\mathcal{T}$ of a graph G, one first computes a labeled binary tree T^* and a formula φ^*, such that $T^* \models \varphi^*$ iff $G \models \varphi$. Then, by the well-known methods for MSO model-checking on trees, a tree automaton $\mathcal{A}$ with $L(\mathcal{A}) = L(\varphi^*)$ is constructed that tests membership $T^* \in L(\mathcal{A})$ in linear time. Since φ^* only depends on φ and the treewidth, the construction of $\mathcal{A}$ is independent of G, the input graph.

proach. Naturally, the details are a bit more complicated than in the construction of classical tree automata for MSO formulas, but the method in general (see Figure 1) is well-understood: Given a formula and a tree decomposition, the problem is first reduced to the MSO model checking problem on labeled binary trees. The well-known methods for MSO model-checking on trees involving a deterministic tree automaton that recognizes this labeled binary tree then yield the FPT run time bounds, since the automaton only depends on φ and the treewidth, and the actual recognition of the tree decomposition can be done in linear time. In particular, the construction of $\mathcal{A}$ is independent of G, the input graph. There are a couple of other proofs of Courcelle's Theorem known (see, e.g., [6,5,1,8,9,7,17,24]). The automata-theoretic approach, however, is the most common and has been investigated to a large extent, in particular with respect to upper and lower bounds of the constants hidden in the Landau-notation [17,14,32].

Even if the algorithm asymptotically only requires linear time in the number of nodes, the size of these hidden constants can not be neglected in practical applications. Unfortunately, they can become extremely huge [18] (see also [25,23]): Unless P = NP, there is no elementary function f, such that these constants can be bounded by $f(|\Phi|, w)$, where $|\Phi|$ depends on the formula Φ defining the problem and w is the treewidth of the graph. Hence, it is not surprising that the only known upper bounds are extremely large. To obtain some real numbers, we can use the quite precise estimations Weyer [32, pp. 213ff] found: It turns out that even for for rather simple Σ_t-FO formulas (MSO adds an additional exponential) with only four quantifiers the upper bound he obtains is as large as

$$2^{2^{2^{(w+4)^{|\Phi|}}}}.$$

It is easy to see that even for constant treewidth $w = 1$ (i.e., trees and forests) automata of that size can not be constructed in practice. At the same time, the lower bound [18,32] of

$$2^{2^{2^{2^{o(|\Phi|)}}}}$$

for FO and $w = 1$ indicate that this upper bound is already quite good, at least if FTP $\neq$ W[1] holds, which is widely believed. Therefore, even restrictive MSO formulas like

$$\exists D \, \forall x_1 \exists x_2 \forall x_3 \, \varphi(D, x_1, x_2, x_3), \qquad \varphi \text{ quantifier-free}, \tag{1}$$

can cause heavy blow-up in automata size.

The automata-theoretic approaches usually suffer from a bottleneck in a similar way, even if an automaton is not constructed explicitly. Roughly speaking, all the information that is required to a allow linear run time in the second phase—the recognition or dynamic programming—is computed in a constant time first phase, let it be the construction of the automaton, or the computation of decision rules or look-up-tables for the dynamic programming. This first step usually depends on the formula and the treewidth only. For example, the automaton constructed (cf., Figure 1) is the same for all graphs bounded by that treewidth. In other words, the automaton that is computed has to contain all the states it needs to keep track of all the different "possibilities" appearing in *worst-case* instances—and hence is (non-elementary) large even if the graph that is part of the actual input to the model-checking problem is simple. A similar bottleneck occurs in related approaches. For instance, Courcelle and Mosbah [8] use direct dynamic programming on the tree decomposition. The dynamic programming step uses Boolean connectives as decision rules, which are obtained from a variant (see [6]) of the Feferman-Vaught-Theorem [13]. The Feferman-Vaught-Theorem is also used by Makowsky [24] to compute a large look-up-table for dynamic programming. Finally, Frick's approach [17] is to compute the set of all MSO types for the formula, which can again be done with the help of the Feferman-Vaught-Theorem [20]. Since, however, the constructive proof of this theorem resembles the construction of the tree automaton, it is not surprising that one experiences a similar explosion [23].

Several techniques have been developed to address this problem where possible, with mixed results (see, e.g., [2,22]). Since formulas occurring in practice are often comparably simple and do not cause this explosion, the corresponding automata typically remain much smaller. On the other hand, we find that Courcelle's Theorem is used whenever one is currently not able to give a better, direct algorithm for the problem at hand. This view is strengthened by a comment by Grohe [19] in his aforementioned work on the Crossing Number problem: "This statement involves a non-trivial quantifier alternation, which is what makes the translation to an algorithm difficult. At this point, I think that the route through logic and Courcelle's theorem is essential for our proof." It is not hard to see that quantifier alternations are the reason for the explosion in automata size. Hence, whenever we actually need Courcelle's Theorem in practice because the formula is too complex to give direct algorithms, the corresponding automaton becomes large. This is unsatisfactory, because it should be perfectly possible to solve the problem, given as a formula only, on simple graphs, e.g., long paths, even if the automaton might be too large even with tricks.

We believe that the aforementioned problems are the reason why there is de facto no practical implementation of Courcelle's Theorem available and hence none for, e.g., the Crossing Number problem. A fast algorithm would not only yield a fast implementation for this problem—important on its own—but also for any other result based on Courcelle's Theorem.

Our goal was therefore to develop an alternative proof of Courcelle's Theorem that is not subject to this bottleneck, which is independent of the graph at hand.

Instead, it should incorporate the knowledge on the whole input, i.e., take into account the graph that is part of the input, such that worst-case lower bounds remain those until both, the formula and the graph, are worst-case instances. To this end, we aimed for a more direct approach, i.e., it shall not use methods lend from the automata-theoretic background and directly work on the tree decomposition and the graph. In this paper, we cannot yet prove the full strength Courcelle's Theorem: The restriction is that for now we only allow one set quantifier, and the respective set shall be minimized or maximized in size (such as in equation 1). This yields formulas of the the form

$$\operatorname{opt} U \subseteq V \ \varphi(U),$$

where φ is a first-order formula with vocabulary (adj, U) and $\operatorname{opt} \in \{\min, \max\}$. Note that still many common optimization problems such as Minimum Vertex Cover, Minimum Dominating Set, and Maximum Independent Set can be expressed by such formulas, and that the non-elementary lower bounds also hold, since we can express all first-order properties.

The basic idea of our approach is to use dynamic programming on the tree decomposition, but in each step, we use Hintikka games (model checking games) to decide whether the formula still holds in the graph G. Model checking games are only the game-theoretic formalism that eases the proofs, but can nicely and straight-forwardly be translated to direct algorithms. Since an MSO-formula with quantifier-rank q cannot distinguish subgraphs of G iff the duplicator has a winning strategy in the Ehrenfeucht-Fraïssé-game with q rounds, we only use equivalency classes of subgraphs for the model checking game. If the graph is rather "simple", which in this context does not necessarily mean "small", there are only few ways to play the Ehrenfeucht-Fraïssé-game in q rounds, and hence few equivalency classes and the tables remain small. This way, we take into account the actual graph that is part of the input.

Of course, our approach is subject to the non-elementary worst-case lower bounds mentioned above as well, because these do already hold for FO. However, if we are stuck with a bad formula that yields a very huge automaton, we might still be able to solve the problem on a wide variety of graphs occurring in practice.

For a proof of concept and in order to show feasibility, we developed a C++ program based on our approach. At the time of this writing, it is already able to solve problems defined by a formula with few quantifiers on graphs with small pathwidth, and indeed the space requirements are much smaller than what we could expect from the current known bounds.

2 Preliminaries

We start with the notation used in this paper and the introduction of the essential concepts underlying our approach. The power set of a set U is denoted by 2^U. For graphs $G = (V, E)$ and $U \subseteq V$, the induced subgraph of U is denoted by $G[U]$ and the set of nodes of G is denoted by $V(G)$.

A *tree decomposition* (T, X) for a graph $G = (V, E)$ is a rooted tree $T = (I, F)$,

and a function $X : I \to 2^V$, such that:

(i) For each node $v \in V$ there is an $i \in I$ with $v \in X(i)$.

(ii) For each edge $\{u, v\} \in E$ there is an $i \in I$ with $\{u, v\} \subset X(i)$.

(iii) For each node $v \in V$, the subtree of T induced by $\{\, i \in I \mid v \in X(i) \,\}$ is connected.

The sets $X(i)$ are called *bags*. The width of a tree decomposition is the size of its largest bag minus one and the *treewidth* of G is the smallest width over all tree decompositions of G. Without loss of generality, we assume that all tree decompositions used throughout this paper are *nice*, i.e.,

(i) each $i \in I$ has at most two children,

(ii) if $i \in I$ has exactly one child $j \in I$, then $X(i)$ and $X(j)$ differ by exactly one node $v \in V$, and

(iii) if $i \in I$ has two children $l, r \in I$, then $X(i) = X(l) = X(r)$.

Nodes of T having two children are called *join* nodes, while nodes i with exactly one child j are called *forget* (resp. *introduce*) nodes, if $X(i) \subset X(j)$ ($X(j) \subset X(i)$). A *terminal graph* is a labeled graph (V, E, X) with *terminals* X. Two terminal subgraphs T_1 and T_2 with the same set of terminals X and $T_1[X] = T_2[X]$ can effectively be "glued" on X, denoted by $T_1 \oplus T_2$, by identifying the nodes in X. This yields a new terminal subgraph. A notable property of tree decompositions exploited by most algorithms using treewidth—including our own—is the fact that each bag $X(i)$ is a graph separator of G. Hence with each bag $X(i)$ we can associate two (forget, introduce) or three (join) well-defined terminal subgraphs with terminal set $X(i)$. For further information on treewidth and tree decompositions, we refer the reader to Bodlaender's guide to treewidth [3].

The *model-checking problem* for a logic on a domain of structures is to decide whether a given structure is a model for a formula in this logic. In this paper, we consider the domain of graphs and formulas over vocabulary $\{\mathrm{adj}\}$ in (Extended) Monadic Second-Order ((E)MSO) logic. MSO extends First-Order (FO) logic by universal and existential quantification over sets, and EMSO furthermore allows to evaluate the size of these sets (see [1]). We shall use lower case letters for first-order variables and upper case letters for monadic second-order (set) variables. Throughout this work, we only use the EMSO extension to express optimization problems and restrict ourselves to EMSO formulas of the form $\mathrm{opt}\, U\, \varphi(U)$, where $\mathrm{opt} \in \{\min, \max\}$, and φ contains exactly the free set variable U, but no further set quantification (first-order quantifiers are still allowed). In other words, φ is an FO-formula with vocabulary (adj, U). In slight abuse of the model-theoretic formalism, the semantics is that the formula shall "return" the size of an optimal set U on which $\varphi(U)$ holds. Furthermore, we wlog assume φ is in negation normal form, i.e., negations only occur on atomic expressions. The nesting depth of quantifiers in a formula Φ is called the quantifier rank $\mathrm{qr}(\varphi)$.

We only briefly introduce the game-theoretic foundations used in this paper, for more information see, e.g. [10]. Firstly, it is a well-known fact that the model

checking problem for relational MSO and FO can be expressed as a winning strategy problem in finite Hintikka (model checking) games: To any structure $\mathcal{A}$ with universe A, any formula $\Phi(\overline{x})$ of the same vocabulary in negation normal form, and any assignment $\overline{a}$ to the free variables $\overline{x}$ of Φ, we associate a model checking game $\mathrm{MCG}(\mathcal{A}, \Phi(\overline{a}))$. There are two players, the *verifier* and *falsifier*. The former tries to prove that $\mathcal{A} \models \Phi(\overline{a})$, while the latter wants to show the opposite. The game is positional with positions $(\varphi, \overline{b})$, where φ is a subformula of Φ and $\overline{b}$ is an assignment to the free variables of φ. If the current position is $(\exists u\varphi, \overline{b})$ or $(\exists U\varphi, \overline{b})$, the verifier has to choose $u \in A$ or $U \subseteq A$, and the game continues at position (φ, b'), where b' is obtained from b by assigning u or U according to his choice. At position $(\psi_1 \vee \psi_2, \overline{b})$, the verifier has to choose $\psi \in \{\psi_1, \psi_2\}$ and the game continues in $(\psi, \overline{b})$. Similarly, the falsifier moves at universal quantification or conjunctions. The game stops on positions $(\varphi, \overline{b})$ if φ is an atom or a negated atom. The verifier has won iff $\mathcal{A} \models \varphi(\overline{b})$. Then $\mathcal{A} \models \varphi(\overline{a})$ iff the verifier has a winning strategy for $\mathrm{MCG}(\mathcal{A}, \Phi(\overline{a}))$ starting in $(\Phi, \overline{a})$. We call two positions $(\varphi, \overline{b}_1)$ and $(\varphi, \overline{b}_2)$ *equivalent*, if $\mathcal{A} \models \varphi(\overline{b}_1) \Leftrightarrow \mathcal{A} \models \varphi(\overline{b}_2)$. If $\mathcal{A}$ is finite, it is easy to see that a simple recursive algorithm can be used to decide $\mathcal{A} \models \varphi(\overline{a})$ by exhaustively simulating $\mathrm{MCG}(\mathcal{A}, \Phi(\overline{a}))$.

Secondly, Ehrenfeucht-Fraïssé-games show that the quantifier rank bounds the ability of an MSO-formula to distinguish structures: Let $\mathcal{A}$ and $\mathcal{B}$ be structures over the same relational vocabulary τ. We write $\mathcal{A} \equiv_q \mathcal{B}$ iff $A \models \Phi \Leftrightarrow \mathcal{B} \models \Phi$ for every MSO-formula Φ over τ with $\mathrm{qr}(\Phi) \leq q$. The Ehrenfeucht-Fraïssé-game is played in up to $q \in \mathbf{N}$ rounds. In each round, the *spoiler* has to choose new elements or sets from $\mathcal{A}$ or $\mathcal{B}$. Next, the *duplicator* has to answer with previously not chosen elements or sets from the opposite structure, such that the selected elements and sets $\overline{a}$ in $\mathcal{A}$ and $\overline{b}$ in $\mathcal{B}$ form a partial isomorphism between $(\mathcal{A}, \overline{a})$ and $(\mathcal{B}, \overline{b})$. If one of the players cannot move, he lost the game. The duplicator wins the overall game if it is always possible to find a partial isomorphism. Then $\mathcal{A} \equiv_q \mathcal{B}$ if and only if the duplicator has a winning strategy for the Ehrenfeucht-Fraïssé-game on $\mathcal{A}$ and $\mathcal{B}$ with q rounds [16,11]. Note that if we understand the sets that have been chosen as labels, each such partial isomorphism corresponds to induced, labeled subgraphs of $\mathcal{A}$ and $\mathcal{B}$. We furthermore point out the relation between model-checking and Ehrenfeucht-Fraïssé-games: The elements and sets chosen in either structure in the latter game can be used for assignments to the free variables of Φ in the former.

Finally, the non-elementary function $\mathrm{tow}(u, v) : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ is defined as:

$$\mathrm{tow}(u, v) := \begin{cases} 2^{v \cdot \mathrm{tow}(u-1, v)} & u > 0 \\ 1 & u = 0 \end{cases}$$

3 Proving a Restricted Version of Courcelle's Theorem

From now on, we fix a graph $G = (V, E)$ with (wlog) $V \cap \mathbf{N} = \emptyset$, a nice tree decomposition (T, X) of G of width w, and an EMSO formula

$$\mathrm{opt}\, U\, \Phi(U, x_1) := \mathrm{opt}\, U\, \forall x_1 \varphi(U, x_1),$$

where opt $\in \{\min, \max\}$ and Φ is an FO-formula over vocabulary $\{\mathrm{adj}, U\}$ with quantifier rank $t := \mathrm{qr}(\Phi) = \mathrm{qr}(\varphi) + 1$.

A typical framework for solving problems with the help of the tree decomposition can be sketched as follows: For every bag $X(i)$, one first stores in a table S_i all *local* solutions. Then, by dynamic programming starting in the leaves of the tree decomposition, local solutions are extended to optimal *global* solutions where possible. We use this framework to find an optimal global solution $U \subseteq V$, i.e., a maximum or minimum set $U \subseteq V$ with $G \models \Phi(U)$. We call $U \subseteq X(i)$ a *local* solution for a node i of the tree decomposition iff for all $x_1 \in X(i)$, the verifier has a winning strategy from position $(\varphi, (x_1))$ in the game $\mathrm{MCG}(G, \Phi(U))$. It is now easy to see that $G \models \Phi(U)$ if and only if U is a local solution for every node of the tree decomposition. Traversing the latter bottom-up from the leaves to the root, we compute the (size of) optimal sets that are local solutions for all nodes seen so far. In the root, only global solutions remain.

However, we have to avoid enumerating all possible sets $U \subseteq V$ for each node of the tree decomposition, since otherwise we could also solve the initial problem within the same time. To this end, we use the observation that Φ cannot distinguish two sets $U_1, U_2 \subseteq V$ with $(V, E, U_1) \equiv_t (V, E, U_2)$. For example, consider a formula

$$\min U \, \forall x_1 \forall x_2 \psi(U, x_1, x_2)$$

with $\mathrm{qr}(\psi) = 0$ and assume the model-checking game is in position $(\psi, (U, x_1))$ for some $U \subseteq V$ and $x_1 \in X(i)$. Now the falsifier has $|V|$ possibilities to choose $x_2 \in V$, but only up to five ways to choose $x_2 \in V$ so that the new positions are pairwise inequivalent: Either choose $x_2 = x_1$, or any combination of whether x_2 is adjacent to x_1 and of whether $x_2 \in U$—these are the only five cases that influence atomic formulas. This coincides with the "effective" number of ways the Ehrenfeucht-Fraïssé-game can be played on (V, E, U) (plus one): Again, the partial isomorphism in the finite round is already determined by whether x_1 and x_2 are adjacent and if x_2 is in U. This argument easily carries over to formulas with larger quantifier ranks (even though each quantifier introduces a further exponential into the upper bound). The idea is now that for every set $U \subseteq V$ we only use a representation of the equivalence class of (V, E, U) under the Ehrenfeucht-Fraïssé-game, which can be enumerated faster. Then, given the set of all equivalence classes, we can decide whether members of each class are local solutions for $X(i)$ within FPT run time bounds, and in the dynamic programming, we only save the size of optimal solutions in every equivalence class.

The representation that we use here is a rooted tree that describes how the Ehrenfeucht-Fraïssé-game can be played "locally" on a terminal subgraph T of G: Since the sets shall be local solutions (defined in terms of model-checking games), we also restrict the first round of the Ehrenfeucht-Fraïssé-game to the terminals, but at the same time leave open which which $x \in X(i)$ has been explicitly chosen in the first round. We call these Ehrenfeucht-Fraïssé-games *local*. Then each node of the tree representation corresponds to a round in the local Ehrenfeucht-Fraïssé-game, where the root represents the (open) choice of some $x \in X(i)$, and each other

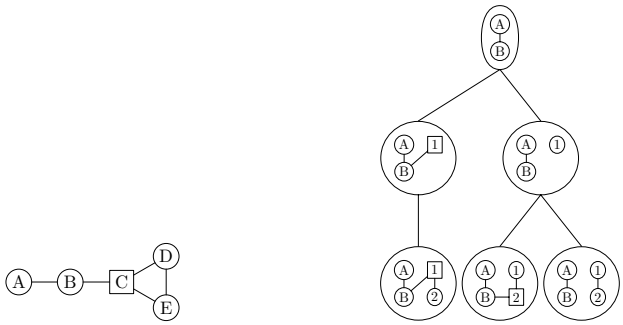

Fig. 2. The Ehrenfeucht-Fraïssé-game representation $\mathrm{EFG}_3(T, U)$ for the terminal graph T on the left with terminals $X = \{A, B\}$ and $U = \{C\}$ (shown as a square node): we restrict the games such that in the first round the players always have to choose any $x \in X$ (root of the tree). In the next round, the players effectively only have two choices to pick the first non-terminal (labeled 1): Either pick the non-terminal C, which yields the left subgame, or choose either of $\{D, E\}$ for the right subgame: The nodes D and E always yield the same induced labeled subgraphs (up to isomorphism) in the following round, where the second non-terminal is chosen (labeled 2), and hence effectively do not introduce new subgames.

node corresponds to the choice of a certain non-terminal. Each node is labeled with the partial isomorphism that is obtained in that round or, more precisely, with the current induced labeled subgraph (see Figure 2 for an example).

Definition 3.1 Let X be a bag, T be a terminal subgraph of G, $U \subseteq V(T)$, and for each $k \geq 0$ let $(v_1, \ldots, v_k) \in (V(T) \setminus X)^k$, such that $v_i \neq v_j$ for each $i \neq j$. The *partial isomorphism representation* $SUB(T, v_1, \ldots, v_k, U)$ of T, $(v_1, \ldots, v_k)$ and U is obtained from the labeled (terminal) subgraph $(T[X \cup \{v_1, \ldots, v_k\}], U \cap (X \cup \{v_1, \ldots, v_k\}))$ by identifying each non-terminal v_j as the integer j (recall that $V \cap \mathbf{N} = \emptyset$).

For any terminal graph T, any set $U \subseteq V(T)$, each $0 \leq k \leq t - 1$ and each $(v_1, \ldots, v_k) \in (V(T) \setminus X)^k$ we recursively define labeled trees $\mathrm{EFG}_{v_1 \cdots v_k}$ as follows. The root of $\mathrm{EFG}_{v_1 \cdots v_k}$ is labeled with $SUB(T, v_1, \ldots, v_k, U)$. Furthermore, if $k < t - 1$, for each

$$\mathrm{EFG} \in \{\mathrm{EFG}_{v_1 \cdots v_k v} \mid v \in V(T) \setminus X \wedge v \neq v_1, \ldots, v_k\}$$

there is a subtree EFG. The *Ehrenfeucht-Fraïssé-game representation* of T and U is defined as $\mathrm{EFG}_t(T, U) := \mathrm{EFG}_\epsilon$, where ϵ is the empty sequence.

If for some set $U \subseteq V$ with $U \not\subseteq V(T)$, we may abbreviate $\mathrm{EFG}_t(T, U \cap V(T))$ as $\mathrm{EFG}_t(T, U)$. It turns out (see the next lemma) that the number of different local Ehrenfeucht-Fraïssé-games is bounded from above in terms of $w \geq |X| - 1$ (the treewidth) and quantifier rank only and particularly does not depend on the size of G (the input size). This is good for the FPT run time we want to achieve, but bad if we want to avoid the across-the-board explosion we mentioned in the introduction. Note however, that even though each quantifier introduces an exponential into the upper bound—which is to be expected due to the known lower bounds—the real number of local games obtained depends on the input (terminal) graph, as the example in Figure 2 shows.

Lemma 3.2 *Let X be a bag (i.e., $|X| - 1 \leq w$), and T be a terminal graph. The number of non-equivalent Ehrenfeucht-Fraïssé-game representations* $\mathrm{EFG}_t(T, U)$*, i.e., the size of*

$$\{\, \mathrm{EFG}_t(T, U) \mid U \subseteq V(T) \,\},$$

is bounded by $O(2^w 2^{\mathrm{tow}(t-1, w+t+2)})$ and the size of each representation is bounded by $O(\mathrm{tow}(t-1, w+t+2))$.

Proof. We prove the upper bound by induction over the number of rounds in the local Ehrenfeucht-Fraïssé-game. Recall that we only consider those games where the first round was chosen in X, but we leave open, which x has been explicitly chosen. Hence, we consider all of these games using one representation. For any $0 \leq k < t$, we by n_k denote the number (sub)games the players can play for arbitrary $U \subseteq V(T)$ (structure (T, U)), when

(i) beginning in round $k+1$ with previous choices $(x, v_1, \ldots, v_k)$,

(ii) for all $x \in X$ (at the same time),

(iii) but with distinct $v_1, \ldots, v_k \in V(T) \setminus X$, $v_i \neq v_j$.

If $k = t - 1$, the game is over and hence only the empty game can be played. If $k < t - 1$, the players have to move to some new $v \in V(T)$ (since we consider all $x \in X$ at the same time), if possible. There are $2 \cdot 2^{w+1} \cdot 2^k = 2^{w+k+2}$ possibilities for v, such that $(x, v_1, \ldots, v_k, v)$ yields a distinct induced labeled subgraph: Either choice of $v \in U$, an edge between v and any $x \in X$ and to the previous k nodes, and any such v might be present in the current structure (T, U) or not. By induction, we obtain an upper bound of $2^{2^{w+k+2} n_{k+1}}$, and with $k < t$, we obtain the second factor. The additional factor $O(2^w)$ is due to the at most 2^{w+1} ways to choose $U \subseteq X$.

The size bound for each fixed game is obtained analogously. □

We now know, that each bag X splits the graph into two or three terminal subgraphs, and for each such terminal subgraph T and set $U \subset V(T)$ we found a way to represent the locally restricted Ehrenfeucht-Fraïssé-games on (T, U). We now show that the opposite holds as well, i.e., when terminal graphs T_1 and T_2 are glued, we can construct the corresponding Ehrenfeucht-Fraïssé-game representation for $T_1 \oplus T_2$ and any $U_1 \subseteq V(T_1)$ and $U_2 \subseteq V(T_2)$. This will be needed for join nodes in the dynamic programming as well as for finding local solutions (by the corresponding model-checking game).

Lemma 3.3 *Let T_1 and T_2 be terminal graphs with the same set of terminals X and $T_1[X] = T_2[X]$, and let $U_1 \subseteq V(T_1)$ and $U_2 \subseteq V(T_2)$ with $U_1 \cap X = U_2 \cap X$. Given $E_1 = \mathrm{EFG}_t(T_1, U_1)$ and $E_2 = \mathrm{EFG}_t(T_2, U_2)$, one can compute* $\mathrm{EFG}_t(T_1 \oplus T_2, U_1 \cup U_2)$ *in time poly$(|E_1|, |E_2)|)$.*

Proof. Since no two nodes from $V(T_1) \setminus X$ and $V(T_2) \setminus X$ are adjacent, they will not be adjacent in $T := T_1 \oplus T_2$ either. Hence, each round of the local Ehrenfeucht-Fraïssé-game on $T_1 \oplus T_2$ is already determined by the possible moves on (T_1, U_1) and (T_2, U_2): Assume the game is in round $k+1$, i.e., $(x, v_1, \ldots, v_k)$ have already been chosen, where we again assume x is a placeholder for any terminal. If $k < t - 1$, the

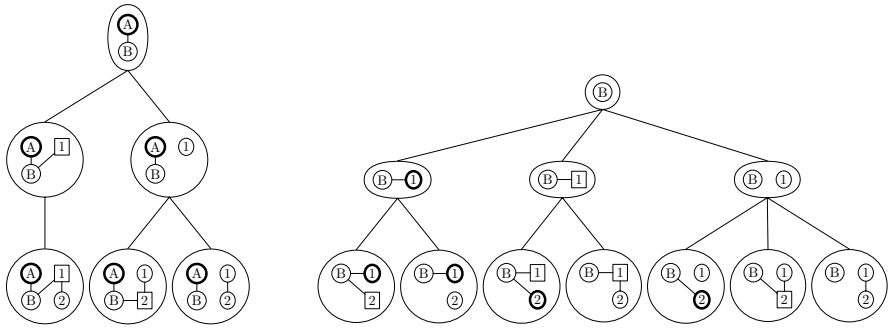

Fig. 3. Forgetting a terminal in the Ehrenfeucht-Fraïssé-game representation: The original terminal set is $\{A, B\}$ (left), but A is forgotten and hence is no longer a terminal (right). Since we only study those games where the first round is restricted to terminals, A no longer occurs in the root. However, it can still be chosen in the next two rounds, which adds a couple of new moves to the game. The places where A, now a non-terminal, has been chosen are marked bold.

player now has to choose some $v \in V(T) \setminus (X \cup \{v_1, \ldots, v_k\})$. However, since any such v originates in either T_1 or T_2, we can just enumerate all possible choices that are available in $\mathrm{EFG}_t(T_1, U_1)$ or $\mathrm{EFG}_t(T_2, U_2)$.

We omit the full algorithm since the actual construction is rather technical and boring. Instead, we only hint how a simple recursive algorithm can traverse E_1 and E_2. In some sense, the algorithm exhaustively simulates the Ehrenfeucht-Fraïssé-game and works as follows: There are an integer $k \leq t$ counting the current round, two pointers p_1 and p_2 that mark the current position in E_1 and E_2, respectively, and a pointer for position p in the to-be-constructed $\mathrm{EFG}_t(T, U_1 \cup U_2)$: If $k < t - 1$, for every child c of p_1 a corresponding child c' is added below p and the algorithm recursively continues at positions c, p_2 and c' and with $k' := k + 1$. The same is done for all children of p_2. When finished with p_1 and p_2, duplicates below p have to be eliminated. □

As mentioned, this lemma can already be used in the dynamic programming to construct the possible game representations for join nodes. Of course, we also need to make sure that we can do so for introduce and forget nodes as well. For the former, this task is easily done.

Lemma 3.4 *Let i be an introduce node of the tree decomposition and j its unique child. Let $X(i) = X(j) \cup \{x\}$ be the corresponding bags, and let T_i and T_j be the terminal graphs with terminal sets $X(i)$ and $X(j)$, respectively, such that $T_i[X(i) \setminus \{x\}] = T_j$. Then $\{\, \mathrm{EFG}_t(T_i, U) \mid U \subseteq V(T_i) \,\}$ can computed from $\{\, \mathrm{EFG}_t(T_j, U) \mid U \subseteq V(T_j) \,\}$ in linear time.*

Proof. The node x is the only new node in T_j and is a terminal. Since the Ehrenfeucht-Fraïssé-games are local, x will always be chosen in the first round. On the other hand, x is not adjacent to any node in $V(T_i) \setminus X(i) = V(T_j) \setminus X(j)$. Hence, we only need to slightly modify the labels for each game-representation on T_j accordingly, which yields two representations corresponding to the cases $x \in U \subseteq V(T_j)$ and $x \notin U \subseteq V(T_j)$ for each $\mathrm{EFG} \in \{\, \mathrm{EFG}_t(T_j, U) \mid U \subseteq V(T_j) \,\}$. The algorithm is straight-forward. □

Forget nodes require slightly more work: The "forgotten" node becomes a non-terminal, and thus in the local game we will never choose it in the first round. Instead, it can now be chosen in any subsequent round. This possibly adds new subtrees to the game representation whenever it has not been chosen in previous rounds (see Figure 3).

Lemma 3.5 *Let i be a forget node of the tree decomposition and j its unique child. Let $X(i) = X(j) \setminus \{x\}$ be the corresponding bags, and let T_i and T_j be the terminal graphs with terminal sets $X(i)$ and $X(j)$, respectively, such that $T_j = T_i$ (seen as regular graphs). Then for any $U \subseteq V(T_i)$, $E_i := \mathrm{EFG}_t(T_i, U)$ can computed from $E_j := \mathrm{EFG}_t(T_j, U)$ in time poly$(|E_j|)$.*

Proof. (Sketch.) Since $V(T_i) = V(T_j)$ and the set U remains the same, it is easy to see that we only need to make sure that x is correctly used in the (representation of the) local Ehrenfeucht-Fraïssé-game.

We omit the technical and not very interesting algorithm, and again give implementation hints only: As before, the algorithm in some sense exhaustively simulates the local Ehrenfeucht-Fraïssé-game on (T_i, U) that we want to represent: There are two functions. The first one recursively traverses E_j top-down, and at each node of E_j first calls itself for all children. This first step corresponds to the case that a non-terminal not equal to x is chosen, which still allows x to be taken in subsequent rounds. In a second step, we add a subtree for the case that x is chosen. This subtree for x is obtained from the current subtree of E_j by a second function: The leaves might have to be cut off (only t rounds are played), and the labels have to be modified in a way such that x becomes a non-terminal chosen in the current round k, and all other non-terminals $j > k$ must have their indices increased. Finally, the label for the root of the new tree is determined from the old tree by just deleting x, and duplicate subtrees, i.e., those that represent equivalent subgames, must be eliminated. □

We can now sketch our main algorithm (depicted in Algorithm 1), and obtain our main result.

Theorem 3.6 *Algorithm 1 returns an optimal solution $U \subseteq V$ for $\Phi(U)$ in time bounded by*

$$O(2^w 2^{4 \cdot \mathrm{tow}(t-1, w+t+2)} \cdot |T|).$$

Proof. Wlog, let opt = max. For any node i, let $T_{i,p}$ be the terminal graph in the *past* of i, i.e., the terminal graph that contains all nodes in bags below i, and let $T_{i,f}$ be the *future* of i, i.e., the unique $T_{i,f}$, such that $G = T_{i,p} \oplus T_{i,f}$.

By a straight-forward induction over the first bottom-up traversal, one first can show (omitted here) with the help of Lemmata 3.3, 3.4 and 3.5, that for each i, P_i contains an entry $\mathrm{EFG}_t(T_{i,p}, U)$ for all $U \subseteq V(T_{i,p})$. Similarly, the top-down traversal yields an entry $\mathrm{EFG}_t(T_{i,f}, U)$ in F_i for all $U \subseteq V(T_{i,f})$ and all i.

By an induction over the second bottom-up traversal of i, we shall now show that the algorithm computes the correct values. That is, if S_i contains an entry

Algorithm 1 *Input: A graph $G = (V, E)$, a nice tree decomposition (T, I) of G of width w, an EMSO-formula $\operatorname{opt} U\, \Phi(U) = \operatorname{opt} U\, \forall x_1 \varphi(U, x_1)$ without set quantifiers in Φ and $\operatorname{qr}(\Phi) = t$.*
Output: The size of an optimal set U with $G \models \Phi(U)$.

(i) *For each $i \in V(T)$, create empty tables P_i and F_i (called* past *and* future*) that can hold a set of local Ehrenfeucht-Fraïssé-game representations together with a list of pointers to entries in P_j and F_j for each child j of i.*

(ii) *Traverse T bottom-up and update the* past *P_i accordingly: For leaves i, enumerate all $U \subseteq X(i)$ and insert $\mathrm{EFG}_t(G[X(i)], U)$ into P_i. For join, introduce and forget-nodes, use the algorithms of Lemmata 3.3, 3.4, and 3.5, respectively.*

(iii) *In the root r, enumerate all $U \subseteq X(r)$ and insert $\mathrm{EFG}_t(G[X(r)], U)$ into F_i.*

(iv) *Traverse T top-down, and update the* future *F_i accordingly: introduce and forget use the respective opposite algorithm for their children; for join nodes i with two children l and r, F_l is "joined" from F_i and P_r; F_r analogue.*

(v) *For each $i \in V(T)$, create a table S_i that for pairs of game-representations holds values in $\mathbf{N} \cup \{\text{fail}\}$, where $\text{fail} \in \{+\infty, -\infty\}$ (depending on* opt *) is the default value.*

(vi) *Traverse T bottom-up and update each S_i as follows: If i is a leaf, introduce, or join node, then for each $E_p := \mathrm{EFG}_t(T_p, U_p) \in P_i$ and $E_f := \mathrm{EFG}_t(T_f, U_f) \in F_i$, use the model-checking game to test whether $U_p \cup U_f$ is a local solution for $T_p \oplus T_f = G$. If not, let $S_i[E_p, E_f] := \text{fail}$ and continue. If yes, $S_i[E_p, E_f]$ must be updated using entries in the previously computed tables (here, we use the pointers):*
- *If i is a leaf node, just set $S_i[E_p, E_f] := |U_p \cap X(i)|$, which can be deducted from E_p.*
- *If i is an introduce node with child j, let E'_p be the unique entry from which E_p was obtained bottom-up, and let E'_f be the entry obtained by E_f top-down, such that $x \in U$ in E'_f iff $x \in U$ in E'_p. Then let $S_i[E_p, E_f] := S_j[E'_p, E'_f] + |\{x\} \cap U'|$.*
- *If i is a forget node with child j, let $S_i[E_p, E_f]$ be the optimal value among all entries $S_j[E'_p, E'_f]$, such that E_p was obtained from E'_p bottom up, and E'_f is the unique entry that was obtained from from E_f top-down.*

If otherwise i is a join node with children l and r, we iterate over all $E_{l,p} := \mathrm{EFG}_t(T_l, U_l) \in P_l$, all $E_{r,p} := \mathrm{EFG}_t(T_r, U_r) \in P_r$ and all $E_f := \mathrm{EFG}_t(T_f, U_f) \in F_i$ (where U_l, U_r and U_f are only implicit). Let $E_p := \mathrm{EFG}_t(T_l \oplus T_r, U_l \cup U_r)$ (Lemma 3.3). We now again use the model-checking game to test whether $U_f \cup U_l \cup U_r$ is a local solution for $T_f \oplus T_l \oplus T_r = G$. If not, let $S_i[E_p, E_f] := \text{fail}$ and continue. Otherwise, we possibly update S_i, i.e., we let

$$S_i[E_p, E_f] := \operatorname{opt}\{S_i[E_p, E_f], s_l + s_r - |U_p \cap X(i)|\},$$

where $s_l = S_l[E_{l,p}, \mathrm{EFG}_t(T_f \oplus T_r, U_f \cup U_r)]$, $s_r = S_r[E_{r,p}, \mathrm{EFG}_t(T_f \oplus T_l, U_f \cup U_l)]$, and $|U_p \cap X(i)|$ can be deducted from E_p.

$S_i[E_p, E_f] \in \mathbf{N}$, then $S_i[E_p, E_f]$ is the optimal size $|U \cap V(T_{i,p})|$ among all $U \subseteq V$ such that

(i) $E_p = \mathrm{EFG}_t(T_{i,p}, U)$,

(ii) $E_f = \mathrm{EFG}_t(T_{i,f}, U)$, and

(iii) U is a local solution on i and on all nodes j below i in the tree decomposition.

Otherwise, $S_i[E_p, E_f] =$ fail iff there is no such set. This then yields the size of an optimal global solution in root table.

For leaves i, the claim is easily seen, because $V(T_{i,p}) = X(i)$. For the induction step, we only prove the "join" case; the induction steps for introduce and forget nodes are deducted similarly. Hence, let i be a join node with children l and r, and by hypothesis assume the claim holds for S_l andS_r. Furthermore, let $U \subseteq V$ be such that U is optimal for the aforementioned properties. Since F_i, P_l and P_r are complete, they contain entries $\mathrm{EFG}(T_{i,f}, U)$, $\mathrm{EFG}(T_{l,p}, U)$, and $\mathrm{EFG}(T_{r,p}, U)$, respectively. Let $s_l := S_l[\mathrm{EFG}_t(T_{l,p}, U), \mathrm{EFG}_t(T_{l,f}, U)]$. By hypothesis, s_l is the optimal value among all sets that are local solutions for l and all nodes below l, and hence $s_l \geq |U \cap V(T_{l,p})|$ and analogously, $s_r \geq |U \cap V(T_{r,p})|$. Thus,

$$S_i[\mathrm{EFG}_t(T_{l,p} \oplus T_{r,p}, U), \mathrm{EFG}_t(T_{i,f}, U)] = s_l + s_r - |X(i) \cap U| \geq |U \cap T_{i,p}|.$$

For equality, assume wlog $s_l \neq |U \cap V(T_{l,p})|$. Hence, there is $U' \subseteq V$, such that

(i) $\mathrm{EFG}_t(T_{l,p}, U') = \mathrm{EFG}_t(T_{l,p}, U)$,

(ii) $\mathrm{EFG}_t(T_{l,f}, U') = \mathrm{EFG}_t(T_{l,f}, U) = \mathrm{EFG}_t(T_{r,p} \oplus T_{i,f}, U)$,

(iii) U' is a local solution for l and all nodes below l in the tree decomposition, and

(iv) $|U' \cap V(T_{l,p})| > |U \cap V(T_{l,p})|$.

Let $U'' := (U' \cap V(T_{l,p})) \cup (U \setminus V(T_{l,p}))$. Then $\mathrm{EFG}_t(T_{l,p}, U'') = \mathrm{EFG}_t(T_{l,p}, U') = \mathrm{EFG}_t(T_{l,p}, U)$ and $\mathrm{EFG}_t(T_{i,f}, U'') = \mathrm{EFG}_t(T_{i,f}, U)$, and hence U'' is a also local solution for r and all nodes below r. Similarly, U'' is a local solution for l and all nodes below l. Therefore, U'' is a local solution for i and all nodes below i, but $|U \cap V(T_{i,p})| < |U'' \cap V(T_{i,p})|$—a contradiction to the choice of U.

Finally, Lemma 3.2 yields all mentioned run time bounds, where the 4 in $2^{4\mathrm{tow}\ldots}$ originates from the cubic complexity required for join, plus a rather pessimistic upper bound of the time required to modify and compute game representations. Note that the first factor still remains $2 \cdot 2^w = |2^{U \cap X(i)}|$ since we can fix $U \cap X(i)$ before doing anything else. □

On a last remark, we note that Algorithm 1 can easily be modified to work for formulas of the form $\mathrm{opt}\, U\, \exists x_1 \varphi(U, x_1)$.

4 Implementation

In the introduction we mentioned that our long-term goal is to obtain a practical implementation of Courcelle's Theorem that can be used for a wide variety of problems. Currently, such an implementation is already in development. It is written

in C++ and strictly follows Algorithm 1 sketched above. The current state of work is seen as a proof-of-concept, whose primary purpose is to prove feasibility of the approach presented in this paper. In particular, we want to use it to estimate the table size for real graphs occurring in practice. Currently, the parts involving join nodes are still under development, which means we currently can only work on path decompositions of graphs, i.e., where the tree decomposition itself is a path. Since in general the pathwidth is larger than the treewidth, this is no real limitation for our purposes.

Despite its unoptimized prototype state, the implementation is already now able to solve optimization given as a formula with few quantifiers on graphs with small pathwidth, e.g., a grid of size 2×100. Among the problems that we tested are Minimum Vertex Cover, Minimum Dominating Set, Maximum Independent Set, and—to a limited extend–also Minimum Distance Two Dominating Set (MDTDS), which requires an additional quantifier.

It turns out that indeed the table sizes on graphs occurring in practice are much smaller than the incredible large upper bound suggests. For example, on paths of arbitrary length the tables P_i and F_i contain only up to 960 entries for all formulas with three FO quantifiers (including, e.g., MDTDS). However, the implementation itself is quite naive and requires a lot more work and optimization before it can compete with existing tools like MONA.

5 Conclusion

The main result of this paper is an alternative proof for a restricted version of Courcelle's Theorem. Our direct approach uses dynamic programming and respects the input graph to the model-checking problem early. Even a naive implementation without further optimization can avoid space explosion on "simple" graphs.

However, a lot of work remains: Firstly, our method needs to be adopted to all formulas in MSO, in particular we need to allow more set quantifiers. While we believe that task is not too hard, EMSO contains evaluation terms and relations that must be addressed by the framework.

Secondly, Courcelle's Theorem and its extensions also hold for two-sorted (E)MSO, where quantification over sets of edges is allowed. Two-sorted MSO is a proper superset of the one-sorted MSO used in this paper, for example one can express the Hamiltonian Path problem, which is known to be impossible in one-sorted MSO. Using two-sorted structures as in, e.g., [1], i.e., a structure with a disjoint set of *objects* and two relations V and E that distinguishes vertex and edge objects, our method should carry over to two-sorted MSO. However, such structures have larger treewidth (each edge object appears in each bag), and it is open how this affects the running time of our approach.

Finally, even though the preliminary results encourage further development and optimization of our implementation, a lot of work remains until a real and widely applicable implementation is available. In particular, the formula used for the Crossing Number problem mentioned in the introduction of this work contains many

quantifiers, and additional tricks are required to avoid the explosion even on comparably simple graphs. Maybe we can also combine the automata-theoretic methods with our approach for an overall improvement.

References

[1] S. Arnborg, J. Lagergren, and D. Seese. Easy Problems for Tree-Decomposable Graphs. *J. Algorithms*, 12(2):308–340, 1991.

[2] M. Biehl, N. Klarlund, and T. Rauhe. Algorithms for guided tree automata. In *First International Workshop on Implementing Automata, WIA '96*, Lecture Notes in Computer Science, pages 6–25, 1997.

[3] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[4] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[5] B. Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 193–242. MIT Press, Cambridge, MA, USA, 1990.

[6] B. Courcelle. The monadic second order theory of Graphs I: Recognisable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[7] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[8] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1-2):49–82, 1993.

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[10] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999.

[11] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

[12] C. Elgot. Decision problems of finite-automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

[13] S. Feferman and R. Vaught. The first order properties of algebraic systems. *Fund. Math*, 47:57–103, 1959.

[14] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[16] R. Fraïssé. Sur quelques classifications des systèmes de relations. *Publications Scientifiques de l'Université d'Alger*, Series A(1):35–182, 1954.

[17] M. Frick. *Easy Instances for Model Checking*. PhD thesis, Universität Freiburg, 2001.

[18] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1–3):3–31, 2004.

[19] M. Grohe. Computing crossing numbers in quadratic time. *Journal of Computer and System Science*, 68(2):285–302, 2004.

[20] M. Grohe. Logic, graphs, and algorithms. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, pages 357–422. Amsterdam University Press, Amsterdam, 2007.

[21] K. Kawarabayashi and B. Reed. Computing crossing number in linear time. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 382–390, 2007.

[22] N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA Implementation Secrets. In *Proc. of CIAA00*, pages 182–194. Springer-Verlag, 2001.

[23] R. E. Ladner. Application of Model Theoretic Games to Discrete linear orders and finite automata. *Information and Control*, 33(4):281–303, 1977.

[24] J. A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.

[25] A. R. Meyer. Weak monadic second-order theory of successor is not elementary recursive. In *Proc. Symposium on Logic*, volume 453 of *Lecture Notes in Mathematics*, pages 132–154, 1975.

[26] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[27] N. Robertson and P. D. Seymour. Graph minors I. Excluding a forest. *Journal on Combinatorial Theory Series B*, 35:39–61, 1983.

[28] N. Robertson and P. D. Seymour. Graph minors—a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge University Press, 1985.

[29] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[30] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[31] W. Thomas. *Languages, automata, and logic*, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[32] M. Weyer. *Modifizierte parametrische Komplexitätstheorie*. PhD thesis, Universität Freiburg, 2008.