Special tree-width and the verification of monadic second-order graph properties

Bruno Courcelle[1]

1 **Institut Universitaire de France**
Bordeaux University and LaBRI (CNRS)
F-33405, Talence, France
courcell@labri.fr

Abstract

The model-checking problem for *monadic second-order logic* on graphs is *fixed-parameter tractable* with respect to tree-width and clique-width. The proof constructs finite deterministic automata from monadic second-order sentences, but this computation produces automata of hyper-exponential sizes, and this is not avoidable. To overcome this difficulty, we propose to consider particular monadic second-order graph properties that are nevertheless interesting for Graph Theory and to *interpret automata* instead of trying to compile them (joint work with I. Durand).

For checking monadic second-order sentences written with *edge set quantifications*, the appropriate parameter is tree-width. We introduce *special tree-width*, a graph complexity measure between path-width and tree-width. The corresponding automata are easier to construct than those for tree-width.

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

It is well-know from [9,12,14] that the model-checking problem for monadic second-order (MS) logic on graphs is fixed-parameter tractable (FPT) with respect to tree-width and clique-width. The proof uses two main notions. First tree-decompositions (for tree-width as parameter) and k-expressions (for clique-width as parameter), and second, constructions of finite deterministic automata from the MS sentences that express the properties to check. These constructions use inductions on the structure of sentences. "Small" deterministic automata are built for atomic formulas. Conjunction and disjunction are reflected by products of automata. Existential quantification is easy but it introduces nondeterminism. Universal quantifications are replaced by negations and existential quantifications. Negation is reflected by complementation hence is easy on deterministic automata, but since existential quantifications produce nondeterminism, determinization must be performed before each application a complementation and this is the source of the hyper-exponential sizes of the constructed automata.

Two difficulties arise. Although the fact that a graph has tree-width at most k can be checked in linear time, the corresponding algorithm (by Bodlaender, see [12]) is not practically usable. The situation is even more difficult for clique-width (see [20] or Chapter 6 of [6]). One can argue that graphs are frequently given *with* decompositions witnessing that their tree-width or clique-width is at most some fixed k, but another difficulty arises : the automata to be constructed are extremely large and their computations abort by lack of memory space. This is actually unavoidable if one wants algorithms taking as input any MS sentence (see, e.g., [15, 21, 22]). One possibility is to forget the idea of implementing the general theorem, and to work only on particular problems, as in [16-19] but we do not

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–18

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

follow this direction: we present some techniques that can improve the situation in many significant cases.

First, we limit our constructions to a fragment of MS logic whose sentences have limited alternations of quantifiers but that has nevertheless an interesting expressive power. Using Boolean set terms also helps to limit quantifier alternation and does not cost much in terms of sizes of automata. Second, we define deterministic automata for some basic graph properties and not only for the atomic formulas as is done usually in the proof of the general construction. Third, we *do not compile automata*, but instead we recompute their transitions whenever they are needed. Determinization is done "on the fly". These three ideas arise from joint work with I.Durand ([8]). Together with the necessary background notions, they will be presented in Sections 2 and 3 for graphs of bounded clique-width.

In Sections 4 to 6, we will explain how these constructions can be adapted, for graphs of bounded tree-width, to the verification of MS properties expressed with edge set quantifications. It appears that the corresponding automata, even for the basic property of adjacency, have exponential size in the bound on tree-width. This exponential blow-up does not occur if one uses path-decompositions instead of tree-decompositions. This observation motivates the introduction of *special tree-width*, a graph complexity measure intermediate between path-width and tree-width, and for which the basic automata are no more difficult to construct (or to specify) than for graphs of bounded clique-width. We will present some results that enlighten the differences between tree-width and special tree-width.

In this communication, we present the ideas and the main results. Technical details and proofs can be found in Chapter 6 of [6] and in [7].

2 Clique-width

Graphs are finite and simple. Just to shorten the definitions, we consider loop-free graphs. An undirected edge is handled as a pair of opposite directed edges. Each vertex has a label in a set C that we first take equal to $[k] := \{1, ..., k\}$ for some positive integer k. We denote by $\pi(G)$ the set of labels of the vertices of a graph G and by $\pi_G(x)$ the label of a vertex x. The operations on graphs are $\oplus$, the union of disjoint graphs, the unary relabelling $relab_h$ that changes every label a into $h(a)$ (where h is a mapping from C to C) and the unary edge-addition $\overrightarrow{add}_{a,b}$ that adds directed edges from every vertex labelled a to every vertex labelled b where $a \neq b$. Since we wish to define simple graphs, parallel edges are fused (hence $\overrightarrow{add}_{a,b}(\overrightarrow{add}_{a,b}(G)) = \overrightarrow{add}_{a,b}(G)$). A different interpretation of $\overrightarrow{add}_{a,b}$ can be given (cf. Section 5) so as to define graphs with multiple edges. For constructing undirected graphs, we use $\overrightarrow{add}_{a,b} \circ \overrightarrow{add}_{b,a}$ that we abbreviate into $add_{a,b}$. We will denote $relab_h$ by $relab_{a\to b}$ if $h(a) = b$ and $h(c) = c$ if $c \neq a$. The constant symbol $\mathbf{a}$ denotes one vertex (with no edge) labelled by $a \in C$. We let F_k be the set of these operations and constant symbols. Every term t in $T(F_k)$ is called a *k-expression* and defines a graph $G(t)$ with vertex set equal to the set occurences of the constant symbols in t. A graph has *clique-width* at most k if it is isomorphic to $G(t)$ for some t in $T(F_k)$.

Terms representing graphs and properties of their vertices

Let $P(X_1, ..., X_n)$ be a property of sets of vertices $X_1, ..., X_n$ of a graph $G(t)$, denoted by a term t in $T(F_k)$. Here are some examples of properties: $Link(X, Y)$: there is an edge from some x in X to some y in Y; $Dom(X, Y)$: for every x in X, there is an edge from

some y in Y to x; $Path(X,Y) : X$ has two vertices linked by a path in $und(G[Y])$ ($G[Y]$ is the subgraph of G induced on Y and $und(G[Y])$ is the corresponding undirected graph) and $Conn(X) : G[X]$ is connected.

We let $F_k^{(n)}$ be obtained from F_k by replacing each constant $\mathbf{a}$ by the constants $(\mathbf{a}, w)$ where $w \in \{0,1\}^n$ and we let $pr : F_k^{(n)} \to F_k$ be the mapping that erases the sequences w. It extends into $pr : T(F_k^{(n)}) \to T(F_k)$. A term t in $T(F_k^{(n)})$ defines the graph $G(pr(t))$ and the n-tuple of sets of vertices $(A_1, ..., A_n)$ such that A_i is the set of vertices which are occurrences of constant symbols $(\mathbf{a}, w)$ such that the i-th component of w is 1. Then, if $P(X_1, ..., X_n)$ is a property as above, we define $L_{P(X_1,...,X_n),k}$ as the set of terms t in $T(F_k^{(n)})$ such that $P(A_1, ..., A_n)$ is true in $G(pr(t))$, where $(A_1, ..., A_n)$ is the n-tuple of sets of vertices encoded by t.

3 Monadic second-order logic

Graph properties can be expressed by monadic second-order formulas (more generally by formulas of any logical language) via two (main) representations of graphs by relational structures. The first representation associates with every graph G the logical structure $\lfloor G \rfloor := \langle V_G, edg_G \rangle$ where edg_G is the binary relation on vertices such that $(x,y) \in edg_G$ if and only if there is an edge from x to y; the relation edg_G is symmetric if G is undirected. The second representation will be discussed below in Section 4.

Monadic second-order formulas will be written with the set variables $X_1, ..., X_n, ...$ (without first-order variables), with the atomic formulas $X_i \subseteq X_j$, $X_i = \emptyset$, $Sgl(X_i)$ (to mean that X_i denotes a singleton set) and $edg(X_i, X_j)$ (to mean that X_i and X_j denote singleton sets $\{x\}$ and $\{y\}$ such that $(x,y) \in edg_G$), and without universal quantifications. These synctactical constraints are not a loss of generality. A graph property $P(X_1, ..., X_n)$, where $X_1, ..., X_n$ denote sets of vertices, is an *MS graph property* if there exists an MS formula $\varphi(X_1, ..., X_n)$ such that, for every graph G and for all sets of vertices $X_1, ..., X_n$ of this graph, we have:

$$\lfloor G \rfloor \models \varphi(X_1, ..., X_n) \text{ if and only if } P(X_1, ..., X_n) \text{ is true in } G.$$

For each MS property $P(X_1, ..., X_n)$, the set of terms $L_{P(X_1,...,X_n),k}$ is regular, hence is the set accepted by a finite automaton over the functional signature $F_k^{(n)}$. However, the corresponding automata are frequently much too large to be constructed. This is due partly to the level of nesting of negations in the formulas but also to the number k: for example, the number of states of the minimal automaton recognizing $L_{Conn(X_1),k}$ is a two-level exponential in k. Instead of trying to construct automata for the most general sentences, we will restrict our attention to particular but expressive ones (and we will address later the difficulty concerning k).

Definition 1: $\exists MS(\mathcal{P})$ **sentences**

We let $\mathcal{P}$ be a set of MS graph properties consisting of the properties defined by the atomic formulas and of basic properties such as $Link(X_1, X_2)$, $Path(X_1, X_2)$, $Conn(X_1)$. We let $\{X_1, ..., X_n\}$ be a set of set variables. A *Boolean set term* is a term written with these variables, the operations $\cap$, $\cup$ and complementation. For example, $S = X_1 \cup \overline{X_3}$. A $\mathcal{P}$*-atomic formula* is a formula of the form $P(S_1, ..., S_m)$ where $S_1, ..., S_m$ are Boolean set terms and P belongs to $\mathcal{P}$. An $\exists MS(\mathcal{P})$ sentence is a sentence of the form $\exists X_1, ..., X_n.\varphi$

where φ is a positive Boolean combination of $\mathcal{P}$-atomic formulas. Note that this definition depends on a set $\mathcal{P}$ that we leave "extensible", according to the needs.

Examples 2: We now give some examples of properties of simple undirected graphs expressible by $\exists MS(\mathcal{P})$ sentences.

(1) The property of *p-vertex colorability* is expressed by the sentence :

$$\exists X_1, ..., X_p \ .(Part(X_1, ..., X_p) \wedge St(X_1) \wedge ... \wedge St(X_p))$$

where $Part(X_1, ..., X_p)$ expresses that $X_1, ..., X_p$ define a partition of the vertex set and $St(X_i)$ expresses that X_i is *stable*, i.e., that the induced graph $G[X_i]$ has no edge. A p-vertex coloring defined by $X_1, ..., X_p$ is *acyclic* if furthermore, each induced graph $G[X_i \cup X_j]$ is acyclic (i.e., is a forest). The existence of an acyclic p-coloring for G (we will say that G is *p-AC-colorable*) is expressed by:

$$\begin{array}{c}\exists X_1, ..., X_p \ .(Part(X_1, ..., X_p) \wedge St(X_1) \wedge \\ ... \wedge St(X_p) \wedge ... \wedge NoCycle(X_i \cup X_j) \wedge ...)\end{array}$$

with one formula $NoCycle(X_i \cup X_j)$ for all i, j with $1 \leq i < j \leq p$.

(2) *Minor inclusion.* Let H be a simple, loop-free and undirected graph with vertex set $\{v_1, ..., v_p\}$. A graph G contains H as a minor if and only if it satisfies the sentence :

$$\begin{array}{c}\exists X_1, ..., X_p \ .(Disjoint(X_1, ..., X_p) \wedge Conn(X_1) \wedge ... \\ \wedge Conn(X_p) \wedge ... \wedge Link(X_i, X_j) \wedge ...)\end{array}$$

where $Disjoint(X_1, ..., X_p)$ expresses that $X_1, ..., X_p$ are pairwise disjoint; there is one formula $Link(X_i, X_j)$ for every edge of H that links v_i and v_j.

(3) *Perfect graphs.* A (simple, loop-free and undirected) graph G is *perfect* if the chromatic number of each induced subgraph H is equal to the maximum size of a clique in H. This definition is not monadic second-order expressible (because the fact that two sets have equal cardinalities is not) but the characterization established by Chudnovsky et al. [3] in terms of excluded holes and antiholes is. A *hole* is an induced cycle of odd length at least 5 and an *antihole* is the edge-complement of a hole. A graph has a hole if and only if it satisfies the following sentence:

$$\begin{array}{c}\exists X, Y, Z, U, V.(Disjoint(X, Y, Z, U, V) \wedge edg(Z, U) \wedge edg(U, V) \wedge \\ \neg edg(Z, V) \wedge \deg_2(X, Z \cup Y) \wedge \deg_0(X, U \cup V) \wedge \\ \deg_2(Y, X \cup V) \wedge \deg_0(Y, U \cup Z) \wedge \deg_2(V, U \cup Y) \wedge \deg_2(Z, U \cup X))\end{array}$$

where $\deg_0(X, Y)$ means that $X \cap Y = \emptyset$, X is stable and not empty and there is no edge between X and Y; the property $\deg_2(X, Y)$ means that $X \cap Y = \emptyset$, X is stable and not empty and every vertex in X has exactly 2 neighbours in Y. For every term $t \in T(F_k)$, one can construct (easily) a term $\overline{t} \in T(F_{2k})$ that defines the edge complement of the graph $G(t)$ ([10]). We obtain that $G(t)$ is perfect if and only if the F_{2k}-automaton for holes rejects both t and $\overline{t}$. The algorithm of [2] can test if a graph is perfect in time $O(n^9)$ (n is the number of vertices). From the above logical expression of holes, we get a fixed-parameter *linear* algorithm for testing perfectness (with clique-width or even tree-width as parameter).

(4) *Constrained domination* and other problems. Let $P(X_1) \in \mathcal{P}$. The sentence $\exists X.(P(X) \wedge Dom(\overline{X}, X))$ expresses that there exists a set X satisfying property P that *dominates all other vertices*. Many vertex *partitionning problems* considered in [23] can be expressed by $\exists MS(\mathcal{P})$ sentences in similar ways.

From $\exists MS(\mathcal{P})$ sentences to automata

We review the main steps of the inductive construction of an automaton associated with a sentence of $\exists MS(\mathcal{P})$. (See [4] for automata on terms). We first consider the $\mathcal{P}$-atomic formulas. We assume that for each property $P(X_1, ..., X_m)$ of $\mathcal{P}$ and each k, we have constructed a finite automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ that accepts the set of terms $L_{P(X_1,...,X_m),k}$. Actually, these automata depend on k in a uniform way (see the end of this section for the use of this observation).

Claim 3 : For set terms $S_1, ..., S_m$ over $\{X_1, ..., X_n\}$, the set of terms $L_{P(S_1,...,S_m),(X_1,...,X_n),k}$ is $h^{-1}(L_{P(X_1,...,X_m),k})$ where h is an *alphabetic homomorphism*: $T(F_k^{(n)}) \to T(F_k^{(m)})$ that replaces each constant symbol $(\mathbf{a}, w)$ for $w \in \{0,1\}^n$ by $(\mathbf{a}, w')$ for some $w' \in \{0,1\}^m$ and does not modify the nonnullary function symbols.

We only give an example: consider a property $P(X_1)$, $n = 3$ and $S = X_1 \cup \overline{X_3}$. Then $L_{P(S),(X_1,X_2,X_3),k} = h^{-1}(L_{P(X_1),k})$ where, for every $x = 0, 1$:

$$h(1x0) = h(1x1) = h(0x0) = 1 \text{ and } h(0x1) = 0,$$

i.e., $h(x_1, x_2, x_3) = x_1 \vee \neg x_3$, hence h encodes the set term S in a natural way. The subscript (X_1, X_2, X_3) in $L_{P(S),(X_1,X_2,X_3),k}$ indicates that, although $P(S)$ depends only on X_1 and X_3, the set of terms is defined as if $P(S)$ depended on X_1, X_2 and X_3.

From an automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ that accepts $L_{P(X_1,...,X_m),k}$ one gets an automaton $\mathcal{A}_{P(S_1,...,S_m),(X_1,...,X_n),k}$ with same number of states (but more transitions in many cases) that accepts $L_{P(S_1,...,S_m),(X_1,...,X_n),k}$. If $\mathcal{A}_{P(X_1,...,X_m),k}$ is deterministic, then the automaton $\mathcal{A}_{P(S_1,...,S_m),(X_1,...,X_n),k}$ is also deterministic. This claim can also be used if the terms $S_1, ..., S_m$ are just variables, say X_{i_1},..., X_{i_m}, hence for handling a substitution of variables.

Claim 4 : If φ is a positive Boolean combination of $\mathcal{P}$-atomic formulas $\alpha_1, ..., \alpha_d$ for which we have constructed complete non-deterministic (resp. deterministic) automata $\mathcal{A}_1, ..., \mathcal{A}_d$ with respectively $N_1, ..., N_d$ states, one can construct a complete *product* non-deterministic (resp. deterministic) automaton for φ with $N_1 \times ... \times N_d$ states (or less after deletion of useless states).

Claim 5 : If θ is the sentence $\exists X_1, ..., X_n.\varphi$, and we have constructed an automaton $\mathcal{A}$ recognizing $L_{\varphi(X_1,...,X_n),k}$, we can obtain one recognizing $L_{\theta,k}$, with the same number of states by applying the mapping pr that deletes the sequences of Booleans from the constant symbols of $F_k^{(n)}$. The automaton for θ is not deterministic in general, even if $\mathcal{A}$ is. If $\mathcal{A}$ is deterministic, this construction defines an automaton with 2^n transitions associated with each constant symbol $\mathbf{a}$. Hence, the values of n should not be too large.

By these claims, one can construct for every k and every sentence φ in $\exists MS(\mathcal{P})$ a nondeterministic automaton $\mathcal{A}_{\varphi,k}$ that accepts the regular set of terms $L_{P,k} = L_{\varphi,k}$ where

P is the property expressed by φ, provided the automata for the atomic formulas and the properties of $\mathcal{P}$ are known.

Automata for the atomic and basic formulas.

We cannot detail all constructions, but we consider as an example the automaton $\mathcal{A} := \mathcal{A}_{edg(X_1,X_2),k}$. Its set of states is S consisting of $Ok, Error, 0, a(1), a(2), ab$ for all $a, b \in [k]$, $a \neq b$. It has $k^2 + k + 3$ states. Their meanings are described in Table 1. This table shows for each state s the property P_s that it encodes: s is the state reached by the automaton after reading a term t in $T(F_k^{(2)})$ if and only if P_s holds for this term. In its description, (V_1, V_2) is the pair of sets of vertices of the graph $G(pr(t))$ (that we denote more simply by $G(t)$) encoded by the Boolean components of the constants occurring in t. Table 2 specifies the transitions. All transitions not listed go to $Error$. Ok is the accepting state.

State s	Property P_s
0	$V_1 = V_2 = \emptyset$
$a(1)$	$V_1 = \{v\}, V_2 = \emptyset, \pi_{G(t)}(v) = a$
$a(2)$	$V_1 = \emptyset, V_2 = \{v\}, \pi_{G(t)}(v) = a$
Ok	$V_1 = \{v_1\}, V_2 = \{v_2\}$, $(v_1, v_2) \in edg_{G(t)}$
ab	$V_1 = \{v_1\}, V_2 = \{v_2\}$, $v_1 \neq v_2$, $\pi_{G(t)}(v_1) = a$, $\pi_{G(t)}(v_2) = b$ and $(v_1, v_2) \notin edg_{G(t)}$
$Error$	All other cases

Table 1 Meanings of the states of $\mathcal{A}$.

Transition rules	Conditions
$(\mathbf{a}, 00) \to 0$	
$(\mathbf{a}, 10) \to a(1)$	
$(\mathbf{a}, 01) \to a(2)$	
$(\mathbf{a}, 11) \to Error$	
$relab_h[s] \to s$	$s \in \{0, Ok\}$
$relab_h[a(i)] \to h(a)(i)$	$i \in \{1, 2\}$
$relab_h[ab] \to cd$	$c = h(a), d = h(b), c \neq d$
$\overrightarrow{add}_{a,b}[s] \to s$	$s \neq ab$
$\overrightarrow{add}_{a,b}[ab] \to Ok$	
$\oplus[a(1), b(2)] \to ab$	$a \neq b$
$\oplus[b(2), a(1)] \to ab$	
$\oplus[a(2), b(1)] \to ba$	
$\oplus[b(1), a(2)] \to ba$	
$\oplus[s, 0] \to s$	$s \in S$
$\oplus[0, s] \to s$	

Table 2 The transition rules of $\mathcal{A}$.

We have one automaton for each k, but all these automata have a same concise description. (If we let C be the set of positive integers, we can consider that Table 2 specifies a unique automaton with infinitely many states. We will use this observation when discussing below *fly-automata*.)

Theorem 6 : Let $\mathcal{P}$ be a set of basic graph properties. For each $P(Y_1, ..., Y_m) \in \mathcal{P}$ and for each k, let a deterministic automaton $\mathcal{A}_{P(Y_1,...,Y_m),k}$ with $N(P,k)$ states be already specified. For every sentence θ of the form $\exists X_1, ..., X_n.\varphi$ where φ is a Boolean combination of $\mathcal{P}$-atomic formulas $\alpha_1, ..., \alpha_d$, a nondeterministic automaton $\mathcal{A}_{\theta,k}$ (over the signature F_k because θ has no free variables) having at most $N := N_1 \times ... \times N_d$ states can be constructed where $N_i := N(P_i, k)$ and P_i is the property used to define α_i. □

A complete and deterministic automaton over F_k (where we use only the elementary relabellings $relab_{a\to b}$) with N states, has $k + 2k(k-1).N + N^2$ transitions. The automaton $\mathcal{A}_{\theta,k}$ has thus $k.2^n + 2k(k-1).N + N^2$ transitions. Its nondeterministic transitions are only associated with the constant symbols. In the following theorem, we let m be an upper-bound to the time necessary to determine the output state of a deterministic transition or the i-th output state of a nondeterministic one. With these hypotheses and notation:

Theorem 7 : For every term t in $T(F_k)$, one can decide in time $m.(2^n + N^2).\mid t \mid$ if the graph $G(t)$ satisfies θ.

Proof : We use a bottom-up computation on t to determine at each node u of its syntactic tree the set of states that can occur at u. For the q occurrences of constants symbols, this takes total time at most $m.2^n.q$. For the occurrences of unary and binary symbols, this takes total time at most $m.N^2.(\mid t \mid -q)$.□

Note that we do not determinize the automaton $\mathcal{A}_{\theta,k}$. We only compute the transitions of $det(\mathcal{A}_{\theta,k})$, the determinized automaton of $\mathcal{A}_{\theta,k}$, that are needed for checking a given term.

Some basic graph properties and their automata.

We classify the atomic formulas and some "basic" graph properties (to be included in $\mathcal{P}$) in terms of the numbers of states $N(k)$ of deterministic F_k-automata that check them.

Polynomial-sized automata: The automata for $X_1 \subseteq X_2$, $X_1 = \emptyset$, $Part(X_1, ..., X_p)$ and $Disjoint(X_1, ..., X_p)$ have 2 states, the one for $Sgl(X_1)$ has 3 states. For $edg(X_1, X_2)$, we have defined above an F_k-automaton with $k^2 + k + 3$ states. The F_k-automaton for the property that X_1 has at most p elements has $p + 2$ states.

Single-exponential sized automata: The F_k-automaton for $St(X_1)$ has $2^k + 1$ states. Those for $Link(X_1, X_2)$ and $Dom(X_1, X_2)$ have $2^{2k} + 1$ states.

For $Path(X_1, X_2)$, we can constructed a (non-minimal) F_k-automaton with less than 2^{k^2+2} states. For the property *maximum degree at most p,* we can construct an F_k-automaton with $2^{k^2 p \log(p)}$ states.

Double-exponential sized automata: For connectedness, we can build an F_k-automata with about 2^{2^k} states and the unique minimal F_k-automaton has more than $2^{2^{k/2}}$ states. However, if we have a upperbound p to the degree of the graphs to be checked, then an F_k-automaton with $2^{p.k^2}$ states suffices. For the property of being a forest, we can construct an F_k-automaton with $2^{2^{O(k)}}$ states but we have no lower bound showing that a double exponential is necessary.

Fly-automata

A *fly-automaton* is an automaton (possibly not deterministic) whose transition rules are not listed in a table but are defined by finitely many clauses that we can call *transition meta-rules*. Table 2 shows such rules. It shows actually the meta-rules of a fly-automaton with infinitely many states (where we replace $[k]$ by the set of positive integers). Each time a transition is needed it is computed from the specifications of Table 2. On input t in $T(F_k)$, this infinite automaton only uses the rules concerning the states of the set S defined above (see Table 1).

The finite F_k-automata described by Table 2 have $O(k^4)$ transitions, which makes difficult to compile their rules in a table unless k is small. This is even impossible for automata like the ones for connectedness that have $2^{2^{\Theta(k)}}$ states. Their tables cannot be constructed, even for small values of k. Hence, using fly-automata is necessary in such cases.

The automata currently used in compilation are "small" (typically, they have to recognize the key words of a programming language) whereas the input words (programs) are much larger. In the present case, the situation is to the opposite: the automata are huge and the input terms are "small" (typically, terms of size 200 to define graphs with 50 vertices). But since these automata have concise descriptions, instead of trying to *compile* them, we propose to *interpret* them, that is to compute only their transitions that are needed for particular input terms.

It is clear that the constructions of Claims 3 and 4 can be used for fly-automata. For example, the meta-rules for two automata $\mathcal{A}$ and $\mathcal{B}$ can be combined to form those of their product. The algorithm of Theorem 7 that checks if a term is accepted by a nondeterministic automaton without determinizing it is applicable to a nondeterministic fly-automaton such that finitely many transitions are possible at each node.

Experiments have been conducted by I. Durand with her software Autowrite that implements automata on terms [13]. Here are some results concerning colorability and acyclic colorability. Grünbaum has given an example of a 3-colorable planar graph with 6 vertices (the clique K_6 minus the 3 edges of a perfect matching) that is not 4-AC-colorable but is 5-AC-colorable. These facts have been verified in a few seconds by using a term in $T(F_3)$ of size 15 that defines this graph and in 94 minutes by using a term in $T(F_5)$ of size 21. The Petersen graph (10 vertices, 15 edges) is 3-colorable, not 3-AC-colorable, but it is 4-AC-colorable. This last fact has been verified in 17 minutes on a term in $T(F_7)$ that defines this graph. The corresponding automata are too large to be constructible.

4 Edge set quantifications

We will now consider graphs that can have multiple edges. Because of the chosen representation of graphs, the *MS* properties cannot take into account the multiplicity of edges: that a pair of vertices (x, y) belongs to edg_G does not tell us the exact number of edges from x to y. We define another representation to remedy this drawback. The *incidence graph* of an undirected graph G is the simple directed bipartite graph $Inc(G) := \langle V_G \cup E_G, in_G \rangle$ where in_G is the set of pairs (e, x) such that e belongs to the set of edges E_G and x is an end vertex of e. We no longer use the convention that an undirected edge is a pair of opposite directed edges, and we use the simpler notation in_G instead of $edg_{Inc(G)}$. If G is directed, we define $Inc(G) := \langle V_G \cup E_G, in_{1G}, in_{2G} \rangle$ where in_{1G} (resp. in_{2G}) is the set of pairs (e, x) such that $e \in E_G$ and x is the tail vertex of e (resp. its head vertex). Hence, $Inc(G)$ is directed and bipartite with two types of edges. We will denote by $\lceil G \rceil$ the graph $Inc(G)$ considered as a relational structure, either over $\{in\}$ or over $\{in_1, in_2\}$.

A graph property $P(X_1, ..., X_n, Y_1, ..., Y_m)$, where $X_1, ..., X_n$ denote sets of vertices and $Y_1, ..., Y_m$ denote sets of edges, is an *MS_2 graph property* if there exists an MS formula $\varphi(X_1, ..., X_n, Y_1, ..., Y_m)$, such that, for every graph G, for all sets of vertices $X_1, ..., X_n$ and for all sets of edges $Y_1, ..., Y_m$, we have:

$$\lceil G \rceil \models \varphi(X_1, ..., X_n, Y_1, ..., Y_m)$$
$$\text{if and only if } \; P(X_1, ..., X_n, Y_1, ..., Y_m) \text{ is true in } G.$$

The property that a simple undirected graph has at least 3 vertices and a Hamiltonian cycle is an MS_2-property that is not MS (see [6], Chapter 5). Hence, using $\lceil G \rceil$ instead of $\lfloor G \rfloor$ improves, also for simple graphs, the expressive power of monadic second-order logic. The appropriate parameter in the FPT algorithms that check MS_2 graph properties is not clique-width but tree-width.

For each k, there is a finite set H_k of graph operations such that a graph has tree-width at most k if and only if it is defined by a term over H_k. Linear time algorithms can convert tree-decompositions (the well-known definition is recalled in the next section) into terms and vice-versa. For every *MS_2 graph property* expressed by an MS sentence φ and every integer k, one can construct a finite automata that recognizes the set of terms in $T(H_k)$ that define graphs G such that $\lceil G \rceil \models \varphi$, i.e., that satisfy that property. However, since H_k uses the operation of *parallel-composition* (denoted by $//$) that combines graphs by fusing vertices instead of the disjoint union $\oplus$, the corresponding automata are much more complicated than those constructed above. This observation motivates the introduction of a special type of tree-decomposition, whence of a variant of tree-width, that lacking of a better term, we call *special tree-width*. The algebraic representation of the corresponding *special tree-decompositions* need not use parallel-composition.

5 Special tree-width

In order to simplify the presentation, we will only consider undirected graphs. However, the definitions and results extend easily to directed graphs. Our definition is based on the operations that define clique-width. We will use the operations $add_{a,b}$ instead of the operations $\overrightarrow{add}_{a,b}$. In order to define *graphs with multiple edges*, we will change the interpretation of the operation $add_{a,b}$ in the following way: if in a graph G there is already an edge between a vertex x labelled by a and a vertex y labelled by b, then the operation $add_{a,b}$ applied to G adds another edge between x and y. The corresponding notion of clique-width for graphs with multiple edges is studied in [7]. However, we will use here this feature in a restricted situation.

Definition 8: *Special terms*

We will use the graph operations that define clique-width (cf. Section 2). The labels of vertices will be taken from the sets $[k]_\perp := [k] \cup \{\perp\}$ instead of $[k]$ and the corresponding sets of operations will be denoted by $F_{k,\perp}$. (The label $\perp$ will be used as a default label.) We (still) denote by $\pi(G)$ the set of labels of the vertices of a graph G and by $\pi_1(G)$ the subset of those that label a single vertex of G. If $t \in T(F_{k,\perp})$, then $\pi(t)$ denotes $\pi(G(t))$ and $\pi_1(t)$ denotes $\pi_1(G(t))$.

A term t in $T(F_{k,\perp})$ is a *special term* if it satisfies the following conditions:

1) $\pi(t') - \pi_1(t') \subseteq \{\perp\}$ for every subterm t' of t (we consider t as one of its subterms),
2) if $t_1 \oplus t_2$ is a subterm of t, then $\pi(t_1) \cap \pi(t_2) \subseteq \{\perp\}$,

3) for every relabelling $relab_h$ occurring in t, we have $h(\bot) = \bot$,
4) for every operation $add_{a,b}$ that occurs in t, we have $a \neq \bot$ and $b \neq \bot$,
5) the constant symbol $\bot$ has no occurrence in t.

We denote by $SpT(F_{k,\bot})$ the sets of special terms in $T(F_{k,\bot})$. The *special tree-width* of a graph G, denoted by $sptwd(G)$, is the least integer k such that $G = G(t)$ for some term t in $SpT(F_{k+1,\bot})$. The comparison with tree-width will justify the "+1" in the definition. The special tree-width of a graph consisting of isolated vertices is 0. Since the sets $\pi(t)$ and $\pi_1(t)$ are computable inductively on the structure of a term t, the sets $SpT(F_{k+1,\bot})$ are regular.

A graph defined by a special term in $SpT(F_{k,\bot})$ has at most one vertex labelled by each a in $[k]$ and possibly several vertices labelled by $\bot$. No new edge can be added between vertices such that one of them is labelled by $\bot$. These vertices are somehow "terminated". Furthermore, each occurrence of an operation $add_{a,b}$ adds at most one edge (by Conditions 1) and 4)). It may add no edge if the argument graph has no vertex labelled by a or no vertex labelled by b. We will say that such an occurrence is *useful* if it adds an edge. (Occurrences that are not useful can be deleted, which gives a smaller *reduced* term defining the same graph.) The graph $G(t)$ defined by a special term t can be constructed with vertex set $Occ_0(t)$, the set of occurrences of constant symbols in t, and edge set $Occ_1(t)$, the set of useful occurrences of edge addition operations. This remark will be used in Section 6.

Example: Trees have special tree-width 1. An undirected tree with one distinguished node called its *root*, is labelled as follows: the root is labelled by 1, all other nodes by $\bot$. Let T_1, T_2 be two such trees, defined by terms $t_1, t_2 \in SpT(F_{2,\bot})$. Then, we let $T := T_1 \ltimes T_2$ be defined by the term

$$t := relab_{2 \longrightarrow \bot}\ (add_{1,2}(t_1 \oplus relab_{1 \longrightarrow 2}(t_2))) \in SpT(F_{2,\bot}).$$

This tree is built as the disjoint union of the trees T_1 and T_2 augmented with an undirected edge between their roots, and the root of T is defined as that of T_1. Every rooted and undirected tree is generated by $\ltimes$ from the trees reduced to isolated roots, that are defined (up to isomorphism) by the constant symbol $\mathbf{1}$. Hence, every rooted and undirected tree is defined by a term in $SpT(F_{2,\bot})$. One can forget the root by applying the operation $relab_{1 \longrightarrow \bot}$.□

We now consider tree-decompositions. A rooted and directed tree T is directed from the root towards the leaves.

Definition 9: A *special tree-decomposition* of a graph G is a pair (T, f) such that T is a rooted and directed tree with set of nodes N_T and $f : N_T \longrightarrow \mathcal{P}(V_G)$ is a mapping such that:
1) Every vertex of G belongs to $f(u)$ for some u in N_T.
2) Every edge has its ends in $f(u)$ for some u in N_T.
3) For each vertex x, the set $f^{-1}(x) := \{u \in N_T \mid x \in f(u)\}$ is a directed path in T.

Condition 3) characterizes special tree-decompositions. The *width* of a decomposition (T, f) is the the maximal cardinality minus 1 of a *box*, i.e. of a set $f(u)$. A *path-decomposition* is defined as a tree-decomposition such that T is a directed path (hence it is special). The *tree-width* $twd(G)$ (the *path-width* $pwd(G)$) of a graph G is the minimal width of a tree-decomposition (a path-decomposition) of this graph. It is known from [5,10] that a set of

simple graphs, directed or not, that has bounded tree-width has bounded clique-width: if G undirected has tree-width k, then it has clique-width at most 3.2^{k-1} and in some cases, more that $2^{k/2}$. However, its clique-width is at most $pwd(G)+2$.

Proposition 10: The special tree-width of a graph is the minimal width of a special tree-decomposition of this graph. There are linear-time algorithms for converting a term t in $SpT(F_{k+1,\perp})$ into a special tree-decomposition of width k of the graph $G(t)$ and vice-versa.

Proposition 11: For every graph G we have:
(1) $twd(G) \leq sptwd(G) \leq pwd(G)$,
(2) $cwd(G) \leq sptwd(G)+2$.

These facts are clear from Proposition 10 and the definitions. Note that clique-width behaves with respect to special tree-width exactly as with respect to path-width, and without the exponential increase. We will denote by $STWD(\leq k)$ the class of undirected graphs of special tree-width at most k.

Proposition 12: For each k, the class $STWD(\leq k)$ is closed under the following transformations:
1) Removal of vertices and edges,
2) addition of edges parallel to existing edges,
3) smoothing vertices of degree 2. □

Smoothing a vertex of degree 2 means contracting any one of its two incident edges. For the case of directed graphs (see [7]), reversals of edge directions also preserve special tree-width, whereas they do not preserve clique-width. It follows from items 1) and 3) of Proposition 12 that the class $STWD(\leq k)$ is closed under taking *topological minors* ([11]). It is not closed under taking minors as we will see in Proposition 16. In the following proposition, $pwd(L)$ denotes the least upper bound of the path-widths of the graphs in a set L and similarly for the other notions of width.

Proposition 13: The class of graphs of tree-width 2 has unbounded special tree-width. For every set of graphs L:

$$pwd(L) < \infty \Longrightarrow sptwd(L) < \infty \Longrightarrow twd(L) < \infty \text{ and}$$

$$sptwd(L) < \infty \Longrightarrow cwd(L) < \infty,$$

whereas the converse implications do not hold.□

Proof: We will use the following claim (where $G \otimes *$ is G augmented with a new vertex $*$ and edges between it and all vertices of G):

For every graph G, the special tree-width of $G \otimes *$ is equal to its path-width.

For proving the first assertion, we assume that every graph of tree-width 2 has special tree-width at most k. If T is any tree, then $T \otimes *$ has tree-width at most 2, hence special tree-width at most k, and path-width at most k by the claim. It follows that T, since it is a subgraph of $T \otimes *$, has path-width at most k, but trees have unbounded path-width ([11]), which gives a contradiction.

The implications follow from Proposition 11. Trees have special tree-width at most 1 and unbounded path-width. Graphs of tree-width 2 have unbounded special tree-width, hence the opposite implications are false. The converse of $sptwd(L) < \infty \Longrightarrow cwd(L) < \infty$ is false if L the set of cliques because it is of maximal clique-width 2 and of unbounded tree-width and special tree-width. □

Definition 14: A *tree-partition* of a graph G is a pair (T, f) such that T is a rooted tree with set of nodes N_T and $f : N_T \longrightarrow \mathcal{P}(V_G)$ is a mapping such that:

1) Every vertex of G belongs to $f(u)$ for a unique node u of T,
2) Every edge has its two ends in some box or in two boxes $f(u)$ and $f(v)$ such that v is the father of u.

The *width* of (T, f) is defined as the maximal cardinality of a box, (no -1 here !), and the *tree-partition-width* (also called *strong tree-width*) of a graph G is the minimal width of its tree-partitions. We denote it by $tpwd(G)$. The *wheels*, i.e., the graphs $C_n \otimes *$ where C_n is the undirected cycle with n vertices ($n \geq 3$) have path-width (and special tree-width) 3 but unbounded tree-partition width (see [1, 24]). $MaxDeg(G)$ denotes the maximum degree of a graph G. The proof of the following proposition uses results from [24].

Proposition 15: For every graph G :

1) $sptwd(G) \leq 2.tpwd(G) - 1$,
2) $sptwd(G) \leq 20.(twd(G) + 1).MaxDeg(G)$.

A set of graphs of bounded degree has bounded special-tree-width if and only if it has bounded tree-width.□

This result suggests a question:

Which conditions on a set of graphs, other than bounded degree, imply that it has bounded tree-width if and only if it has bounded special tree-width?

Planarity does not since the graphs of tree-width at most 2 are planar but of unbounded special tree-width. From this case, we can see that conditions like excluding a fixed graph as minor or being uniformly k-sparse for some k do not either. All these conditions however, imply that, for simple graphs, bounded tree-width is equivalent to bounded clique-width (see [6], Chapter 2).

Proposition 16: Every graph of tree-width k is obtained by edge contractions from a graph of special tree-width at most $2k + 1$. The class of graphs of special tree-width at most k is not closed under taking minors for any $k \geq 5$.

Proposition 17: The special tree-width of a graph is the maximal special tree-width of its connected components. It is at most one plus the maximal special tree-width of its biconnected components. This upper bound is tight.

Open question 18: *The parsing problem*: Does there exist fixed functions f and g and an approximation algorithm able to do the following in time $O(n^{g(k)})$, where n is the number of vertices of the given graph:

Given a simple graph G and an integer k, either it answers (correctly) that G has special tree-width more than k, or it outputs a special term witnessing that its special tree-width is at most $f(k)$?

Stronger requirements would be that $f(k) = k$, giving an exact algorithm and/or the computation time $O(g(k).n^c)$ for some fixed c instead of $O(n^{g(k)})$. Since by a result by Bodlaender (presented in detail in [12]) such an algorithm exists for tree-width, with $f(k) = k$ and $c = 1$, one can think that this algorithm can be adapted in order to find special tree-decompositions.

6 Automata for monadic second-order formulas with edge set quantifications

Our objective is to adapt the constructions of Section 3 to the model-checking of MS_2 graph properties for graphs defined by special terms. We will obtain fixed-parameter linear algorithms for graphs of bounded special tree-width given by the relevant terms or decompositions.

MS_2 formulas and the encoding of assignments

In order to use MS_2-formulas, i.e., monadic second-order formulas with edge set quantifications, we will represent a graph G by the relational structure $\lceil G \rceil := Inc(G)$ defined in Section 4. As in Section 3, we will use formulas written without first-order variables and universal quantifications. We will use the "standard" set variables $X_1, ..., X_n, ...$ for denoting sets of vertices and similarly the variables $Y_1, ..., Y_m, ...$ for denoting sets of edges. The atomic formulas are of the forms $edg(X_i, X_j)$, $in(Y_i, X_j)$ (graphs are undirected), and of course, $X_i \subseteq X_j$, $Y_i \subseteq Y_j$, $Z = \emptyset$, $Sgl(Z)$, where Z is X_i or Y_j. The meaning of $in(Y_i, X_j)$ is that Y_i and X_j are singletons, respectively $\{y\}$ and $\{x\}$ such that $(y, x) \in in_G$.

We now discuss the encoding of assignments in terms. Let t be a special term and $G(t)$ be the (concrete) graph it defines. Its vertex set is $Occ_0(t)$, the set of occurrences of constant symbols and it edge set is $Occ_1(t)$, the set of useful occurrences of edge addition operations (cf. Definition 8.) In order to encode $\{X_1, ..., X_n, Y_1, ..., Y_m\}$-assignments, we will use, the signatures $F^{(n,m)}_{k,\perp}$ obtained from $F^{(n)}_{k,\perp}$ by replacing every edge addition operation f by the unary operations (f, w), for all w in $\{0, 1\}^m$.

We will use the projections pr as in Claim 5 and the projections pr', that delete the Booleans in the unary operations (f, w). It is clear that a term $t \in T(F^{(n,m)}_{k,\perp})$ such that $pr(pr'(t))$ is a special term and the occurrences of edge addition operations in $pr(pr'(t))$ are all useful, defines a graph $G(pr(pr'(t))$ and an $\{X_1, ..., X_n, Y_1, ..., Y_m\}$-assignment γ such that $\gamma(X_i)$ is a set of vertices (for $i \in [n]$) and $\gamma(Y_j)$ is a set of edges (for $j \in [m]$).

We will denote by $RT(F^{(n,m)}_{k,\perp}) \subseteq SpT(F^{(n,m)}_{k,\perp})$ the set of *reduced terms*, defined as the set of special terms in which every occurrence of an edge addition operation is useful. Whether a term t in $T(F^{(n,m)}_{k,\perp})$ is in $RT(F^{(n,m)}_{k,\perp})$ or not does not depend on the Boolean components of its constant symbols and of its edge addition operations. In other words, $RT(F^{(n,m)}_{k,\perp}) = pr'^{-1}(pr^{-1}(RT(F_{k,\perp})))$.

Claim 19 : For each triple n, m, k of integers, the set $RT(F^{(n,m)}_{k,\perp})$ is regular and is recognized by a deterministic automaton with 2^k states.

For every MS_2 formula φ with free variables in $\{X_1, ..., X_n, Y_1, ..., Y_m\}$ and every k, we define $L_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}$ as the set of terms t in $RT(F^{(n,m)}_{k,\perp})$ such that $(\lceil G(pr(pr'(t)))\rceil, \gamma(t)) \models \varphi$ where $\gamma(t)$ denotes the $\{X_1, ..., X_n, Y_1, ..., Y_m\}$-assignment encoded by t. The language $L_{P(X_1,...,X_n,Y_1,...,Y_m),k}$ can be defined similarly for a graph property $P(X_1, ..., X_n, Y_1, ..., Y_m)$ independently of its logical expression. Note that we define here sets of reduced terms.

Theorem 20: For every MS_2 graph property $P(X_1, ..., X_n, Y_1, ..., Y_m)$ and every k, the language $L_{P(X_1,...,X_n,Y_1,...,Y_m),k}$ is regular and an automaton recognizing it can be constructed from an MS_2 formula that expresses P.

Proof: The construction is as for Theorem 6. At each step we restrict the defined sets so that they only contain reduced terms. For example, if φ is $\neg\theta$, we construct an automaton that recognizes $L_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k} = RT(F^{(n,m)}_{k,\perp}) - L_{\theta,(X_1,...,X_n,Y_1,...,Y_m),k}$. We do not simply take the complement.

Let us say a few words on the automata for the atomic formulas. Most of the constructions are straightforward from the definitions, as in Theorem 6. We only consider the atomic formulas $edg(X_1, X_2)$ and $in(Y_1, X_1)$.

An $F^{(2)}_{k,\perp}$-automaton $\mathcal{A}'$ for $edg(X_1, X_2)$, that is essentially the same as the automaton $\mathcal{A}$ of Theorem 6, can be constructed so as to work correctly on reduced terms. The automaton $\mathcal{A}_{edg(X_1,X_2),k}$ intended to define the set $L_{edg(X_1,X_2),k}$ is then obtained by a product with the one of Claim 19 that recognizes the set of reduced terms. Its number of states is thus $2^k.O(k^2)$ instead of $O(k^2)$. In the following remark, we discuss this difficulty.

We now construct an automaton $\mathcal{B}$ for $in(Y_1, X_1)$, intended to work on reduced terms. Its set of states is $S := \{0, Error, Ok\} \cup [k]$. Their meanings are described in Table 3, where W_1 denotes the value of the set variable Y_1. Its transitions not yielding $Error$ are in Table 4. As examples of transitions to $Error$ we have $\oplus[Ok, a] \to Error$ and $(add_{a,b}, 1)[Ok] \to Error$. The unique accepting state is Ok.

State s	Property P_s
0	$V_1 = W_1 = \emptyset$
a	$V_1 = \{v\}, W_1 = \emptyset, \pi_{G(t)}(v) = a$
Ok	$V_1 = \{v\}$, $W_1 = \{e\}$, $(e, v) \in in_{G(t)}$
$Error$	All other cases

Table 3 Meanings of the states of $\mathcal{B}$.

Remark 21: The above construction associates with each subformula $\theta(X_1, ..., X_n, Y_1, ..., Y_m)$ of the considered formula φ an automaton $\mathcal{A}_{\theta,(X_1,...,X_n,Y_1,...,Y_m),k}$ that recognizes only reduced terms. This means that each of these automata repeats the verification that the input term is reduced. One can actually postpone this verification to the very end.

Assume that for each atomic formula $\alpha(X_1, ..., X_n, Y_1, ..., Y_m)$, we have an automaton $\mathcal{B}_{\alpha,(X_1,...,X_n,Y_1,...,Y_m),k}$ such that

$$L_{\alpha,(X_1,...,X_n,Y_1,...,Y_m),k} = L(\mathcal{B}_{\alpha,(X_1,...,X_n,Y_1,...,Y_m),k}) \cap RT(F^{(n,m)}_{k,\perp}).$$

This means that $\mathcal{B}_{\alpha(X_1,...,X_n,Y_1,...,Y_m),k}$ is constructed so as to work correctly on reduced terms, and this is what we did above for $\mathcal{A}'$ and $\mathcal{B}$.

Let us build $\mathcal{B}_{\varphi(X_1,...,X_n,Y_1,...,Y_m),k}$ for all formulas φ by applying the general inductive construction described for Theorem 6 with, for the negation:

Transition rules	Conditions
$(\mathbf{a}, 0) \to 0$ $(\mathbf{a}, 1) \to a$	
$relab_h[0] \to 0$ $relab_h[Ok] \to Ok$ $relab_h[a] \to b$	 $b = h(a) \neq \perp$
$(add_{a,b}, 0)[s] \to s$ $(add_{a,b}, 1)[c] \to Ok$	all s $c \in \{a, b\}$
$\oplus[s, 0] \to s$ $\oplus[0, s] \to s$	all s

Table 4 The transition rules of $\mathcal{B}$.

$$L(\mathcal{B}_{\neg\theta,(X_1,...,X_n,Y_1,...,Y_m),k}) = T(F_{k,\perp}^{(n,m)}) - L(\mathcal{B}_{\theta,(X_1,...,X_n,Y_1,...,Y_m),k}).$$

At the end, for the input formula $\varphi(X_1, ..., X_n, Y_1, ..., Y_m)$, we make the restriction to reduced terms by defining $\mathcal{A}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}$ in such a way that:

$$L(\mathcal{A}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}) = L(\mathcal{B}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}) \cap RT(F_{k,\perp}^{(n,m)}).$$

Hence, we use only at the end the restriction to reduced terms. We claim that $L(\mathcal{A}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}) = L_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}$. This is true by the hypotheses on the automata $\mathcal{B}_\alpha$ associated with the atomic formulas and by the following observations:

if L, M, R, T, L', R' and T' are sets such that $L, M, R \subseteq T$ and $L', R' \subseteq T'$, and pr is a mapping from T' to T such that $T' = pr^{-1}(T)$ and $R' = pr^{-1}(R)$ then, $(L \cap R) \cap (M \cap R) = (L \cap M) \cap R$, $(L \cap R) \cup (M \cap R) = (L \cup M) \cap R$, $R - (L \cap R) = (T - L) \cap R$ and $pr(L' \cap R') \cap R = pr(L') \cap R$.

□

$\exists MS_2(\mathcal{P})$ sentences can be defined and fly-automata can be used to check them on graphs of bounded special tree-width.

Tree-width versus special tree-width

We now explain why the constructions of automata are easier for bounded special tree-width than for bounded tree-width.

Definition 22: *Special tree-width-terms.*
We let $H_{k,\perp}$ be the signature obtained from $F_{k,\perp}$ by replacing the operation $\oplus$ by $//$. This operation symbol will be interpreted as follows: for graphs G and H such that, as in Definition 8, $\pi(G) - \pi_1(G) \subseteq \{\perp\}$ and $\pi(H) - \pi_1(H) \subseteq \{\perp\}$, we define $G//H$ from $G \oplus H$ by fusing any two vertices having the same label $a \neq \perp$. A *special tree-width-term* is a term t in $T(H_{k,\perp})$ that satisfies conditions 1), 3), 4) and 5) of Definition 8. We denote by $TWT(H_{k,\perp})$ the set of these terms. Every graph is the value $G(t)$ of a term t in $TWT(H_{k,\perp})$ for some large enough k.

Proposition 23 ([6], Chapter 2): The tree-width of a graph is the least integer k such that this graph is the value of a term in $TWT(H_{k+1,\perp})$. There are linear-time algorithms for

converting a term t in $TWT(H_{k+1,\perp})$ into a tree-decomposition of width k of the graph $G(t)$ and vice-versa.

In view of the verification of MS_2 properties, let us consider the encoding of assignments in terms. Let $t \in TWT(H_{k,\perp})$ and $G = G(t)$. Its edges are in bijection with the set $Occ_1(t)$ defined as for special terms. However, its vertex set is isomorphic to a quotient of $Occ_0(t)$ by the equivalence relation $\approx$ expressing that x and y in $Occ_0(t)$ have a least common ancestor u that is an occurrence of // and that the associated vertices have the same label (different from $\perp$) in $G(t/u)$, the graph defined by the subterm of t issued from u. This means that x and y, because of a fusion occurring at u, yield the same vertex of G. Hence, we loose the nice bijection between vertices of $G(t)$ and particular occurrences of symbols in t. It follows that a set $X \subseteq Occ_0(t)$ represents correctly a set of vertices of $G(t)$ if and only it is saturated for $\approx$ (is a union of classes of this equivalence relation). The $H_{k,\perp}$-automaton analogous to $\mathcal{B}_{\varphi,(X_1,\ldots,X_n,Y_1,\ldots,Y_m),k}$ must check this saturation property, which increases substancially its number of states.

There is actually another possibility for representing vertices by occurrences of symbols in terms. Let us assume that all vertices of $G(t)$ are labelled by $\perp$ and that each vertex corresponds to a unique occurrence of an operation $relab_{a\longrightarrow\perp}$. Such occurrences, let us denote their set by $Occ_1^{vert}(t)$, can be chosen to represent the vertices. In this case, an edge will be represented by a node of t that is *below the nodes representing its ends*. This is not a difficulty for constructing an $H_{k,\perp}$-automaton for the atomic formula $in(Y_1, X_1)$ (like $\mathcal{B}$ in the proof of Theorem 20) having $k + 3$ states. However, the construction of an $H_{k,\perp}$-automata for $edg(X_1, X_2)$ is more complicated. It can be done directly or by the general construction: since $edg(X_1, X_2)$ is equivalent to $X_1 \neq X_2 \wedge \exists Y_1(in_1(Y_1, X_1) \wedge in_2(Y_1, X_2))$, the general construction produces an $H_{k,\perp}$-automaton with $2^{O(k^2)}$ states. (The factor k^2 is due to the use of a product of two automata with $k + 3$ states for the subformula $in_1(Y_1, X_1) \wedge in_2(Y_1, X_2)$, and the exponentiation is due to the determinization that is needed because of $\exists Y_1$). Furthermore, every deterministic $H_{k,\perp}$-automaton for $edg(X_1, X_2)$ *must have* at least $2^{k(k-1)}$ states ([6], Chapter 6). Hence, with this encoding of assignments, an atomic formula like $edg(X_1, X_2)$ needs already fairly "large" automata. This difficulty is avoided in special terms because they use $\oplus$ instead of //.

7 Conclusion

We have presented some tools intended to yield practically usable methods for the verification of *certain* monadic second-order graph properties for graphs of bounded tree-width or clique width. We have proposed to restrict the constructions of automata to the formulas of an appropriate fragment of monadic second-order logic and to use *fly automata* (a notion first presented in [8]). Although some experimental results are encouraging, these ideas have to be tested on more cases.

Special tree-width seems interesting on its own, but the construction of "small" automata has motivated its introduction. The corresponding parsing problem is open.

What about automata for graphs of bounded tree-width? We are presently working on a redundant representation of these graphs that equips terms in $TWT(H_{k,\perp})$ with additional labels. "Small" automata for $edg(X_1, X_2)$ whose transitions use these additional labels (as opposed to the operations of $H_{k,\perp}$) can then be constructed.

8 References

[1] H. Bodlaender, J. Engelfriet, Domino tree-width, *J. Algorithms* **24** (1997) 94-123.

[2] M. Chudnovsky *et al.*, Recognizing Berge graphs, *Combinatorica* **25** (2005) 143-186.

[3] M. Chudnovsky *et al.*, The strong perfect graph theorem, *Ann. Math.* **164** (2006) 51-229.

[4] H. Comon *et al.*, *Tree Automata Techniques and Applications*, On line for free at: http://tata.gforge.inria.fr/

[5] D. Corneil, U. Rotics, On the relationship between clique-width and tree-width. *SIAM J. Comput.* **34** (2005) 825-847.

[6] B. Courcelle, *Graph structure and monadic second-order logic*, book to be published by *Cambridge University Press*. Readable on: http://www.labri.fr/perso/courcell/Book/CourGGBook.pdf

[7] B. Courcelle, On the model-checking of monadic second-order formulas with edge set quantifications, May 2010, to appear in *Discrete Applied Mathematics*. Available from : http://hal.archives-ouvertes.fr/hal-00481735/fr/

[8] B. Courcelle, I. Durand, Verifying monadic second-order graph properties with tree automata, *3rd European Lisp Symposium*, May 2010, Lisbon, Informal proceedings edited by C. Rhodes, pp. 7-21. See: http://www.labri.fr/perso/courcell/ArticlesEnCours/BCDurandLISP.pdf

[9] B. Courcelle, J. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* **33** (2000) 125-150.

[10] B. Courcelle, S. Olariu, Upper bounds to the clique-width of graphs, *Discrete Applied Mathematics* **101** (2000) 77-114.

[11] R. Diestel, *Graph Theory*, 3rd edition, Springer, 2005.

[12] R. Downey, M. Fellows, *Parameterized complexity*, Springer-Verlag, 1999.

[13] I. Durand, Autowrite: A tool for term rewrite systems and tree automata, *Electronic Notes in Theoret. Comput. Sci.* **124** (2005) 29-49.

[14] J. Flum, M. Grohe, *Parametrized complexity theory*, Springer, 2006.

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

[16] R. Ganian, P. Hlineny, On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics* **158** (2010) 851-867

[17] R. Ganian, P. Hlineny, J.Obdrzalek, Better algorithms for satisfiability problems for formulas of bounded rank-width, 2010, arXiv:1006.5621v1[cs. DM]

[18] G. Gottlob, R. Pichler, F. Wei: Abduction with bounded tree-width: from theoretical tractability to practically efficient computation. Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, Chicago, AAAI Press, 2008, pp. 1541-1546

[19] G. Gottlob, R. Pichler, F. Wei: Monadic Datalog over finite structures with bounded tree-width, CoRR abs/0809.3140 (2008)

[20] P. Hlineny, S. Oum: Finding branch-decompositions and rank-decompo-sitions. *SIAM J. Comput.* **38** (2008) 1012-1032.

[21] L. Stockmeyer, A. Meyer, Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM* **49** (2002) 753-784.

[22] M. Weyer, Decidability of S1S and S2S. in *Automata, Logics, and Infinite Games: A Guide to Current Research*. Lecture Notes in Computer Science **2500,** Springer, 2002, pp. 207-230.

[23] M. Rao, MSOL partitioning problems on graphs of bounded tree-width and clique-width. *Theor. Comput. Sci.* **377** (2007) 260-267.

[24] D. Wood, On tree-partition-width, *European Journal of Combinatorics* **30** (2009) 1245-1253.