Automata for the verification of monadic second-order graph properties

Bruno Courcelle, Irène A. Durand

Automata for the verification of monadic second-order graph properties

Bruno Courcelle[1,*], Irène Durand

Labri (CNRS), University of Bordeaux, 351 Cours de la Libération, F-33405 Talence cedex, France

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 automata from monadic second-order sentences. These automata recognize the terms over fixed finite signatures that define graphs satisfying the given sentences. However, this construction produces automata of hyper-exponential sizes, and is thus impossible to use in practice in many cases. To overcome this difficulty, we propose to specify the transitions of automata by programs instead of tables. Such automata are called *fly-automata*. By using them, we can check certain monadic second-order graph properties with limited quantifier alternation depth, that are nevertheless interesting for Graph Theory. We give explicit constructions of automata relative to graphs of bounded clique-width, and we report on experiments.

Keywords: Graph algorithm, automaton, monadic-second order logic, clique-width, fixed-parameter tractability, fly-automaton.

1. Introduction

It is well-known from [5, 6, 10, 12, 16] 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 certain graph decompositions: tree-decompositions for tree-width as parameter and decompositions in complete bipartite graphs for clique-width as parameter. Both types of decompositions are formalized by terms over finite sets of graph operations. The proof uses also finite automata, constructed from the MS sentences that express the properties to check. These automata recognize the terms that define graphs satisfying the given sentences.

There are two difficulties for turning this result into a usable algorithm. The first one is the *parsing problem* consisting in constructing an appropriate decomposition of the given graph. The second difficulty is due to the enormous sizes of the automata constructed from MS sentences. To address the latter, we propose to use *fly-automata*, i.e., automata whose transitions are specified by programs and not compiled in (huge) tables. We also present some tools that limit the number of states of the constructed automata: we construct "small" automata associated

*Corresponding author

Email addresses: courcell@labri.fr (Bruno Courcelle), idurand@labri.fr (Irène Durand)

[1]Honorary member of Institut Universitaire de France

with some basic graph properties (and not with the atomic formulas) and we write formulas with *set terms* defined with the Boolean operations and the set variables.

In this article, we only consider the model-checking problem for monadic second-order sentences not using edge set quantifications. The relevant parameter is clique-width. Since for a class of graphs, bounded tree-width implies bounded clique-width, this approach also applies if tree-width is taken as parameter. Using tree-width as parameter allows to handle sentences written *with edge set quantifications*, but presents other difficulties (see [7]). Our objective is to implement the following theorem of [10] (we denote by F_k the finite set of graph operations that generates the graphs of clique-width at most k).

Theorem 1.

(1) *For every monadic second-order sentence φ and every integer k, one can construct a finite automaton recognizing the terms over F_k that denote graphs satisfying φ.*
(2) *Every monadic second-order graph property P can be checked in time $f_P(k)\cdot n^3$ for a simple directed or undirected graph with n vertices and of clique-width at most k.*

Assertion (1) gives an automaton that accepts or rejects a term in linear time in the size of the given term (a term over F_k for fixed k). For Assertion (2) we need to solve the parsing problem. Checking if a given graph has clique-width at most a given integer k is NP-complete if k is part of the input ([14]) but there exists a cubic approximation algorithm (see below Section 2.2). This algorithm takes time $g(k)\cdot n^3$ to construct a term of size $O(n)$ over $F_{h(k)}$ that denotes a given graph with n vertices and of clique-width at most k, where g and h are fixed functions. In other words the model-checking problem for monadic second-order logic on graphs is *fixed-parameter cubic* for a parameter consisting of a bound on the clique-width of the input graph and the MS sentence expressing the considered property.

Our only concern in this article is the construction of the automata of Assertion (1). They are constructed by induction on the structure of the input sentences (where universal quantifications are replaced by negations and existential quantifications). Those associated with the atomic formulas are easy to build and relatively "small". Products of automata are used for conjunction and disjunction. Complementation applied to deterministic automata is used for negation. A construction usually called "projection" is used for existential quantifications and introduces nondeterminism. It follows that determinization must be performed before each application of complementation. Since existential quantifications produce nondeterminism, quantifier alternation is the source of the hyper-exponential size of the constructed automata in the general case. This is actually unavoidable if one wants a construction taking as input arbitrary MS sentences (see, e.g., [17, 29, 31]). In order to overcome this difficulty, some authors focus their attention on particular problems instead of trying to implement the general theorem (see, e.g., [1, 22, 23, 18, 19]). We do not follow this route: we present some techniques that make the situation manageable for a large fragment of MS logic able to express interesting graph properties.

The article is organized as follows: Sections 2-4 review definitions about graphs, clique-width, automata and monadic second-order logic. Sections 4 and 5 constitute a tool box for the implementation of Theorem 1. Sections 5 and 6 detail the constructions of automata for the

atomic formulas and for some basic graph properties. By using new atomic formulas expressing these properties and *Boolean set terms* (that do not cost much in terms of sizes of automata), we can express significant graph properties without quantifier alternation. Some constructions of Section 6 are somewhat complicated, and we prove their correctness. Section 7 defines fly-automata and some constructions concerning them. Section 8 reports on experiments with fly-automata. Section 9 is a conclusion.

Detailed content

2. Terms, graphs and clique-width

2.1. Terms and graphs

Definition 2. *Terms and their syntactic trees*

A *functional signature* F is a set of *function symbols*, each being given together with a natural number called its *arity*: $\rho(f)$ denotes the arity of the symbol f. The set of *terms over* F is denoted by $T(F)$. A *language over* F is a subset of $T(F)$.

We now introduce some definitions relative to the internal structure of terms. A *position* of t is an occurrence of some symbol. We denote by $Pos(t)$ the set of positions of t and by $Pos(t, f)$ the set of occurrences in t of a symbol f. Hence $Pos(t) = \bigcup\{Pos(t, f) | f \in F\}$. The *size* $|t|$ of t is the cardinality of $Pos(t)$.

Terms will be written with commas and parentheses (that do not count in $|t|$). For example the term $t = f(g(h(a), b), c, g(g(b, c), c))$ has size 11. Positions will be designated by numbers corresponding to their ranks seen from left to right and starting at 1. In this example, $Pos(t, c) = \{6, 10, 11\}$. (They can be denoted in other ways, for instance by Dewey words, as in [4], our main reference for automata on terms).

The *syntactic tree* of a term t is a rooted, labelled and ordered tree. Its set of nodes is $Pos(t)$. Each node u is labelled by the symbol f such that $u \in Pos(t, f)$ and it has an ordered sequence of $\rho(f)$ sons. The root ($root_t$) is the first position, and the occurrences of the nullary symbols are the leaves. (The terminology of rooted trees will thus be applied to terms, via their syntactic trees.) The partial order $\preccurlyeq_t$ on $Pos(t)$ is defined such that $u \preccurlyeq_t v$ if and only if $u = v$ or v is a proper ancestor of u. (Positions are integers but this order is not the usual order on integers.) We denote by t/u the *subterm of* t *issued from a node* u. In the above example, $t/2 = g(h(a), b)$. The *context of* u *in* t consists of the function symbols that occur at positions not below u: formally, it is the unique term over F with a unique occurrence of a variable x and such that t is equal to the substitution in c of t/u for x. In the above example, the context of $u = 2$ in t is $f(x, c, g(g(b, c), c))$.

Let H be a finite signature (possibly $H = F$), and $h : H \to F$ be an arity preserving mapping, i.e., such that $\rho(h(f)) = \rho(f)$ for every $f \in H$. For every $t \in T(H)$, we let $h(t) \in T(F)$ be

the term obtained from t by replacing f by $h(f)$ at each of its occurrences. The mapping h on terms is called a *relabelling*.

Definition 3. *Graphs*

All graphs are finite and simple. They can have loops. A graph G is identified with the relational structure $\langle V_G, edg_G\rangle$ where edg_G is a binary relation representing adjacency: $(x,y) \in edg_G$ if and only if there is a directed edge from x to y; we write this $x \rightarrow_G y$ and we say that y is the *head* of this edge and x is its *tail*. This edge is a *loop* if $x = y$. An undirected graph is a directed graph G such that every edge has an opposite edge (i.e., edg_G is symmetric)[2], and then, we write $x -_G y$ if $x \rightarrow_G y$ and $y \rightarrow_G x$. We denote by $und(G)$ the undirected graph associated with G: $V_{und(G)} := V_G$ and $edg_G := edg_G \cup edg_G^{-1}$.

A *path* is a sequence of vertices such that two consecutive vertices are adjacent and no vertex occurs twice, except for the two ends. If its two ends are equal, this path is a *cycle*. A loop is a cycle. These notions do not depend on edge directions. In a directed graph, we have the restricted notions of directed paths and cycles. An *undirected path (or cycle)* in a directed graph is a path (or a cycle) in $und(G)$, i.e. its edges are traversed in any direction.

If $X \subseteq V_G$, we denote by $G[X]$ the induced subgraph of G with vertex set X, i.e., $G[X] := \langle X, edg_G \cap (X \times X)\rangle$.

In order to build graphs by means of graph operations, we use vertex labels. Let C be a set of labels called *port labels*. A *p-graph* (or *graph with ports*) is a triple $G = \langle V_G, edg_G, \pi_G\rangle$ where π_G is a mapping: $V_G \rightarrow C$. So, $\pi_G(x)$ is the label of x and if $\pi_G(x) = a$, we say that x is an *a-port*. We let G° be $\langle V_G, edg_G\rangle$, i.e., the corresponding graph without ports. If X is a set of vertices, then $\pi_G(X)$ is the set of its port labels. The set $\pi(G) := \pi_G(V_G)$ is the *type* of G. By using the integer 1 as default label, we make every nonempty graph into a p-graph of type $\{1\}$.

A p-graph G whose type $\pi(G)$ is included in a fixed finite set C is identified with the relational structure $\langle V_G, edg_G, (lab_{aG})_{a\in C}\rangle$ where lab_a is a unary relation and lab_{aG} is the set of a-ports of G. Since we only consider simple graphs, two graphs or p-graphs G and H are isomorphic if and only if the corresponding structures are isomorphic. In this article, we will always take C equal to $[k] := \{1, ..., k\}$ for $k \geq 2$.

The *edge complement* $\overline{G}$ of an undirected and loop-free p-graph G is defined as $\langle V_G, edg_{\overline{G}}, \pi_G\rangle$ where $edg_{\overline{G}}$ is the set of pairs (x,y) in $V_G \times V_G$ such that $x \neq y$ and $(x,y) \notin edg_G$.

2.2. Graph operations and clique-width

Definition 4. *Operations on p-graphs*

We let F_k be the following finite set of function symbols that define *operations* on the p-graphs of type included in $C := [k]$:

the binary symbol $\oplus$ denotes the union of two *disjoint*[3] p-graphs,

[2]This convention is inadequate if one uses edge set quantifications as in [7].

[3]If G and H are not disjoint, one can define $G \oplus H$ as $G \oplus H'$ where H' is isomorphic to H and disjoint from G. The resulting p-graph is defined up to isomorphism, hence as an *abstract graph*; see Chapter 2 of [9] for a study of the F_k-algebra of abstract labelled graphs. We say that a graph is *concrete* to stress that it is not defined up to isomorphism.

the unary symbol $relab_h$ denotes the *relabelling* that changes in the argument p-graph every port label a into $h(a)$ (where h is a mapping from C to C),

the unary symbol $\overrightarrow{add}_{a,b}$, for $a \neq b$, denotes the *edge-addition* that adds an edge from every a-port x to every b-port y (unless there is already an edge $x \to y$, this operation is idempotent),

for each $a \in C$, the nullary symbols $\mathbf{a}$ and $\mathbf{a}^\ell$ denote respectively an isolated a-port and an a-port with an incident loop,

and finally, the nullary symbol $\varnothing$ denotes the empty graph.

The operation $\oplus$ being associative, we will use infix notation without parentheses for it. The set $\{\mathbf{a}, \mathbf{a}^\ell \mid a \in C\}$ will be denoted by $\mathbf{C}$. The unary operation $relab_{\mathrm{Id}}$ (where Id is the identity: $C \to C$) is the identity operation.

For constructing undirected graphs, we will use the operation $add_{a,b}$ where $a < b$ (the set C is linearly ordered as it is of the form $[k]$) as an abbreviation of $\overrightarrow{add}_{a,b} \circ \overrightarrow{add}_{b,a}$. For constructing undirected graphs only, we will use the operations $add_{a,b}$ instead of $\overrightarrow{add}_{a,b}$, which yields the signature F_k^{u}.

Definition 5. *k-expressions and clique-width*

(a) Every term t in $T(F_k)$ (or in $T(F_k^{\mathrm{u}})$) is called a *k-expression*. It denotes a concrete p-graph, $cval(t)$, read, the *concrete value* of t, that we now define. We define $Pos_0(t)$ as the set occurrences in t of the symbols from $\mathbf{C}$ (the set of nullary symbols different from $\varnothing$). For each node u of t, we define a concrete p-graph $cval(t)/u$, whose vertex set is $\{x \in Pos_0(t) \mid x \preccurlyeq_t u\}$, i.e., the set of leaves of t below u that are not occurrences of $\varnothing$. The definition of $cval(t)/u$ is by bottom-up induction on u.

If u is an occurrence of $\varnothing$, then $cval(t)/u := \varnothing$.

if u is an occurrence of $\mathbf{a}$, then $cval(t)/u$ is the a-port u (its unique vertex is specified as u),

if u is an occurrence of $\mathbf{a}^\ell$, then $cval(t)/u$ is the a-port u with an incident loop,

if u is an occurrence of $\oplus$ with sons u_1 and u_2, then $cval(t)/u := cval(t)/u_1 \oplus cval(t)/u_2$, (note that $cval(t)/u_1$ and $cval(t)/u_2$ are disjoint p-graphs),

if u is an occurrence of $relab_h$ with son u_1, then $cval(t)/u := relab_h(cval(t)/u_1)$,

if u is an occurrence of $\overrightarrow{add}_{a,b}$ with son u_1, then $cval(t)/u := \overrightarrow{add}_{a,b}(cval(t)/u_1)$,

if u is an occurrence of $add_{a,b}$ with son u_1, then $cval(t)/u := add_{a,b}(cval(t)/u_1)$.

Finally, $cval(t) := cval(t)/root_t$. Its vertex set is thus $Pos_0(t)$. For example, consider the term

$$t = add^1_{b,c}(add^2_{a,b}(\mathbf{a}^3 \oplus^4 \mathbf{b}^5) \oplus^6 relab^7_h(add^8_{a,b}(\mathbf{a}^9 \oplus^{10} \mathbf{b}^{11})))$$

where h replaces b by c and the superscripts 1 to 11 number the occurrences of its function symbols (including the nullary ones), so that $Pos(t) = [11]$. The concrete p-graph $cval(t)$ is

$$3_a - 5_b - 11_c - 9_a$$

where the subscripts a, b, c indicate the port labels. If $u = 2$ and $w = 8$, then $t/u = t/w = add_{a,b}(\mathbf{a} \oplus \mathbf{b})$. However, $cval(t)/u$ is the concrete p-graph $3_a - 5_b$ and $cval(t)/w$ is $9_a - 11_b$, isomorphic to $cval(t)/u$. Both are isomorphic to $cval(t/u)$ whose vertex set is $\{2, 4\}$, since, with numbered positions, $t/u = add^1_{a,b}(\mathbf{a}^2 \oplus^3 \mathbf{b}^4)$.

(b) Let $t \in T(F_k) \cup T(F^{\mathrm{u}}_k)$ and $X \subseteq Pos_0(t)$. Let t' be the term obtained by replacing, for each $u \in X$, the symbol occurring there by $\varnothing$. It is clear from the above definition that $cval(t')$ is the induced subgraph $cval(t)[Pos_0(t) - X]$ of $cval(t)$.

(c) The *clique-width* of a graph G, denoted by $cwd(G)$, is the least integer k such that G is isomorphic to $cval(t)$ for some t in $T(F_k)$ (in $T(F^{\mathrm{u}}_k)$ if it is undirected). A k-expression is *slim* ([CouEng], Chapter 2) if for each of its subterms of the form $t_1 \oplus t_2$, at least one of t_1 and t_2 is a nullary symbol. The *linear clique-width* of G, denoted by $lcwd(G)$, is the least integer k such that G is defined (up to isomorphism) by a slim k-expression.

The parsing problem

The problem of deciding if $cwd(G) \leq k$ for a given pair (G, k) is NP-complete ([14]). It is not known if this problem is NP-complete for any fixed $k \geq 4$. However, algorithms of [26] (for undirected graphs) and [24] (for directed graphs) that use rank-decompositions as intermediate steps together with a construction from [27][4] give cubic approximation algorithms: these algorithms report in time $g(k) \cdot n^3$ that $cwd(G) > k$ or output a term in $T(F_{f(k)})$ that defines G (with n vertices) where g and f are fixed functions. Together with the constructions of automata detailed below, these algorithms yield fixed-parameter *cubic* algorithms for checking monadic second-order graph properties with respect to clique-width as parameter.

If the parameter is tree-width, we obtain a fixed-parameter *linear* algorithm for checking monadic second-order graph properties because it is possible to construct in linear time a tree-decomposition of width at most k of a given graph G if there exists one. The corresponding algorithm, due to Bodlaender, is presented in [12]. Then, this tree-decomposition can be transformed in linear time into an $h(k)$-expression defining G, where h is a fixed function (this function depends on whether G is directed or not; see Chapters 2 and 6 of [9] for details).

2.3. Annotated terms

Definition 6. *Annotations*

(a) An *annotation* of a term $t \in T(F_k) \cup T(F^{\mathrm{u}}_k)$ is a mapping that associates with some nodes u of t an information relative to t/u or to the context of u in t or to both.

(b) We now define a particular annotation intended to represent, for each u the edge additions that occur on the path between u and the root of t; its definition takes also into account the relabellings occurring on this path. It is thus relative to the context of each node. Another notion of annotation will be defined in Section 6.4.

We first consider the case of a term $t \in T(F_k)$ and we introduce some notation. If $a, b \in [k]$ and $u, w \in Pos(t)$, we write $(a, u) \rightarrow_t (b, w)$ if:

[4] See also Chapter 6 of [9].

w is the father of u and,

either w is an occurrence of $\oplus$ or $\overrightarrow{add}_{c,d}$ (for some $c, d \in C$) and then $b = a$, or w is an occurrence of $relab_h$ and $b = h(a)$.

We let $\rightarrow_t^+$ be the transitive closure of $\rightarrow_t$. This relation describes the effect of the relabellings at occurrences between a node and one of its proper ancestors.

We define $ADD_t(u)$ as the set of pairs (a, b) such that $(a, u) \rightarrow_t^+ (c, w)$ and $(b, u) \rightarrow_t^+ (d, w)$ for some occurrence of $\overrightarrow{add}_{c,d}$ at w (which implies that $a \neq b$). These conditions imply that w is a proper ancestor of u and that, if x is an a-port of $cval(t)/u$ and y is a b-port of $cval(t)/u$, then, there is in $cval(t)/w$ (hence also in $cval(t)$) an edge from x to y (because x is a c-port of $cval(t)/w_1$ and y is a d-port of $cval(t)/w_1$ where w_1 is the son of w). If this edge does not exist in $cval(t)/v$ for any v with $u \preccurlyeq_t v \prec_t w$, then it is created by $\overrightarrow{add}_{c,d}$ at w. Otherwise, $\overrightarrow{add}_{c,d}$ at w is superfluous regarding this particular edge.

For each term $t \in T(F_k)$, the sets $ADD_t(u)$ can be computed top-down in linear time as follows:

if u is the root, then $ADD_t(u) := \emptyset$,

if u is a son of an occurrence w of $\oplus$, then $ADD_t(u) := ADD_t(w)$,

if u is the son of an occurrence w of $\overrightarrow{add}_{a,b}$, then $ADD_t(u) := ADD_t(w) \cup \{(a, b)\}$,

if u is the son of an occurrence w of $relab_h$, then $ADD_t(u) := h^{-1}(ADD_t(w)) := \{(a, b) \mid (h(a), h(b)) \in ADD_t(w)\}$.

The validity of this algorithm follows from the definitions.

The definition is similar if $t \in T(F_k^{\mathrm{u}})$: $ADD_t(u)$ is the set of two element sets $\{a, b\}$ such that $(a, u) \rightarrow_t^+ (c, w)$ and $(b, u) \rightarrow_t^+ (d, w)$ for some occurrence w of $add_{c,d}$ or $add_{d,c}$. These relations can be computed in linear time in a similar way as in the first case.

We let (t, ADD_t) be the annotation of t that attaches $ADD_t(u)$ to each node u of $t \in T(F_k) \cup T(F_k^{\mathrm{u}})$. Note that ADD_t does not depend on the nullary symbols: if t' is obtained from t by replacements of nullary symbols, in particular by replacing some nullary symbols by $\varnothing$, then $ADD_{t'} = ADD_t$ and (t', ADD_t) is the corresponding annotation of t'. Note also that $ADD_t(u)$ depends on unary operations in t *strictly above* u.

We show in Figure 1 the annotation ADD_t for the term:

$$t = \overrightarrow{add}_{a,c}(\overrightarrow{add}_{b,a}(\mathbf{a} \oplus \mathbf{b}) \oplus relab_h(\overrightarrow{add}_{a,b}(\mathbf{a} \oplus \mathbf{b}))),$$

where $h(a) = a, h(b) = h(c) = c$. The relation $ADD_t(u)$ is shown to the right of each node u.

2.3.1. Useless operations and redundancy elimination

Some operations in a term may have no effect: for example a disjoint union, one argument of which is the empty graph. This is also the case of an operation $\overrightarrow{add}_{a,b}$ or $add_{a,b}$ at u in term t if a or b does not belong to $\pi(cval(t)/u_1)$ where u_1 is the son of u. A bottom-up traversal of t can identify these cases.

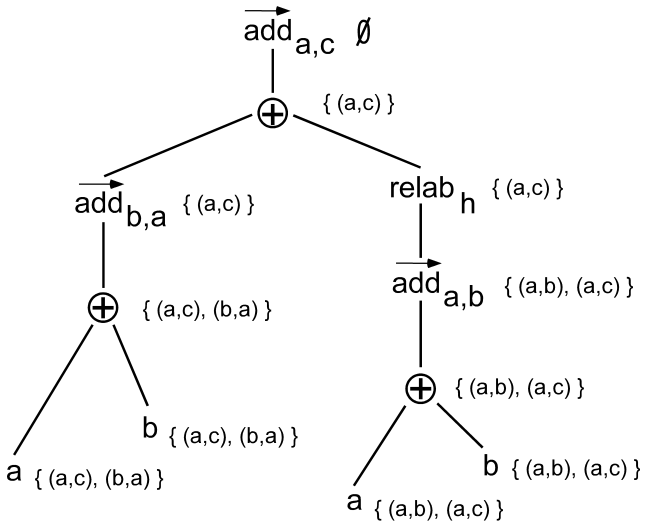

Figure 1: Annotated term.

We now consider other cases where a unary operation is useless. If a term $t' = \overrightarrow{add}_{a,b}(t)$ is such that $cval(t)$ has already an edge e from an a-port to a b-port, then this term presents a redundancy in the sense that the edge e is specified at least twice: once (or more) in $cval(t)$ and another time at the root of t'. So, its specification(s) in $cval(t)$ is (or are) useless.

Definition 7. *Redundancy.*

(a) Let $t \in T(F_k)$. We say that a 6-tuple (u, w, a, b, c, d) *defines* (or *is* to be short) *a redundancy* in t if u is an occurrence of $\overrightarrow{add}_{a,b}$, w is an occurrence of $\overrightarrow{add}_{c,d}$ which is a proper ancestor of u and $(a, b) \in ADD_t(u)$. If $t \in T(F_k^{\mathrm{u}})$ then (u, w, a, b, c, d) is a *redundancy* in t if u is an occurrence of $add_{a,b}$, w is an occurrence of $add_{c,d}$ that is a proper ancestor of u and $\{a, b\} \in ADD_t(u)$. In the term of Figure 1, the tuple (u, w, a, b, a, c) such that u is the occurrence of $\overrightarrow{add}_{a,b}$ and w is the occurrence of $\overrightarrow{add}_{a,c}$ defines a redundancy.

(b) A term is *irredundant* if it has no redundancies. We denote by $IT(F_k)$ (resp. by $IT(F_k^{\mathrm{u}})$) the set of irredundant terms in $T(F_k)$ (resp. in $T(F_k^{\mathrm{u}})$).

Note that the definition of a redundancy does not depend on the nullary symbols: a term written with no other nullary symbol than $\varnothing$ (it defines the empty graph) may be irredundant although all its symbols (except for one occurrence of $\varnothing$) are useless.

Proposition 8. (1) *For each integer k, there exists a linear time algorithm that transforms a term t belonging to $T(F_k) \cup T(F_k^{\mathrm{u}})$ into a term in $IT(F_k) \cup IT(F_k^{\mathrm{u}})$ that defines the same graph and is a relabelling of t.*

(2) *For each k, the set $IT(F_k)$ is recognized by a deterministic F_k-automaton with at most 2^{k^2-k+2} states. The corresponding automaton for $IT(F_k^{\mathrm{u}})$ has at most $2^{k(k-1)/2+2}$ states.*

Proof. (1) We will use the following observation:

Claim: If t has a redundancy (u, w, a, b, c, d) and if t' is obtained from t by the replacement of $\overrightarrow{add}_{a,b}$ (or $add_{a,b}$) at u by $relab_{\mathrm{Id}}$, then $cval(t') = cval(t)$.

Note that since we replace $\overrightarrow{add}_{a,b}$ at u by the identity, we have $Pos(t') = Pos(t)$ and the concrete graphs $cval(t')$ and $cval(t)$ have the same sets of vertices. They are thus compared as concrete graphs and not up to isomorphism.

Proof. We have $cval(t')/w = cval(t)/w$ because the edges of $cval(t)/w$ that are not created in $cval(t')/w$ by $\overrightarrow{add}_{a,b}$ at u are anyway created by $\overrightarrow{add}_{c,d}$ at w. The same holds for $add_{a,b}$ and $add_{c,d}$. □

The computation of the sets $ADD_t(u)$, hence of the annotation (t, ADD_t) of t, can be done in linear time by means of a depth-first traversal of t that starts at the root. The term t' is then obtained from (t, ADD_t) by replacing $\overrightarrow{add}_{a,b}$ occurring at any u by $relab_{\mathrm{Id}}$ whenever $(a, b) \in ADD_t(u)$ (respectively, by replacing $add_{a,b}$ at any u by $relab_{\mathrm{Id}}$ whenever $\{a, b\} \in ADD_t(u)$ if $t \in T(F_k^{\mathrm{u}})$). This can be done also in linear time. (In the example of Figure 1, the operation $\overrightarrow{add}_{a,b}$ can be replaced by $relab_{\mathrm{Id}}$.)

(2) It is clear that t is irredundant if no replacement is made. From Definitions 6 and 7, one gets a bottom-up nondeterministic automaton with $k(k-1)+2$ states, hence, by determinizing it, one gets a complete and deterministic one with at most 2^{k^2-k+2} states. The proof for $IT(F_k^{\mathrm{u}})$ is similar. □

2.3.2. Edge-complement

This transformation concerns undirected graphs without loops. It is another application of the annotation of Definition 6.

Proposition 9. *For each k, there is a linear-time algorithm that transforms a term t in $T(F_k^{\mathrm{u}})$ defining a loop-free graph into a term $\overline{t}$ in $T(F_{2k}^{\mathrm{u}})$ such that $cval(\overline{t})$ is isomorphic to the edge complement of the graph $cval(t)$.*

Proof. We prove the existence of $\overline{t}$ by using $[k+1, 2k]$ as a disjoint copy of $[k]$ and the mappings $r : [2k] \to [k+1, 2k]$ and $r' : [2k] \to [k]$ such that:

- $r(i) := \texttt{if } i \leq k \texttt{ then } i + k \texttt{ else } i$,
- $r'(i) := \texttt{if } i > k \texttt{ then } i - k \texttt{ else } i$.

If A is a set of 2-element subsets of $[2k]$, we abbreviate into add_A the composition (in any order) of the operations $add_{a,b}$ for all $\{a, b\}$ in A. We now define $\overline{t}$ by induction on the number of occurrences of $\oplus$.

If t has no occurrence of $\oplus$, then $cval(t)$ is the empty graph or a vertex and we take $\overline{t} := t$.

Otherwise, let u be the topmost occurrence of $\oplus$ in t, $B := ADD_t(u)$ and m be the composition of the mappings h in the operations $relab_h$ on the path in t from u to the root. It follows that $cval(t) = relab_m(add_B(cval(t)/u))$. Letting u_1 and u_2 be the two sons of u, we define

$B' := \{\{a, b+k\} \mid a, b \in [k], \{a,b\} \notin B\}$ and
$\bar{t} := relab_{m \circ r'}(add_{B'}(\overline{add_B(t/u_1)} \oplus relab_r(\overline{add_B(t/u_2)}))).$

In particular, B' contains all pairs $\{a, a+k\}$ for $a \in [k]$ because $ADD_t(u)$ contains only sets $\{a,b\}$ with $a \neq b$. The terms $add_B(t/u_1)$ and $add_B(t/u_2)$ have less occurrences of $\oplus$, hence $\overline{add_B(t/u_1)}$ and $\overline{add_B(t/u_2)}$ are well-defined by induction.

The mapping $t \mapsto \bar{t}$ satisfies the required properties and is computable in linear time. It can be described as a transformation of (t, ADD_t) into $\bar{t}$. □

Note that the set of positions of $\bar{t}$ is not equal to that of t. Hence, the graphs $cval(t)$ and $cval(\bar{t})$ do not have the same vertices, however, they are isomorphic. The proof of this proposition given in [11] does not give a linear-time algorithm.

2.4. Terms and graph properties

Definition 10. *Graph properties.*

If P is a graph property, say planarity to take an example, we let $L_{P,k}$ be the set of terms t in $T(F_k)$ such that $cval(t)$ satisfies P. We want to extend this notion to any property $P(X_1, ..., X_n)$ of sets of vertices $X_1, ..., X_n$ of a graph $cval(t)$. We also call $P(X_1, ..., X_n)$ a *graph property* to simplify the terminology.

Here are three examples of basic graph properties that we will use to build more complicated ones. The considered graph is G.

- $Link(X, Y)$: there is an edge with tail in X and head in Y,
- $Dom(X, Y)$: every x in X is the head of an edge with tail in Y; we say that Y *dominates* X,
- $Path(X, Y)$: $X \subseteq Y$, $|X| = 2$ and the two vertices of X are linked by a path in $G[Y]$ if G is undirected or in $Und(G)[Y]$ if G is directed.

Definition 11. *Graph properties encoded in terms.*

(a) We define $F_k^{(n)}$ from F_k by replacing there all symbols $\mathbf{a}$ and $\mathbf{a}^\ell$ by the nullary symbols $(\mathbf{a}, w)$ and $(\mathbf{a}^\ell, w)$ for all $w \in \{0,1\}^n$. We define $pr : F_k^{(n)} \to F_k$ as the mapping that deletes the sequences w of the nullary symbols. It extends into a relabelling $pr : T(F_k^{(n)}) \to T(F_k)$. A term t in $T(F_k^{(n)})$ defines thus the graph $cval(pr(t))$ and the n-tuple $(V_1, ..., V_n)$ such that V_i is the set of vertices which are occurrences of nullary symbols $(\mathbf{a}, w)$ or $(\mathbf{a}^\ell, w)$ such that the i-th component of w is 1.

(b) If $P(X_1, ..., X_n)$ is a graph property (the notation $P(X_1, ..., X_n)$ shows that it depends on n arguments; in some cases, its arguments are $Y_1, ..., Y_n$), we define $L_{P(X_1,...,X_n),k}$ as the set of terms t in $T(F_k^{(n)})$ such that $P(V_1, ..., V_n)$ is true in $cval(pr(t))$, where $(V_1, ..., V_n)$ is the n-tuple of sets of vertices encoded by t. Such a term t will be denoted by $s * (V_1, ..., V_n)$ where $s = pr(t) \in T(F_k)$.

Definition 12. *Set terms and substitutions*

(a) A *set term* over a set $\{X_1, ..., X_n\}$ of set variables is a term S written with these variables, the constant symbol $\varnothing$ for denoting the empty set and the operations $\cap$, $\cup$ and – (for complementation). An example is $S = X_1 \cup \overline{X_3}$.

(b) Let $P(Y_1, ..., Y_m)$ denote a graph property and $S_1, ..., S_m$ be set terms over $\{X_1, ..., X_n\}$. Then $P(S_1, ..., S_m)$ defines the property $P(Y_1, ..., Y_m)$ where each argument Y_i is replaced by S_i. The corresponding set of terms in $T(F_k^{(n)})$ is denoted by $L_{P(S_1,...,S_m),(X_1,...,X_n),k}$. The subscript $(X_1, ..., X_n)$ indicates that this set of terms is defined as if $P(S_1, ..., S_m)$ depended actually on *all variables* $X_1, ..., X_n$ although this is not always the case (because some X_i may have no occurrence in $(S_1, ..., S_m)$).

Lemma 13. *For every graph property $P(Y_1, ..., Y_m)$ and m-tuple of set terms $S_1, ..., S_m$ over $\{X_1, ..., X_n\}$, we have:*

$$L_{P(S_1,...,S_m),(X_1,...,X_n),k} = h^{-1}(L_{P(Y_1,...,Y_m),k})$$

where h is a relabelling: $T(F_k^{(n)}) \to T(F_k^{(m)})$ that replaces each nullary symbol $(\mathbf{c}, w)$ for $w \in \{0,1\}^n$ and $\mathbf{c} \in \mathbf{C}$ by $(\mathbf{c}, w')$ for some $w' \in \{0,1\}^m$ and does not modify the other symbols.

The proof is routine, we only give examples. Sequences of Booleans are denoted as words over $\{0,1\}$. Let $n := 4$, $S_1 := X_1 \cup \overline{X_3}$ and P be unary ($m = 1$). Then $L_{P(S_1),(X_1,...,X_4),k} = h^{-1}(L_{P(Y_1),k})$ where, for every $x, y \in \{0,1\}$ and $\mathbf{c} \in \mathbf{C}$:

$h((\mathbf{c}, 1x0y)) = h((\mathbf{c}, 1x1y)) = h((\mathbf{c}, 0x0y)) = (\mathbf{c}, 1)$ and
$h((\mathbf{c}, 0x1y)) = (\mathbf{c}, 0)$,

i.e., $h((\mathbf{c}, x_1x_2x_3x_4)) = (\mathbf{c}, x_1 \vee \neg x_3)$. Hence h encodes the set term S_1 in a natural way.

For another example, consider $P(Y_1, Y_2, Y_3)$ and $Q(X_1)$ defined as $P(X_1, \varnothing, \overline{\varnothing})$. Then we have: $L_{Q(X_1),k} = h^{-1}(L_{P(Y_1,Y_2,Y_3),k})$ where $h((\mathbf{c}, 0)) = (\mathbf{c}, 001)$ and $h((\mathbf{c}, 1)) = (\mathbf{c}, 101)$.

Lemma 13 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. From an automaton recognizing, say $L_{P(Y_1,Y_2,Y_3),k}$ to take an example, we can easily obtain one recognizing $L_{P(X_2,X_4,X_4),(X_1,...,X_4),k}$.

Definition 14. *Relativization*

For a graph property $P(X_1, \cdots, X_m)$, we let $P(X_1, \cdots, X_m)[X_{m+1}]$ be the property $Q(X_1, \cdots, X_m, X_{m+1})$ such that,

> for all sets of vertices $X_1, \cdots, X_{m+1}$ of a graph G, $Q(X_1, \cdots, X_m, X_{m+1})$ is true if and only if $P(X_1 \cap X_{m+1}, \cdots, X_m \cap X_{m+1})$ is true in the induced subgraph $G[X_{m+1}]$.

We define h as the mapping: $F_k^{(m+1)} \to F_k^{(m)}$ such that, for every $\mathbf{c} \in \mathbf{C}$ and $w \in \{0,1\}^m$, we have $h((\mathbf{c}, w0)) := \varnothing$ and $h((\mathbf{c}, w1)) := (\mathbf{c}, w)$. With these hypotheses and notation:

Lemma 15. *We have $L_{P(X_1,\cdots,X_m)[X_{m+1}],k} = h^{-1}(L_{P(X_1,\cdots,X_m),k})$.*

Proof. Let $t * (V_1, \cdots, V_{m+1})$ belong to $L_{P(X_1,\cdots,X_m)[X_{m+1}],k}$ and $G := cval(t)$. Then

$$h(t*(V_1,\cdots,V_{m+1})) = t'*(V_1\cap V_{m+1},\cdots,V_m\cap V_{m+1}),$$

where, by the definitions, t' evaluates to $G' := G[V_{m+1}]$ (because replacing nullary symbols by $\varnothing$ corresponds to defining induced subgraphs, cf. Definition 5(b))). It follows that $t'*(V_1\cap V_{m+1},\cdots,V_m\cap V_{m+1}) \in L_{P(X_1,\cdots,X_m),k}$. This proves the inclusion from left to right. The proof of the opposite inclusion is similar. □

This lemma is not a special case of Lemma 13 because, in general, $P(X_1,\cdots,X_m)[X_{m+1}]$ is not equivalent to $P(X_1\cap X_{m+1},\cdots,X_m\cap X_{m+1})$ in G. Take for a counter-example the property $P(X_1)$ expressing that any two vertices of X_1 are linked by a path (not necessarily in $G[X_1]$). Then $P(X_1)[X_2]$ is not equivalent to $P(X_1\cap X_2)$.[5] However, if P is *Link*, *Dom* or *Path* (these properties are defined above, after Definition 10), then $P(X_1,X_2)[X_3]$ is equivalent to $P(X_1\cap X_3, X_2\cap X_3)$.

In the following definition, we combine the constructions of Definitions 12 and 14. We let $\mathcal{P}$ be a set of *basic* graph properties containing those defined by atomic formulas (e.g., $X_1\subseteq X_2$) and properties such as *Link*, *Dom*, *Path* and *Conn*. We will specify later the atomic formulas (Definition 18) and the other basic properties (Section 4.1). We consider $\mathcal{P}$ as a parameter.

Definition 16. *$\mathcal{P}$-atomic formulas*

A *$\mathcal{P}$-atomic formula* is a formula of the form $P(S_1,...,S_m)$ or $P(S_1,...,S_m)[S_{m+1}]$ such that $P(Y_1,...,Y_m)\in\mathcal{P}$ and $S_1,...,S_m,S_{m+1}$ are set terms over $\{X_1,...,X_n\}$. Its free variables are in $\{X_1,...,X_n\}$.

Lemmas 13 and 15 entail that $L_{P(S_1,...,S_m)[S_{m+1}],(X_1,...,X_n),k} = h^{-1}(L_{P(Y_1,\cdots,Y_m),k})$ for some relabelling h that modifies only nullary symbols.

3. Automata on terms

Although finite automata on terms (frequently called "tree-automata"; our reference is the book on line [4]) are well-known, we review notation and basic facts relative to them and to *infinite* automata that we will also use in Section 7.

Definition 17. *Automata.*

All automata will be bottom-up (or frontier-to-root) without ε-transition.

(a) Let F be a finite or countably infinite signature. An *F-automaton* (or just an *automaton* if F need not be specified) is a 4-tuple $\mathcal{A} = \langle F, Q_\mathcal{A}, \delta_\mathcal{A}, Acc_\mathcal{A}\rangle$ such that $Q_\mathcal{A}$ is a finite or countably infinite set called the set of *states*, $Acc_\mathcal{A}$ is a subset of $Q_\mathcal{A}$ called the set of *accepting states* and $\delta_\mathcal{A}$ is a set of tuples called the *transition rules* (or the *transitions*). Each transition rule is of the form $(q_1,\ldots,q_m,f,q)$ with $q_1,\ldots,q_m,q\in Q_\mathcal{A}$, $f\in F$, with $\rho(f)=m\geq 0$.

For better readability, we will denote it by $f[q_1,\ldots,q_m]\to_\mathcal{A} q$ (and by $f\to_\mathcal{A} q$ if f is nullary). This transition is said to *yield* q. (See (f) below for an alternative notation.)

(b) We say that $\mathcal{A}$ is *finite* if F and $Q_\mathcal{A}$ are finite. In this case, the number of states is denoted by $\sharp\mathcal{A}$. The size of $\mathcal{A}$, defined as the space needed to store its transitions as a list of

[5] $P[X_1]$ expresses that the induced subgraph $G[X_1]$ is connected.

tuples can be evaluated in two ways. To simplify the discussion, we only consider the case of function symbols that are at most binary (we will only use this case).

If we assume that each state and operation symbol occupies a unit space, then the size of $\mathcal{A}$ is $O(|F| \cdot (\sharp\mathcal{A})^3)$; it is only $O(|F| \cdot (\sharp\mathcal{A})^2)$ if $\mathcal{A}$ is deterministic (the definition is recalled in (f) below). If the number of states is very large (which is also our case), a state may need $\lceil \log(\sharp\mathcal{A}) \rceil$ bits to be stored[6], in this case, the sizes are respectively $O(|F| \cdot (\sharp\mathcal{A})^3 \cdot \log(\sharp\mathcal{A}))$ and $O(|F| \cdot (\sharp\mathcal{A})^2 \cdot \log(\sharp\mathcal{A}))$. Other similar parameters will be defined in Section 7 about *fly automata*.

(c) A *run* of an automaton $\mathcal{A}$ on a term $t \in T(F)$ is a mapping $r : Pos(t) \to Q_{\mathcal{A}}$ such that:

> if u is an occurrence of a function symbol $f \in F$ and $u_1, ..., u_{\rho(f)}$ is the sequence of sons of u, then $f[r(u_1), \ldots, r(u_{\rho(f)})] \to_{\mathcal{A}} r(u)$; (if $\rho(f) = 0$, the condition reads $f \to_{\mathcal{A}} r(u)$).

(d) For a state q, we let $L(\mathcal{A}, q)$ be the set of terms t in $T(F)$ on which there is a run r of $\mathcal{A}$ such that $r(root_t) = q$. A run r on t is *accepting* if $r(root_t)$ is an accepting state. We let $L(\mathcal{A}) := \bigcup_{q \in Acc_{\mathcal{A}}} L(\mathcal{A}, q) \subseteq T(F)$. We say that $L(\mathcal{A})$ is the *language accepted* (or *recognized*) by $\mathcal{A}$. Two automata are *equivalent* if they accept the same language. We define a language (a set of terms) as *regular* if it is accepted by a finite automaton.

(e) A state q of an automaton $\mathcal{A}$ is *accessible* if $L(\mathcal{A}, q) \neq \emptyset$, i.e., if it occurs in a run of $\mathcal{A}$ on some term, not necessarily in $L(\mathcal{A})$. We say that $\mathcal{A}$ is *trim* if each state occurs in an accepting run. (In particular, each state is accessible). It is well-known, see [4], that one can *trim* a finite automaton $\mathcal{A}$, that is, one can replace it by an equivalent trim one by deleting some states and transitions. If $L(\mathcal{A})$ is empty, one gets in this way an automaton with an empty set of states. A *sink* is a state s such that, for every transition $f[q_1, \ldots, q_m] \to_{\mathcal{A}} q$, we have $q = s$ if $q_i = s$ for some i. If F has at least one symbol of arity at least 2, then an automaton can have at most one sink. A state named $Success$ (resp. $Error$) will be an accepting (resp. nonaccepting) sink.

(f) *Complete and deterministic automata.*

An F-automaton $\mathcal{A}$ is *complete* if for every $f \in F$ of arity m and every $q_1, \ldots, q_m$ in $Q_{\mathcal{A}}$, there is at least one transition $f[q_1, \ldots, q_m] \to_{\mathcal{A}} q$. By adding at most one nonaccepting sink, one can transform an automaton $\mathcal{A}$ that is not complete into a complete one $\mathcal{B}$ such that $L(\mathcal{B}, q) = L(\mathcal{A}, q)$ for every state q of $\mathcal{A}$. We omit the details.

A transition $f[q_1, \ldots, q_m] \to_{\mathcal{A}} q$ of an automaton $\mathcal{A}$ is *deterministic* if there is no transition $f[q_1, \ldots, q_m] \to_{\mathcal{A}} q'$ with $q' \neq q$. An automaton is *deterministic* if all its transitions are deterministic, and in this case, we denote q such that $f[q_1, \ldots, q_m] \to_{\mathcal{A}} q$ by $\gamma_{\mathcal{A}}(f, q_1, \ldots, q_m)$.

If $\mathcal{A}$ is deterministic and complete, then it has on each $t \in T(F)$ a unique run, which we denote by $run_{\mathcal{A},t}$. This run can be computed during a bottom-up traversal of t in time $a \cdot |t|$ where a is an upper-bound to the time taken to perform a transition, that is, to find or compute $\gamma_{\mathcal{A}}(f, q_1, \ldots, q_m)$. This value a is significant if the transition has to be computed (cf. Section 7),

[6]Space efficient representations of automata are used in the software MONA: see [25].

but also if the automaton is finite but is so large that some time (no longer considered as constant) is required to find the appropriate transition in a table.[7]

By adding a nonaccepting sink to a deterministic automaton, one makes it complete while preserving determinism. The presence of an accepting sink (resp. of a nonaccepting sink in a deterministic automaton) accelerates acceptance (resp. rejection) of a term if this sink occurs in a run on this term.

(g) *Determinization*

We recall the existence of a *determinization* algorithm for finite automata (cf. Section 5 of [20] or Theorem 1.1.9 of [4]): For every finite automaton $\mathcal{A}$ that is not deterministic, one can construct a trim, deterministic finite automaton $\mathcal{B}$ that is equivalent to $\mathcal{A}$. The states of $\mathcal{B}$ are sets of states of $\mathcal{A}$, hence, $\sharp\mathcal{B} \leq 2^{\sharp\mathcal{A}}$. To get a complete deterministic automaton, one can use the empty subset of $Q_{\mathcal{A}}$ as nonaccepting sink.

For every automaton $\mathcal{A}$ and every term $t \in T(F)$, we denote by $run^*_{\mathcal{A},t}$ the mapping: $Pos(t) \to \mathcal{P}(Q_{\mathcal{A}})$, that associates with every u in $Pos(t)$ the set of states of the form $r(root_{t/u})$ for some run[8] r on the subterm t/u of t. In particular, $run^*_{\mathcal{A},t} = run_{\mathcal{B},t}$ where $\mathcal{B}$ is the determinized automaton of a finite automaton $\mathcal{A}$ with set of states included in $\mathcal{P}(Q_{\mathcal{A}})$. We define $ndeg_{\mathcal{A}}(t)$, the *degree of nondeterminism of* $\mathcal{A}$ *on* t as the maximal cardinality of $run^*_{\mathcal{A},t}(u)$ for u in $Pos(t)$.

(h) *Images and inverse images*

Let $h : T(H) \to T(F)$ be a relabelling (cf. Definition 2). If $L \subseteq T(H)$, then $h(L) := \{h(t) \mid t \in L\}$. If $\mathcal{A}$ is an H-automaton, we let $h(\mathcal{A})$ be the F-automaton obtained from $\mathcal{A}$ by replacing each transition $f[q_1, \cdots, q_{\rho(f)}] \to_{\mathcal{A}} q$ by $h(f)[q_1, \cdots, q_{\rho(f)}] \to q$.

Clearly, $h(L(\mathcal{A}, q)) = L(h(\mathcal{A}), q)$ for every state q and $h(L(\mathcal{A})) = L(h(\mathcal{A}))$ (because $h(\mathcal{A})$ has the same accepting states as $\mathcal{A}$). We say that $h(\mathcal{A})$ is the *image of* $\mathcal{A}$ under h. If $\mathcal{A}$ is deterministic, then $h(\mathcal{A})$ is not necessarily deterministic, but it is if h is injective.

We now consider inverse images. If $K \subseteq T(F)$, we define $h^{-1}(K)$ as $\{t \in T(H) \mid h(t) \in K\}$. If $\mathcal{A}$ is an F-automaton, then we let $h^{-1}(\mathcal{A})$ be the H-automaton with transitions of the form $f[q_1, \cdots, q_{\rho(f)}] \to q$ such that $h(f)[q_1, \cdots, q_{\rho(f)}] \to_{\mathcal{A}} q$. We have $L(h^{-1}(\mathcal{A}), q) = h^{-1}(L(\mathcal{A}, q))$ for every state q, and $L(h^{-1}(\mathcal{A})) = h^{-1}(L(\mathcal{A}))$. We call $h^{-1}(\mathcal{A})$ the *inverse image* of $\mathcal{A}$ under h. Note that $h^{-1}(\mathcal{A})$ is deterministic (resp. complete) if $\mathcal{A}$ is so.

(i) *Subsignatures and subautomata*

We say that a signature H is a *subsignature* of F, written $H \subseteq F$, if every operation of H is one of F with the same arity. We say that an H-automaton $\mathcal{B}$ is a *subautomaton* of an F-automaton $\mathcal{A}$ (is *included in* $\mathcal{A}$), which we denote by $\mathcal{B} \subseteq \mathcal{A}$, if:

$H \subseteq F$, $Q_{\mathcal{B}} \subseteq Q_{\mathcal{A}}$, $Acc_{\mathcal{B}} = Acc_{\mathcal{A}} \cap Q_{\mathcal{B}}$, and

$\delta_{\mathcal{B}}$ is the set of transitions $f[q_1, \ldots, q_r] \to q$ of $\mathcal{A}$ such that $f \in H$ and $q_1, \ldots, q_r \in Q_{\mathcal{B}}$.

[7] In most classical uses of automata, e.g., in compilation or text processing, the size of the input is much larger than the number of states, hence, the value a may be considered as a constant. But in this article, we will construct automata that are much larger than their intended input terms.

[8] If $\mathcal{A}$ is not complete, this run may not be the restriction of a run on t.

These definitions imply that $L(\mathcal{B}) = L(\mathcal{A}) \cap T(H)$. (For proving the inclusion from right to left, we observe by bottom-up induction that all states of a run of $\mathcal{A}$ on a term in $T(H)$ belong to $Q_{\mathcal{B}}$. Hence, this run is a run of $\mathcal{B}$.)

If $\mathcal{A}$ is an F-automaton and $H \subseteq F$, there is a unique automaton $\mathcal{B} \subseteq \mathcal{A}$ that is minimal for inclusion. Every increasing sequences of F-automata $\mathcal{B}_1 \subseteq \mathcal{B}_2 \subseteq \cdots \mathcal{B}_n \subseteq \cdots$ has a *union*: it is an F-automaton $\mathcal{A}$ such that $\mathcal{B}_n \subseteq \mathcal{A}$ for each n and its set of states is the union of the sets $Q_{\mathcal{B}_n}$. It recognizes the union of the languages $L(\mathcal{B}_n)$.

4. Monadic second-order logic

We now review the expression of graph properties by monadic second-order formulas and sentences. (A sentence is a formula without free variables).

Definition 18. *Monadic second-order formulas expressing graph properties.*

(a) We have defined a simple graph G as the relational structure $\langle V_G, edg_G \rangle$ with domain V_G and a binary relation edg_G such that $(x, y) \in edg_G$ if and only if there is an edge from x to y (or between x and y if G is undirected). A p-graph G whose type $\pi(G)$ is included in a fixed finite set C (cf. Definition 3) is identified with the structure $\langle V_G, edg_G, (lab_{aG})_{a \in C} \rangle$ where lab_{aG} is the set of a-ports of G.

(b) Monadic second-order formulas (MS formulas in short) will be written with the set variables $X_1, ..., X_n, ...$, i.e., without first-order variables, which is not a loss of generality (see, e.g., Chapter 5 of [9]). The atomic formulas intended to describe the edges and port labels of a graph G are:

$edg(X_i, X_j)$ meaning that X_i and X_j denote singleton sets $\{x\}$ and $\{y\}$ such that $x \rightarrow_G y$ and

$lab_a^{\forall}(X_i)$ meaning that every element of X_i has port label a.

The other atomic formulas are $X_i \subseteq X_j$, $X_i = \emptyset$, $Sgl(X_i)$ (meaning that X_i denotes a singleton set) and $Card_{p,q}(X_i)$ (meaning that the cardinality of X_i is equal to p modulo q, with $0 \leq p < q$ and $q \geq 2$).[9]

Furthermore, MS formulas are written without universal quantifications, and the free variables of every (sub)formula of the form $\exists X_n.\varphi$ are among $X_1, ..., X_{n-1}$. These syntactic constraints yield no loss of generality (see Chapter 6 of [9] for details) but they make easier the construction of automata.

(c) A graph property $P(X_1, ..., X_n)$ is an *MS property* if there exists an MS formula $\varphi(X_1, ..., X_n)$ such that, for every graph G and for all sets of vertices $V_1, ..., V_n$ of this graph, we have:

$$\langle V_G, edg_G, (lab_{aG})_{a \in C} \rangle \models \varphi(V_1, ..., V_n) \text{ if and only if } P(V_1, ..., V_n) \text{ is true in } G.$$

[9] We will not distinguish monadic second-order formulas from *counting* monadic second-order formulas, defined as those using $Card_{p,q}(X_i)$, because all our results will hold in the same way for both types. See Chapter 5 of [9] for situations where the distinction matters.

The formulas that do not use the atomic formulas $lab_a^{\forall}(X_i)$ express properties of graphs, equivalently, properties of p-graphs that do not depend on port labels.

For each MS graph property $P(X_1, ..., X_n)$, the set of terms $L_{P(X_1,...,X_n),k} \subseteq T(F_k^{(n)})$ is regular. This result yields, for every monadic second-order property, a fixed-parameter cubic model-checking algorithm with respect to clique-width as parameter. Detailed constructions of automata will be given in Section 5 (see also Section 6.3.4 of [9]). However, the corresponding automata are frequently much too large to be constructed in practice. This is partly due to the level of nesting of negations in the formulas, but also to the number k: for example, the number of states of the unique *minimal (complete and deterministic) automaton* (cf. [4]) recognizing $L_{Conn,k}$ is a two-level exponential in k (by Example 4.54.4 of [9]). 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 by introducing fly-automata).

By Definition 10, the language $L_{P(X_1,...,X_n),k}$ depends only on property $P(X_1, ..., X_n)$ and not on the logical language in which it is expressed. However, if $P(X_1, ..., X_n)$ is monadic second-order expressible, say by $\varphi(X_1, ..., X_n)$, then $L_{P(X_1,...,X_n),k}$ is defined by a unique minimal automaton. The construction of an automaton recognizing $L_{P(X_1,...,X_n),k}$ uses an induction on the structure of φ. It may happen that, even if the minimal automaton of $L_{P(X_1,...,X_n),k}$ is "small", the intermediate steps of its construction involve so large automata that this construction fails. This happens if φ is $\theta \vee \neg\theta$ and $\mathcal{A}_{\theta,k}$ is huge. Of course one can see immediately that $\theta \vee \neg\theta$ is equivalent to $True$, but the same happens for $\theta \vee \neg\theta'$where θ' is equivalent to θ, and this fact is not decidable.

4.1. Basic graph properties

We define a set of basic MS expressible graph properties and we will show later how they can be used to describe more complex properties without introducing quantifier alternations. The properties in Table 1 concern a directed or undirected graph G. (We allow some components of a partition $X_1, ..., X_p$ to be empty. We allow loops in the definition of stability. Hence, a graph with loops can be p-colorable. Property Deg_d concerns undirected graphs.)

This set can of course be extended according to needs. In particular, we could add a directed version of $Path$, the properties that a directed graph is strongly connected or is a directed forest (a disjoint union of directed and rooted trees).

Definition 19. *Monadic second-order sentences over $\mathcal{P}$*

(a) We let $\mathcal{P}$ consists of the graph properties of Table 1, of those defined by atomic formulas (cf. Definition 18(b)) and possibly other MS expressible properties. We denote by $MS(\mathcal{P})$ the set of MS formulas written with $\mathcal{P}$-atomic formulas (Definition 16) and not only with the atomic formulas. Hence, this definition extends the syntax of monadic second-order logic but not its expressive power : every formula of $MS(\mathcal{P})$ can be translated into an equivalent MS formula (routine construction).

(b) We denote by $\exists MS(\mathcal{P})$ the subset of $MS(\mathcal{P})$ consisting of formulas of the form $\exists X_{n+1}, ..., X_{n+p}.\varphi$ where φ is a Boolean combination of $\mathcal{P}$-atomic formulas with free variables in $\{X_1, ..., X_{n+p}\}$. Every such formula is equivalent to a finite disjunction of formulas in $\exists MS(\mathcal{P})$ of the

Property	Description
$Disjoint(X_1, ..., X_p)$	$X_1, ..., X_p$ are pairwise disjoint
$Partition(X_1, ..., X_p)$	$X_1, ..., X_p$ form a partition
St	all edges are loops (G is *stable*)
$St_2(X_1)$	$St[X_1] \wedge Card_2[X_1]$
$Clique$	any two vertices are adjacent
$Link(X_1, X_2)$	there is a edge with tail in X_1 and head in X_2
$Dom(X_1, X_2)$	every vertex of X_1 is the head of an edge with tail in X_2
$Path(X_1, X_2)$	$X_1 \subseteq X_2$, $\|X_1\| = 2$ and the two vertices of X_1 are linked by an undirected path in $G[X_2]$
$Card_p$	$\|V_G\| = p$ (where $p \geq 2$)
$Card_{\leq p}$	$\|V_G\| \leq p$ (where $p \geq 2$)
$Deg_d(X_1, X_2)$	every vertex of X_1 is adjacent to exactly d vertices in X_2
$InDeg_{\leq d}(X_1, X_2)$	every vertex of X_1 is the head of at most d directed edges with tail in X_2
$InDeg_d(X_1, X_2)$	same with "exactly" instead of "at most"
$Conn$	G is connected
$ConnIfDeg_d$	see Section 6.3
$Cycle$	G has undirected cycles
$DirCycle$	G has directed cycles

Table 1: Some basic graph properties

form $\exists X_{n+1}, ..., X_{n+p}.\gamma$ where γ is a conjunction of $\mathcal{P}$-atomic and negated $\mathcal{P}$-atomic formulas with free variables in $\{X_1, ..., X_{n+p}\}$. (One rewrites φ as a disjunction of conjunctions of $\mathcal{P}$-atomic and negated $\mathcal{P}$-atomic formulas and one distributes existential quantifications over disjunctions). Hence, we can focus our attention on the model-checking of these particular formulas.

The following examples show that the sentences in $\exists MS(\mathcal{P})$ can express interesting graph properties.

Example 20. *Vertex colorability.* The property of *p-vertex colorability*, which we abbreviate into *p-Col*, is expressed by the sentence:

$$\exists X_1, ..., X_p \quad (Partition(X_1, ..., X_p) \wedge St[X_1] \wedge ... \wedge St[X_p]) \ .$$

The formula $St[X_i]$ expresses that $G[X_i]$ has no other edges than loops. We denote by $Col(X_1, ..., X_p)$ the property $Partition(X_1, ..., X_p) \wedge St[X_1] \wedge ... \wedge St[X_p]$.

A p-vertex coloring defined by $X_1, ..., X_p$ is *acyclic* if each graph $und(G)[X_i \cup X_j]$ 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:

$$\exists X_1, ..., X_p \ (Col(X_1, ..., X_p) \wedge \ldots \wedge \neg Cycle[X_i \cup X_j] \wedge \ldots)$$

with one formula $\neg Cycle[X_i \cup X_j]$ for each pair i, j such that $1 \leq i < j \leq p$.

Example 21. *Minor inclusion.* Let H be a simple, loop-free and undirected graph with vertex set $\{v_1, ..., v_p\}$. An undirected graph G contains H as a minor if and only if it satisfies the sentence:

$$\exists X_1, ..., X_p \ (Disjoint(X_1, ..., X_p) \wedge Conn[X_1] \wedge ... \wedge Conn[X_p] \wedge ... \wedge Link(X_i, X_j) \wedge ...)$$

where there is one formula $Link(X_i, X_j)$ for each edge of H that links v_i and v_j.

Example 22. *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 loop-free undirected graph has a hole if and only if it satisfies the following sentence:

$$\begin{aligned} \exists X_1, ..., X_5 \ \ & [Disjoint(X_1, ..., X_5) \wedge St[X_1] \wedge St[X_2] \wedge \\ & edg(X_3, X_5) \wedge edg(X_4, X_5) \wedge \neg edg(X_3, X_4) \wedge \\ & \neg Link(X_1, X_4 \cup X_5) \wedge \neg Link(X_2, X_3 \cup X_5) \wedge \\ & Deg_2(X_1, X_2 \cup X_3) \wedge Deg_2(X_2, X_1 \cup X_4) \wedge \\ & Deg_2(X_3, X_1 \cup X_5) \wedge Deg_2(X_4, X_2 \cup X_5)]. \end{aligned}$$

By Proposition 9, for every term $t \in T(F_k^{\mathrm{u}})$, one can construct a term $\bar{t} \in T(F_{2k}^{\mathrm{u}})$ that defines the edge complement of the graph $cval(t)$ (assumed to be loop-free). Hence, $cval(t)$ is perfect if and only if the F_{2k}^{u}-automaton for holes rejects both t and $\bar{t}$. The algorithm of [2] can test if a graph is perfect in time $O(n^9)$ (n is the number of vertices). The above logical expression of holes, Theorem 1 stated in the Introduction and the remarks on the parsing problem after Definition 5 provide fixed-parameter *linear* and *cubic* algorithms for testing perfectness with respective parameters tree-width and clique-width.

Example 23. *Existence of induced chordless cycles.*

Chordal graphs are perfect graphs that have several equivalent characterizations. One of them states that they are undirected, simple, loop-free, connected and without any induced cycle C_n for $n \geq 4$. The existence of such a cycle is expressed by the sentence:

$$\exists X, Y \ (St[X \cap Y] \wedge Path(X \cap Y, X) \wedge Path(X \cap Y, Y)).$$

We can "optimize" it by noting that the validity of $Path(X \cap Y, X)$ implies that $X \cap Y$ has cardinality 2. Hence the condition $St[X \cap Y]$ can be replaced by $St_2(X \cap Y)$ where $St_2(Z)$ means that Z is stable and has two elements. This is interesting because a F_k^{u}- automaton for this property with only $(k^2 + 3k + 4)/2$ states (instead of $2^k + 1$ for $St[Z]$; see Section 5.2.1 below) can be easily constructed. Hence, by this observation, we get (before minimization), a smaller automaton for the nonchordality of connected, simple, undirected and loop-free graphs.

Example 24. *Constrained domination* and other problems.

Let $P(X_1)$ be a graph property. 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*. (A vertex dominates itself). Many vertex *partitioning problems* considered in [28] can be expressed by sentences of $\exists MS(\mathcal{P})$ in similar ways.

Examples 20-24 motivate the inclusion of $Partition(X_1, \cdots, X_p)$, $Disjoint(X_1, \cdots, X_p)$, St, $St_2(X_1)$, $Dom(X_1, X_2)$, $Link(X_1, X_2)$, $Deg_2(X_1, X_2)$, $Cycle$ and $Conn$ in our set of basic graph properties.

4.2. From monadic second-order sentences to finite automata

We review the main steps of the inductive construction of a finite automaton associated with a formula of $MS(\mathcal{P})$.

4.2.1. $\mathcal{P}$-atomic formulas

We assume that for each property $P(X_1, ..., X_m)$ of $\mathcal{P}$ and each k, we have defined a finite $F_k^{(m)}$-automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ that accepts the set of terms $L_{P(X_1,...,X_m),k}$. Actually, in all constructions given below in Section 5, these automata depend on k in a uniform way (see Section 7 for the use of this observation). By Lemmas 13, 15 and the inverse image construction (see Definition 17(h)), for set terms $S_1, ..., S_m, S_{m+1}$ over $\{X_1, ..., X_n\}$ and from a finite automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ that accepts $L_{P(X_1,...,X_m),k}$, one gets easily automata $\mathcal{A}_{P(S_1,...,S_m),(X_1,...,X_n),k}$

and $\mathcal{A}_{P(S_1,...,S_m)[S_{m+1}],(X_1,...,X_n),k}$ with same number of states (but more transitions in most cases) that accept $L_{P(S_1,...,S_m),(X_1,...,X_n),k}$ and $L_{P(S_1,...,S_m)[S_{m+1}],(X_1,...,X_n),k}$ respectively. These automata are also deterministic if $\mathcal{A}_{P(X_1,...,X_m),k}$ is deterministic. Hence, we obtain finite automata for all $\mathcal{P}$-atomic formulas.

We will construct finite automata $\mathcal{A}_{P(X_1,...,X_m),k}$ for the properties P of Table 1. In most cases, these automata will be deterministic and complete, not minimal and even, they will have inaccessible states. We wish to have easy descriptions of the transitions rather than small numbers of states. Section 7 will justify this choice. Note that if $\varphi(X_1,...,X_m)$ is written with the atomic formulas $lab_a^{\forall}$ for some a in $[p]$, then the automaton $\mathcal{A}_{\varphi(X_1,...,X_m),k}$ is well-defined only if $k \geq a$ for all such a's. We will always assume this. However, in most cases, the properties to check will not depend on port labels.

4.2.2. Boolean combinations

Lemma 25. (1) *If φ is a conjunction of formulas $\alpha_1,...,\alpha_d$ for which we have constructed finite $F_k^{(m)}$-automata $\mathcal{A}_1,...,\mathcal{A}_d$ with respectively $N_1,...,N_d$ states, we can construct a product $F_k^{(m)}$-automaton for φ with $N_1 \times ... \times N_d$ states. It is deterministic if $\mathcal{A}_1,...,\mathcal{A}_d$ are so.*

(2) *If φ is a disjunction of formulas $\alpha_1,...,\alpha_d$ for which we have constructed finite pairwise disjoint $F_k^{(m)}$-automata $\mathcal{A}_1,...,\mathcal{A}_d$ with respectively $N_1,...,N_d$ states, we can construct a (nondeterministic) $F_k^{(m)}$-automaton for φ with $N_1 + ... + N_d$ states.*

(3) *From a complete and deterministic finite $F_k^{(m)}$-automaton $\mathcal{A}$ for a formula α, we get a complete and deterministic $F_k^{(m)}$-automaton $\mathcal{B}$ for $\neg\alpha$ having the same states and transitions.*

Proof. These are classical constructions (cf. [4]). In Case (3), we take $Acc_{\mathcal{B}} := Q_{\mathcal{A}} - Acc_{\mathcal{A}}$. If $\mathcal{A}$ has a state *Error*, we rename it into *Success* in $\mathcal{B}$ and *vice-versa* (cf. Definition 17(e)). □

By trimming these automata, we may reduce their sizes. Hence, $N_1 \times ... \times N_d$ is an upper bound in the construction of Lemma 25(1). If, for example, $d = 2$, α_1 implies α_2 and $\mathcal{A}_1$ and $\mathcal{A}_2$ are trim and deterministic with respectively N_1 and N_2 states, then the product construction for $\alpha_1 \wedge \alpha_2$ yields an automaton with $N_1 \times N_2$ states. The corresponding trim automaton is isomorphic to $\mathcal{A}_1$, hence has only N_1 states.

4.2.3. Existential quantifications

We denote (in the same way for all n) by $pr^{(m)}$ the relabellings: $F_k^{(n+m)} \to F_k^{(n)}$ that delete the last m Booleans of the sequence w in the nullary symbols $(\mathbf{a}, w)$ and $(\mathbf{a}^\ell, w)$.

Lemma 26. *Let θ be the formula $\exists X_{n+1},...,X_{n+m}.\varphi$ with free variables in $\{X_1,...,X_n\}$.*

(1) *Then $L_{\theta(X_1,...,X_n),k} = pr^{(m)}(L_{\varphi(X_1,...,X_{n+m}),k})$.*

(2) *If $\mathcal{A}$ is a finite $F_k^{(n+m)}$-automaton that recognizes $L_{\varphi(X_1,...,X_{n+m}),k}$, then the $F_k^{(n)}$-automaton $\mathcal{B} := pr^{(m)}(\mathcal{A})$ recognizes $L_{\theta(X_1,...,X_n),k}$. It is not deterministic in general, even if $\mathcal{A}$ is. If $\mathcal{A}$ is deterministic, then $\mathcal{B}$ is nondeterministic with 2^m transitions associated with each nullary symbol $(\mathbf{a}, w)$ or $(\mathbf{a}^\ell, w)$, and all its other transitions are deterministic.*

Proof. Follows easily from the definitions. □

Lemmas 13, 15, 25, 26 and the automata constructions recalled in Definition 17 entail the following result.

Theorem 27. *Let $\mathcal{P}$ be a set of basic MS graph properties for which $F_k^{(.)}$-automata are known for all k. For every k and every $MS(\mathcal{P})$ sentence φ, one can define a finite F_k-automaton $\mathcal{A}_{\varphi,k}$ that accepts the regular language $L_{P,k} = L_{\varphi,k}$ where P is the graph property expressed by φ.*

The automata for the atomic formulas and the properties of $\mathcal{P}$ will be detailed in the next section. Some of them have very large sizes because of the alternations of quantifications that impose nested determinizations, and this is unavoidable. We will focus our attention on formulas with few quantifier alternations.

Corollary 28. *Let $\mathcal{P}$ be a set of basic graph properties. For each k and each property $P(Y_1, ..., Y_m)$ that belongs to $\mathcal{P}$ or is the negation of a property in $\mathcal{P}$, we assume that we have defined a finite $F_k^{(m)}$-automaton $\mathcal{A}_{P(Y_1,...,Y_m),k}$ with $N(P,k)$ states (that need not be deterministic). For every sentence θ of the form $\exists X_1, ..., X_n.\varphi$ where φ is a conjunction of $\mathcal{P}$-atomic and negated $\mathcal{P}$-atomic formulas $\alpha_1, ..., \alpha_d$, we can construct a finite F_k-automaton $\mathcal{A}_{\theta,k}$ with $N_1 \times ... \times N_d$ states where, for each i, $N_i := N(P_i, k)$ and P_i is the property used to define α_i.*

If $\mathcal{A}_{P_i(Y_1,...,Y_m),k}$ is complete and deterministic, it can be used (up to the toggling of accepting states and/or taking an inverse image) for α_i of the form either $P_i(S_1, ..., S_m)$ or $\neg P_i(S_1, ..., S_m)$ (and similarly for $P_i(S_1, ..., S_m)[S_{m+1}]$). If $\mathcal{A}_{P_i(Y_1,...,Y_m),k}$ is not deterministic, it can be used for $P_i(S_1, ..., S_m)$. In order to avoid a determinization, we may use in some cases a nondeterministic automaton $\mathcal{A}_{\neg P_i(Y_1,...,Y_m),k}$ with much less states than $\mathcal{A}_{P_i(Y_1,...,Y_m),k}$. We will do this for connectedness : see Sections 6.1 and 6.4.

4.2.4. Irredundant vs. annotated terms

For certain graph properties $P(X_1, ..., X_m)$, the automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ is quite complicated whereas there exists a simpler (and smaller) automaton $\mathcal{B}_{P(X_1, ...,X_m),k}$ that works correctly on irredundant terms. We mean by this that:

$$L(\mathcal{B}_{P(X_1,...,X_m),k}) \cap IT(F_k^{(m)}) = L_{P(X_1,...,X_m),k} \cap IT(F_k^{(m)}).$$

The algorithm of Proposition 8(1) transforms in linear time a term in $T(F_k)$ into an equivalent one in $IT(F_k)$. Hence, we can design automata intended to work correctly on irredundant terms only. Using such automata will not affect the fixed-parameter tractability results of Theorem 1 (stated in the introduction).

We recall from Definition 7 that the notion of redundancy does not depend on the nullary symbols. Hence, a term t in $IT(F_k^{(m)})$ is irredundant if and only if the term $pr^{(m)}(t)$ (belonging to $T(F_k)$) is irredundant. For the same reasons the relabellings of Lemmas 13 and 15 preserve redundancy and irredundancy. Hence, all constructions of Sections 4.2.1 to 4.2.3 apply to automata for irredundant terms.

The transformation of Proposition 8(1) is based on the annotation ADD_t on a term t. The idea of this transformation is to replace an edge addition f at an occurrence u by an identity if

some operation above u defines all edges defined by f. Instead of transforming t in this way into an equivalent irredundant term, we transform as follows an $F_k^{(m)}$-automaton $\mathcal{B} := \mathcal{B}_{P(X_1,...,X_m),k}$ built for irredundant terms, so as to make it work correctly on annotated terms:

> we replace every transition of $\mathcal{B}$ of the form $\overrightarrow{add}_{a,b}[p] \to q$ by the transitions of the following forms:

- $(\overrightarrow{add}_{a,b}, R)[p] \to p$ for each R containing (a, b), (the new transition acts as if $\overrightarrow{add}_{a,b}$ had been replaced by an identity operation),
- $(\overrightarrow{add}_{a,b}, R)[p] \to q$ for all R not containing (a, b).

The transitions relative to the other symbols are not modified. In a run on a term t, the component R at a node u takes for value the set $ADD_t(u)$. This transformation of automata simulates the replacement of $\overrightarrow{add}_{a,b}$ at u by the identity in the proof of Proposition 8(1) in the case where R contains (a, b).

If $\mathcal{B}$ is an $F_k^{\mathrm{u}(m)}$-automaton, then we replace $add_{a,b}[p] \to q$ by the transitions $(add_{a,b}, R)[p] \to p$ for all R containing $\{a, b\}$ and by $(add_{a,b}, R)[p] \to q$ for all R not containing $\{a, b\}$. (We recall that for $t \in T(F_k^{\mathrm{u}})$, each $ADD_t(u)$ is a set of two labels from $[k]$). In both cases, letting $\mathcal{B}'$ be obtained from $\mathcal{B}$, we have, for every $t \in T(F_k^{(m)}) \cup T(F_k^{\mathrm{u}(m)})$:

$$t \in L_{P(X_1,...,X_m),k} \text{ if and only if } (t, ADD_t) \in L(\mathcal{B}').$$

5. Automata for basic properties

Our objective is now to detail some constructions of finite automata for the atomic formulas (Definition 18(b)) and the basic properties of Table 1 (Section 4.1). Our method makes it possible to prove that the constructions are correct, even if we do not give full proofs. In this section all automata will be finite, hence, *automaton* will mean *finite automaton*.

5.1. First constructions

5.1.1. Easy cases

We consider the atomic formulas and the basic properties that do not depend on adjacency or port labels:

> $Card_p, Card_{\leq p}, X_1 = \emptyset, Sgl(X_1), Card_{p,q}(X_1), X_1 \subseteq X_2, X_1 = X_2, Partition(X_1, ..., X_m)$ and $Disjoint(X_1, ..., X_m)$.

Table 2 shows the characterizations of the corresponding languages (over F_k, $F_k^{(1)}$, $F_k^{(2)}$ or $F_k^{(m)}$) and, in each case, the number of states of the minimal complete and deterministic automaton. (We recall that $\mathbf{C} := \{\mathbf{a}, \mathbf{a}^\ell \mid a \in C := [k]\}$). In this table, we use the following notation:

$|t|_A$ is the number of occurrences in t of symbols from a set A,

$\mathbf{C}' := \{(\mathbf{c}, 1) \mid \mathbf{c} \in \mathbf{C}\}$,

$\mathbf{C}_{i\setminus j} := \{(\mathbf{c}, w) \mid \mathbf{c} \in \mathbf{C} \text{ and } w[i] = 1, w[j] = 0\}$,

$\mathbf{C}_{ij} := \{(\mathbf{c}, w) \mid \mathbf{c} \in \mathbf{C} \text{ and } w[i] = w[j] = 1\}$,

$\mathbf{C}_0 := \{(\mathbf{c}, w) \mid \mathbf{c} \in \mathbf{C} \text{ and } w[i] = 0 \text{ for all } i\}$.

P	$t \in L_{P,k}$	$\sharp\mathcal{A}_{P,k}$
$X_1 = \emptyset$	$\|t\|_{\mathbf{C}'} = 0$	2
$Sgl(X_1)$	$\|t\|_{\mathbf{C}'} = 1$	3
$X_1 \subseteq X_2$	$\|t\|_{\mathbf{C}_{1\setminus 2}} = 0$	2
$X_1 = X_2$	$\|t\|_{\mathbf{C}_{1\setminus 2}} = \|t\|_{\mathbf{C}_{2\setminus 1}} = 0$	2
$Card_p$	$\|t\|_{\mathbf{C}} = p$	$p+2$
$Card_{\leq p}$	$\|t\|_{\mathbf{C}} \leq p$	$p+2$
$Card_{p,q}(X_1)$	$\|t\|_{\mathbf{C}'} = p \bmod q$	q
$Partition(X_1, ..., X_m)$	$\|t\|_{\mathbf{C}_0} = \|t\|_{\mathbf{C}_{ij}} = 0$, all $i \neq j$	2
$Disjoint(X_1, ..., X_m)$	$\|t\|_{\mathbf{C}_{ij}} = 0$, all $i \neq j$	2

Table 2: Languages for some atomic and basic formulas

These automata are straightforward to construct. The next ones will not be so easy. We introduce a way to describe the meaning of their states.

Let $\mathcal{A}$ be a complete deterministic $F_k^{(m)}$-automaton. Each term $t * (V_1, ..., V_m)$ in $T(F_k^{(m)})$ belongs to $L(\mathcal{A}, q)$ for a unique state q. We will say that a property P_q of $(t, V_1, ..., V_m)$ *characterizes* an accessible state q if, for every $t * (V_1, ..., V_m) \in T(F_k^{(m)})$:

$$t * (V_1, ..., V_m) \in L(\mathcal{A}, q) \text{ if and only if } P_q(t, V_1, ..., V_m) \text{ holds.} \quad (1)$$

We will actually specify automata by defining their states and the characteristic properties of their accessible states. Provided the states are informative enough, the constructions of the transitions will follow easily. We will specify in most cases deterministic automata that are complete (each with an *Error* state). This state and the corresponding transitions may be omitted, but they are needed for the complementation operation (Lemma 25(3)).

5.1.2. Adjacency

We first construct the deterministic and complete automaton $\mathcal{A}_{edg(X_1,X_2),k}$ denoted below by $\mathcal{A}$. Its set of states (where $C := [k]$) is:

$$Q := \{0, Ok, Error\} \cup \{a(1), a(2), ab \mid a, b \in C, a \neq b\}.$$

The meaning of each state $q \in Q$ is described in Table 3 in terms of its characteristic property[10] $P_q(t, V_1, V_2)$. The automaton $\mathcal{A}$ will be constructed so that Condition (1) holds for all accessible states q and terms $t * (V_1, V_2) \in T(F_k^{(2)})$.

The transitions are in Table 4. We want $\mathcal{A}$ to be complete with *Error* as nonaccepting sink: the transitions not listed in this table yield *Error*. (This will be the same for all automata defined below and similarly for the accepting sink *Success*). The accepting state is *Ok*. One can prove that the resulting automaton satisfies the conditions of Table 4. Without giving a detailed proof, we indicate the method for doing it.

Let $\mathcal{A}$ be defined by the transitions of Table 4 (and the above convention about *Error*). For proving the correctness of $\mathcal{A}$, it suffices to prove, by induction on t, that for all $t*(V_1, V_2) \in T(F_k)$ and states q:

$$t * (V_1, V_2) \in L(\mathcal{A}, q) \text{ implies that } P_q(t, V_1, V_2) \text{ holds.} \tag{2}$$

which is one direction of Equivalence (1). Since $\mathcal{A}$ is complete and deterministic, and since the properties P_q are mutually exclusive and cover all cases, we get the opposite implications. The choice of *Ok* as single accepting state is then correct by the conditions of Table 4.

The proof of correctness proceeds as follows. The implication (2) holds if t is a nullary symbol. A typical case of an inductive step is when $t = t_1 \oplus t_2$. Assume that $t * (V_1, V_2) \in L(\mathcal{A}, ab)$. Then, the last transition used is one of:

$$\oplus[a(1), b(2)] \to ab, \oplus[b(2), a(1)] \to ab, \oplus[ab, 0] \to ab \quad \text{or} \quad \oplus\,[0, ab] \to ab.$$

In the first case, this means (by using the induction hypothesis) that $V_1 = \{v_1\}$, $V_2 = \{v_2\}$, v_1 is an a-port of $cval(t)/u_1$ and v_2 is a b-port of $cval(t)/u_2$. By the definition of $\oplus$, there is no edge from v_1 to v_2, and so $P_{ab}(t, V_1, V_2)$ is true. The other proofs are similar.

Among the transitions to *Error*, we have the following ones: $\oplus[Ok, Ok] \to Error$, $(\mathbf{a}, 11) \to Error$ and $relab_h[ab] \to Error$ if $h(a) = h(b)$. (Note that *Ok* is *not* an accepting sink. Otherwise, we would call it *Success*.) The number of states is $k^2 + k + 3$ (and Table 4 specifies $O(k^4)$ transitions, with or without counting the transitions to *Error*).

State q	Property $P_q(t, V_1, V_2)$
0	$V_1 = V_2 = \emptyset$
Ok	$V_1 = \{v_1\}, V_2 = \{v_2\}, edg_{cval(t)}(v_1, v_2)$ for some v_1, v_2 in $V_{cval(t)}$
$a(1)$	$V_2 = \emptyset, V_1 = \{v_1\}, \pi_{cval(t)}(v_1) = a$
$a(2)$	$V_1 = \emptyset, V_2 = \{v_2\}, \pi_{cval(t)}(v_2) = a$
ab	$V_1 = \{v_1\}, V_2 = \{v_2\}, \pi_{cval(t)}(v_1) = a,$ $\pi_{cval(t)}(v_2) = b, \neg edg_{cval(t)}(v_1, v_2)$
Error	All other cases

Table 3: Meaning of the states of $\mathcal{A}_{edg(X_1,X_2),k}$

[10] Since in a state ab we have by definition $a \neq b$, the characteristic property P_{ab} implies $v_1 \neq v_2$.

Transitions	Conditions
$\varnothing \to 0$ $(\mathbf{c}, 00) \to 0$ $(\mathbf{c}, 10) \to a(1)$ $(\mathbf{c}, 01) \to a(2)$	$\mathbf{c}$ is $\mathbf{a}$ or $\mathbf{a}^\ell$
$(\mathbf{a}^\ell, 11) \to Ok$	
$relab_h[q] \to q$	$q \in \{0, Ok\}$
$relab_h[a(1)] \to b(1)$ $relab_h[a(2)] \to b(2)$	$b = h(a)$
$relab_h[ab] \to cd$	$c = h(a), d = h(b), c \neq d$
$\overrightarrow{add}_{a,b}[q] \to q$ $\overrightarrow{add}_{a,b}[ab] \to Ok$	$q \neq ab$
$\oplus[a(1), b(2)] \to ab$ $\oplus[b(2), a(1)] \to ab$ $\oplus[q, 0] \to q$ $\oplus[0, q] \to q$	$a \neq b$

Table 4: The transitions of $\mathcal{A}_{edg(X_1,X_2),k}$

For checking properties of undirected graphs defined by terms in $T(F_k^{\mathrm{u}})$, we can use $add_{a,b}$ (for $a < b$) instead of the composition of $\overrightarrow{add}_{a,b}$ and $\overrightarrow{add}_{b,a}$. We modify $\mathcal{A}_{edg(X_1,X_2),k}$ as follows: we replace everywhere ba by ab if $a < b$, and we replace the transitions relative to $\overrightarrow{add}_{a,b}$ by the transitions: $add_{a,b}[ab] \to Ok$ and $add_{a,b}[q] \to q$ for $q \neq ab$. The number of states is then $k(k+3)/2 + 3$.

A smaller automata for annotated terms

If the input terms are annotated as in Definition 6, then $\mathcal{A}$ can be replaced by a smaller automaton $\mathcal{A}'$ with set of states $Q' := \{0, Ok, Error\} \cup \{a(1), a(2) \mid a \in C\}$ of cardinality $2k+3$. Consider for example an occurrence u of $\oplus$ where the transition $\oplus[a(1), b(2)] \to ab$ is used by $\mathcal{A}$ (so that $run_{\mathcal{A},t}(u) = ab$). If $(a, b) \in ADD_t(u)$, then we can replace the state ab by Ok because we are certain that $v_1 \to v_2$ in $cval(t)$, although this is not yet the case in $cval(t)/u$. If $(a, b) \notin ADD_t(u)$, we can replace the state ab by $Error$.

The automaton $\mathcal{A}'$ uses only the annotation at the occurrences of $\oplus$. Its transitions are shown in Table 5. In this table, a pair $(\oplus, R)$ is used as a function symbol that occurs at a node u if and only if $R = ADD_t(u)$.

Since the annotation ADD_t does not depend on nullary symbols as observed at the end of Definition 6, the direct and inverse image constructions of Section 3 that are useful to implement Lemmas 13 and 15 and to prove Theorem 27 work for automata on terms annotated in this way.

5.1.3. Labels

For completeness sake, we consider the atomic formulas $lab_a^\forall(X_1)$ for $a \in C$, although in most cases, the formulas to be checked express properties of graphs (not of p-graphs) and thus

Transitions	Conditions
$\varnothing \to 0$ $(\mathbf{c}, 00) \to 0$ $(\mathbf{c}, 10) \to a(1)$ $(\mathbf{c}, 01) \to a(2)$	$\mathbf{c}$ is $\mathbf{a}$ or $\mathbf{a}^\ell$
$(\mathbf{a}^\ell, 11) \to Ok$	
$relab_h[q] \to q$	$q \in \{0, Ok\}$
$relab_h[a(1)] \to b(1)$ $relab_h[a(2)] \to b(2)$	$b = h(a)$
$\overrightarrow{add}_{a,b}[q] \to q$	
$(\oplus, R)[a(1), b(2)] \to Ok$ $(\oplus, R)[b(2), a(1)] \to Ok$ $(\oplus, R)[q, 0] \to q$ $(\oplus, R)[0, q] \to q$	$(a, b) \in R$ $(a, b) \in R$

Table 5: The transitions of $\mathcal{A}'$ for annotated terms

do not use these atomic formulas. The set of states is $Q := \mathcal{P}(C)$, the set of subsets of C. The characteristic property of a state A is defined by :

$P_A(t, V_1)$ if and only if $A = \pi_{cval(t)}(V_1)$.

The transitions are straightforward to write. The state $\emptyset$ is accessible because t can denote the empty graph. The accepting states are $\emptyset$ and $\{a\}$. (This automaton computes actually the set of port labels of the vertices of V_1.)

5.2. Other basic properties

5.2.1. Stability

We recall that stability, denoted by St, means that all edges are loops. The empty graph is stable. We define an F_k-automaton $\mathcal{A}_{St,k}$ with set of states $Q := \mathcal{P}(C) \cup \{Error\}$ of cardinality $2^k + 1$. The characteristic properties of its accessible states are in Table 6.

State q	Property $P_q(t)$
$Error$	$cval(t)$ is not stable
A	$cval(t)$ is stable and $A = \pi(cval(t))$

Table 6: Meaning of the states of $\mathcal{A}_{St,k}$

All states are accepting except $Error$. The transitions are easy to define on the basis of the following facts: for all p-graphs G and H and operations $relab_h$, $add_{a,b}$ and $\overrightarrow{add}_{a,b}$:

$G \oplus H$ is stable if and only if G and H are stable,
$relab_h(G)$ is stable if and only if G is stable,
$add_{a,b}(G)$ and $\overrightarrow{add}_{a,b}(G)$ are stable if and only if G is stable and a and b are not both in $\pi(G)$,
G is stable if it is defined by $\mathbf{a}, \mathbf{a}^\ell$ or $\varnothing$.

For the variant where stability forbids loops, each nullary symbols $\mathbf{a}^{\ell}$ yields $Error$.

We recall from Lemma 15 that we can easily obtain from $\mathcal{A}_{St,k}$ an automaton for $St[X_1]$ expressing that the induced subgraph of G with vertex set X_1 is stable.

Annotations are useful in the following way: if the automaton $\mathcal{A}_{St,k}$ reaches a state A at a node u of the tree t such that $a, b \in A$ and (a, b) or $\{a, b\}$ belongs to $ADD_t(u)$, then this state can be replaced by $Error$ because some edge-addition operation above u in t creates at least one edge with two ends in $cval(t)/u$, so that $cval(t)$ is not stable. The computation of the modified automaton can thus be faster than that of $\mathcal{A}_{St,k}$ on the terms to be rejected.

5.2.2. Clique

We let $Clique$ be the property that any two distinct vertices are linked by an edge (in any direction). The empty graph is a clique. As above, we define a complete and deterministic F_k-automaton. Its states are the pairs (A, R) such that $A \subseteq C$ and $R \subseteq A \times A$. The characteristic properties of the accessible states are in Table 7. The accepting states are the pairs $(A, \emptyset)$. We give only two transitions :

State q	Property $P_q(t)$
(A, R)	$A = \pi(cval(t))$ and R is the set of pairs $(a, b) \in A \times A$ such that $cval(t)$ has an a-port x and a b-port y such that $(x, y) \notin edg_G \cup edg_G^{-1}$

Table 7: Meaning of the states of $\mathcal{A}_{Clique,k}$

$$\oplus[(A, R), (A', R')] \to (A \cup A', R \cup R' \cup (A \times A') \cup (A' \times A)),$$
$$\overrightarrow{add}_{a,b}(G)[(A, R)] \to (A, R - \{(a, b), (b, a)\}).$$

The number of states is less than 2^{k^2+k} but certain states are inaccessible, in particular, those such that R is not symmetric. The minimal automaton has $2^{\Theta(k^2)}$ states as one can check easily. We can add a state $Error$ and the transitions:

$$\oplus[(A, R), (A', R')] \to Error, \text{ if } A \cap A' \neq \emptyset,$$
$$relab_h[(A, R)] \to Error \text{ if } h(a) = h(b) \text{ for some } a \text{ and } b \neq a \text{ in } A,$$

because if a graph has two vertices with same label that are not adjacent, then no sequence of operations applied to this graph can create an edge between these two vertices. The states (A, R) such that $(a, a) \in R$ become inaccessible and the resulting automaton is slightly smaller than the previous one, but the same lower bound applies to the number of states.

An automaton for annotated terms (cf. Definition 6), that has only $2^k + 1$ states can be built: its states are $Error$ and A for all $A \subseteq C$; it reaches the state $Error$ at u in t whenever $add_{ADD_t(u)}(cval(t)/u)$ does not satisfy $Clique$ and otherwise, it reaches the state $A = \pi(cval(t)/u)$. (We recall from Proposition 9 that add_R is the composition of the operations $add_{a,b}$ for all $\{a, b\} \in R$; if R is a set of pairs (a, b) with $a \neq b$, then add_R is the composition of the operations $\overrightarrow{add}_{a,b}$ for all (a, b) in R).

5.2.3. Set adjacency

The automaton for the property $Link(X_1, X_2)$ that there is a edge with tail in X_1 and head in X_2 has $2^{2k} + 1$ states that are the pairs (A, B) for all $A, B \subseteq C$ together with an accepting sink *Success*. All these states are accessible and their characteristic properties are defined in Table 8.

State q	Property $P_q(t, V_1, V_2)$
(A, B)	$A = \pi_{cval(t)}(V_1)$, $B = \pi_{cval(t)}(V_2)$, and $Link(V_1, V_2)$ does not hold in $cval(t)$
Success	$Link(V_1, V_2)$ holds in $cval(t)$

Table 8: Meaning of the states of $\mathcal{A}_{Link(X_1,X_2),k}$

The automaton is complete. The only accepting state is the sink *Success*. If the automaton is intended to process annotated terms t, it can replace by *Success* any state (A, B) at a node u if $ADD_t(u) \cap (A \times B) \neq \emptyset$ (or $\{a, b\} \in ADD_t(u)$ for some $a \in A$ and $b \in B$ in case of terms defining undirected graphs) and thus accept quickly.

5.2.4. Domination

We consider the property $Dom(X_1, X_2)$ (meaning that every vertex in X_1 is the head of an edge with tail in X_2). The automaton has $2^{2k} + 1$ states which are *Error* and the pairs (A, B) for all $A, B \subseteq C$. Their characteristic properties are in Table 9.

State q	Property $P_q(t, V_1, V_2)$
(A, B)	$V_1 \cap V_2 = \emptyset$, $B = \pi_{cval(t)}(V_2)$, A is the set of port labels of the vertices in V_1 that are not the head of an edge with tail in V_2
Error	$V_1 \cap V_2 \neq \emptyset$

Table 9: Meaning of the states of $\mathcal{A}_{Dom(X_1,X_2),k}$

The accepting states are the pairs $(\emptyset, B)$, hence *Error* is not the only nonaccepting state. Annotations do not seem to help in this case.

5.2.5. Paths

The following construction concerns paths in undirected graphs, hence we construct an F_k^{u}-automaton. If we want to check the corresponding property of *undirected paths* in directed graphs, i.e., of paths where edges can be traversed in either direction, it suffices to replace $add_{a,b}$ by $\overrightarrow{add}_{a,b}$ and by $\overrightarrow{add}_{b,a}$ in the transitions of the F_k^{u}-automata we will construct.

We recall that $Path(X_1, X_2)$ holds in G if and only if $X_1 \subseteq X_2$, $|X_1| = 2$ and there is an undirected path in $G[X_2]$ that links the two vertices of X_1. This property is monadic second-order expressible (we will write below the formula) but we will construct an $F_k^{\mathrm{u}(2)}$-automaton $\mathcal{A}_{Path(X_1,X_2),k}$ without using the logical expression of $Path(X_1, X_2)$.

We need some auxiliary notions. Let G be an undirected p-graph with ports in a finite fixed set C. For $x \in V_G$, we let[11]:

$$\alpha(G,x) := \{\pi_G(y) \mid y \in V_G \text{ and } x -\!\!-_G^* y\} \subseteq C,$$

and

$$\beta(G) := \{(\pi_G(x), \pi_G(y)) \mid x, y \in V_G, x -\!\!-_G^* y\} \subseteq C \times C.$$

Hence $\beta(G)$ is a symmetric and reflexive relation on the set $\pi(G)$. It determines $\pi(G)$.

We will prove that the functions α and β can be computed inductively on t if $G = cval(t)$. If $h : C \to C$ and $B \subseteq C \times C$, then $h(B) := \{(h(a), h(b)) \mid (a,b) \in B\}$. We extend the composition of binary relations (denoted by $\cdot$) to $A \subseteq C$ and $B \subseteq C \times C$ by defining $A \odot B$ as the set $\{b \in C \mid (a,b) \in B \text{ for some } a \in A\}$. It is the image of A under the multivalued function defined by B. For $a, b \in C$ with $b \neq a$, we let $a \circledast b$ denote the relation $\{(a,a),(a,b),(b,a),(b,b)\}$.

Lemma 29. *For disjoint p-graphs G and H and vertex x, we have:*

(1) $\alpha(G \oplus H, x) = \begin{cases} \alpha(G,x) & \text{if } x \in V_G, \\ \alpha(H,x) & \text{if } x \in V_H. \end{cases}$

(2) $\beta(G \oplus H) = \beta(G) \cup \beta(H)$.

(3) $\alpha(add_{a,b}(G), x) = \begin{cases} \alpha(G,x) \cup \alpha(G,x) \odot ((a \circledast b) \cdot \beta(G)) & \text{if } a, b \in \pi(G), \\ \alpha(G,x) & \text{otherwise.} \end{cases}$

(4) $\beta(add_{a,b}(G)) = \begin{cases} \beta(G) \cup \beta(G) \cdot (a \circledast b) \cdot \beta(G) & \text{if } a, b \in \pi(G), \\ \beta(G) & \text{otherwise.} \end{cases}$

(5) $\alpha(relab_h(G), x) = h(\alpha(G,x))$.

(6) $\beta(relab_h(G)) = h(\beta(G))$.

(7) $\alpha(\mathbf{a}, x) = \alpha(\mathbf{a}^\ell, x) = \{a\}$.

(8) $\beta(\mathbf{a}) = \beta(\mathbf{a}^\ell) = \{(a,a)\}$.

(9) $\beta(\varnothing) = \emptyset$.

Proof. The verifications are easy from the definitions. We only sketch the proof of the inclusion $\subseteq$ in (3).

If c is in $\alpha(add_{a,b}(G), x)$, then either it is in $\alpha(G,x)$ or there exists a path from x to a c-port z that uses one or more edges added to G by $add_{a,b}$. If this path contains only one such edge and goes through an a-port u and immediately after through a b-port w, we have $a \in \alpha(G,x)$ and $(b,c) \in \beta(G)$ and thus $c \in \alpha(G,x) \odot ((a \circledast b) \cdot \beta(G))$. If this path contains several such edges, the first one being $u - w$ and the last one being $u' - w'$, where u and u' are a-ports and w and w' are b-ports, then there is also an edge between u and w' and the previous case gives the result. If $u - w$ and $u' - w'$ are as above except that w' is an a-port and u' is a b-port, then there is also an edge between w and w'. Then $a \in \alpha(G,x), (a,c) \in \beta(G)$ and thus $c \in \alpha(G,x) \odot ((a \circledast b) \cdot \beta(G))$ because $(a,a) \in a \circledast b$. □

[11]Here $x -_G y$ means that x and y are adjacent in G and $x -\!\!-_G^* y$ means that they are equal or linked by an undirected path.

We now construct an automaton with set of states Q defined as:

$$\{Ok, Error\} \cup \{(0, B) \mid B \subseteq C \times C\} \cup$$
$$\{(1, A, B) \mid \emptyset \neq A \subseteq C, B \subseteq C \times C\} \cup$$
$$\{(2, \{A, A'\}, B) \mid A, A' \subseteq C, A \neq \emptyset, A' \neq \emptyset, B \subseteq C \times C\}.$$

State q	Property $P_q(t, V_1, V_2)$
$(0, B)$	$V_1 = \emptyset, B = \beta(cval(t)[V_2])$
$(1, A, B)$	$V_1 = \{v\} \subseteq V_2, A = \alpha(cval(t)[V_2], v)$ $B = \beta(cval(t)[V_2])$
$(2, \{A, A'\}, B)$	$V_1 = \{v, v'\} \subseteq V_2, v \neq v'$, $A = \alpha(cval(t)[V_2], v)$, $A' = \alpha(cval(t)[V_2], v')$, $B = \beta(cval(t)[V_2])$ there is no path between v and v' in $cval(t)[V_2]$
Ok	$Path(V_1, V_2)$ holds in $cval(t)$
Error	All other cases

Table 10: Meaning of the states of $\mathcal{A}_{Path(X_1,X_2),k}$

The meaning of its accessible states is described in Table 10. The transitions are shown in Table 11, where we use the following auxiliary functions :

$$f(B, a, b) := \texttt{ if } \{(a, a), (b, b)\} \subseteq B \texttt{ then } B \cup (B \cdot (a \circledast b) \cdot B) \texttt{ else } B,$$

$$g(A, B, a, b) := \texttt{ if } \{(a, a), (b, b)\} \subseteq B \texttt{ then } A \cup (A \odot ((a \circledast b) \cdot R)) \texttt{ else } A.$$

These definitions reflect respectively Properties (4) and (3) of Lemma 29. We take *Ok* as accepting state. This completes the construction.

Table 10 shows that some states are not accessible : for example, by the definition of β, the component B of each accessible state must be a reflexive and symmetric relation. State $(1, \{a\}, \emptyset)$ is not accessible either. The set Q has cardinality $2 + 2^k + 2^{k+k^2} + (2^k(2^k + 1)/2)2^{k^2}$ (where $k \geq 2$). The cardinality of the set of accessible states is somewhat less than that, but it lies between $2^{k^2/2}$ and 2^{k^2+2k} as one can check easily. Determining its exact value is of no interest. Annotations seem to be usable: the set of states remains the same but the transitions are more complicated to define. The benefit of using them is not clear.

Let us now consider the expression of the property $Path(X_1, X_2)$ by the MS formula:

$$\forall x[x \in X_1 \Rightarrow x \in X_2] \wedge$$
$$\exists x, y\{x \in X_1 \wedge y \in X_1 \wedge x \neq y \wedge \forall z(z \in X_1 \Rightarrow x = z \vee y = z) \wedge$$
$$\forall X_3[x \in X_3 \wedge \forall u, v(u \in X_3 \wedge u \in X_2 \wedge v \in X_2 \wedge edg(u, v) \Rightarrow v \in X_3) \Rightarrow y \in X_3]\}$$

of quantifier-height 5. Its translation into a formula without first-order variables and universal quantifiers has the same quantifier-height. The given construction of $\mathcal{A}_{Path(X_1,X_2),k}$ avoids thus

Transitions	Conditions
$(\mathbf{c}, 00) \to (0, \emptyset), \quad \varnothing \to (0, \emptyset)$ $(\mathbf{c}, 01) \to (0, \{(a, a)\})$ $(\mathbf{c}, 11) \to (1, \{a\}, \{(a, a)\})$	$\mathbf{c} \in \{\mathbf{a}, \mathbf{a}^{\ell}\}$
$relab_h[Ok] \to Ok$ $relab_h[(0, B)] \to (0, h(B))$ $relab_h[(1, A, B)] \to (1, h(A), h(B))$ $relab_h[(2, \{A, A'\}, B)] \to (2, \{h(A), h(A')\}, h(B))$	
$add_{a,b}[Ok] \to Ok$ $add_{a,b}[(0, B)] \to (0, B')$ $add_{a,b}[(1, A, B)] \to (1, D, B')$ $add_{a,b}[(2, \{A, A'\}, B)] \to (2, \{D, D'\}, B')$	$B' = f(B, a, b)$ $D = g(A, B, a, b)$ $D' = g(A', B, a, b)$ $(A \odot ((a \circledast b) \circ B)) \cap A' = \emptyset$
$add_{a,b}[(2, \{A, A'\}, B)] \to Ok$	$(A \odot ((a \circledast b) \circ B)) \cap A' \neq \emptyset$
$\oplus[Ok, (0, B)] \to Ok$ $\oplus[(0, B), Ok] \to Ok$ $\oplus[(0, B), (0, B')] \to (0, B'')$ $\oplus[(0, B), (1, A, B')] \to (1, A, B'')$ $\oplus[(1, A, B'), (0, B)] \to (1, A, B'')$ $\oplus[(1, A, B), (1, A', B')] \to (2, \{A, A'\}, B'')$ $\oplus[(0, B), (2, \{A, A'\}, B')] \to (2, \{A, A'\}, B'')$ $\oplus[(2, \{A, A'\}, B'), (0, B)] \to (2, \{A, A'\}, B'')$	$B'' = B \cup B'$

Table 11: The transitions of $\mathcal{A}_{Path(X_1,X_2),k}$

lengthy computations. The minimal automaton equivalent to $\mathcal{A}_{Path(X_1,X_2),k}$ depends only on the property $Path(X_1, X_2)$ (cf. the beginning of Section 4). It is thus the same as the one derivable from any monadic second-order expression of this property (provided the computations do not abort by lack of memory).

5.2.6. Bounded degree and indegree

We first consider directed graphs. The *indegree* of a vertex is the number of incoming edges. The *indegree* of a graph is the maximum indegree of its vertices. We will construct an automaton for the property $InDeg_{\leq d}(X, Y)$ meaning that every vertex in X is the head of at most d edges with tail in Y. Hence, a graph G has indegree at most d if and only if $InDeg_{\leq d}(V_G, V_G)$ is valid.

We will construct an automaton $\mathcal{B}_{InDeg_{\leq d}(X_1,X_2),k}$ intended to work correctly on irredundant terms (cf. Definition 7 and Proposition 8) and that can be easily modified into one with same set of states intended to work on annotated terms (cf. the remarks of Section 4.2.4). For this purpose, we define $\partial_G(x, Y)$ as the number of edges with head x and tail in Y and $\partial(G)$:= $\max\{\partial_G(x, V_G) \mid x \in V_G\}$. For $X, Y \subseteq V_G$, we let, for every a in C :

$$\alpha_G(X, Y)(a) := \max\{\partial_G(x, Y) \mid x \in X \cap \pi_G^{-1}(a)\},$$
$$\beta_G(Y)(a) := \min\{d + 1, \left|Y \cap \pi_G^{-1}(a)\right|\}.$$

Hence, $\alpha_G(X, Y)$ is a mapping : $C \to [0, \partial(G)]$ and $\beta_G(Y)$ is one : $C \to [0, d + 1]$.

The states of $\mathcal{B}_{InDeg_{\leq d}(X_1,X_2),k}$ are $Error$ and the pairs (α, β) where $\alpha : C \to [0, d]$ and $\beta : C \to [0, d+1]$. (See Section 7.3.1 for another description). Their characteristic properties are in Table 12.

State q	Property $P_q(t, V_1, V_2)$
(α, β)	$InDeg_{\leq d}(V_1, V_2)$ holds in $cval(t)$, $\alpha = \alpha_{cval(t)}(V_1, V_2)$ and $\beta = \beta_{cval(t)}(V_2)$
$Error$	$InDeg_{\leq d}(V_1, V_2)$ does not hold in $cval(t)$.

Table 12: Meaning of the states of $\mathcal{B}_{InDeg_{\leq d}(X_1,X_2),k}$

Every state except $Error$ is accepting. The number of states is $(d + 1)^k(d + 2)^k + 1 \leq (d + 2)^{2k} = 2^{2k \log(d+2)}$. The definition of transitions will use the following lemma where, to simplify the notation, we let for every graph H:

$$\alpha_H(X, Y)(a) := \alpha_H(X \cap V_H, Y \cap V_H)(a) \quad \text{and}$$
$$\beta_H(Y)(a) := \beta_H(Y \cap V_H)(a).$$

Similarly, $InDeg_{\leq d}(X, Y)$ is defined as true in H if $InDeg_{\leq d}(X \cap V_H, Y \cap V_H)$ is true (also in H).

Lemma 30. *(1) If $G = G_1 \oplus G_2$ and $X, Y \subseteq V_{G,}$, then for every $a \in C$:*

$\alpha_G(X,Y)(a) = \max\{\alpha_{G_1}(X,Y)(a), \alpha_{G_2}(X,Y)(a)\}$,
$\beta_G(Y)(a) = \min\{d+1, \beta_{G_1}(Y)(a) + \beta_{G_2}(Y)(a)\}$,
$InDeg_{\leq d}(X,Y)$ is true in G if and only if it is true in G_1 and in G_2.

(2) If $G = relab_h(G_1)$ and $X, Y \subseteq V_G$, then for every $a \in C$:

$\alpha_G(X,Y)(a) = \max\{\alpha_{G_1}(X,Y)(b) \mid h(b) = a\}$,
$\beta_G(Y)(a) = \min\{d+1, \sum_{h(b)=a} \beta_{G_1}(Y)(b)\}$,
$InDeg_{\leq d}(X,Y)$ is true in G if and only if it is true in G_1.

(3) Let $G = \overrightarrow{add}_{a,b}(G_1)$ and $X, Y \subseteq V_G$ be such that G_1 has no edge from an a-port to a b-port.

(3.1) For every $c \in C - \{b\}$, we have $\beta_G(Y)(c) = \beta_{G_1}(Y)(c)$ and $\alpha_G(X,Y)(c) = \alpha_{G_1}(X,Y)(c)$.
(3.2) Let $d' := \alpha_{G_1}(X,Y)(b) + \beta_{G_1}(Y)(a)$. If $d' \leq d$, then $\alpha_G(X,Y)(b) = d'$; otherwise, $\alpha_G(X,Y)(b) > d$. Furthermore, $InDeg_{\leq d}(X,Y)$ is true in G if and only if it is true in G_1 and $d' \leq d$.

Proof. All these facts are easy consequences of the definitions. About Assertion (3.2), we observe that $\alpha_G(X,Y)(b) = \alpha_{G_1}(X,Y)(b) + \left|Y \cap \pi_G^{-1}(a)\right|$. If $\alpha_{G_1}(X,Y)(b) + \beta_{G_1}(Y)(a) = d' \leq d$, then $\beta_{G_1}(Y)(a) \leq d$, hence $\beta_{G_1}(Y)(a) = \left|Y \cap \pi_G^{-1}(a)\right|$ and so, we have $\alpha_G(X,Y)(b) = d'$. Otherwise, since $\beta_{G_1}(Y)(a) \leq \left|Y \cap \pi_G^{-1}(a)\right|$ by definition, we have $\alpha_G(X,Y)(b) > d$ if $d' > d$, and so $InDeg_{\leq d}(X,Y)$ is false in G. □

Lemma 30 yields the definitions of the transitions relative to $\oplus, relab_h$ and $\overrightarrow{add}_{a,b}$. In particular $\overrightarrow{add}_{a,b}[\alpha,\beta] \to Error$ if $\alpha(b) + \beta(a) > d$, by Assertion (3). The construction is correct for irredundant input terms, because of the computation of $\alpha_G(X,Y)(b)$ in the proof of this assertion. The transitions relative to nullary symbols are as follows, for $i = 0, 1$:

$(\mathbf{a}, i0) \to (\mathbf{0}, \mathbf{0}), (\mathbf{a}, i1) \to (\mathbf{0}, 1^a)$,
$(\mathbf{a}^\ell, 00) \to (\mathbf{0}, \mathbf{0}), (\mathbf{a}^\ell, 10) \to (1^a, \mathbf{0})$,
$(\mathbf{a}^\ell, 01) \to (\mathbf{0}, 1^a)$ and $(\mathbf{a}^\ell, 11) \to (1^a, 1^a)$

where $\mathbf{0}$ denotes the mapping : $C \to \mathbb{N}$ with constant value 0, and j^a, for $j > 0$, is the one such that $j^a(x) :=$ `if` $x = a$ `then` j `else` 0.

We now extend this construction to undirected graphs. For an undirected graph G without loops, the degree of any vertex is equal to its indegree in the corresponding directed graph G' (cf. Definition 3). If t is an irredundant term over F_k^{u} that defines G, then we replace each operation $add_{a,b}$ in t by $\overrightarrow{add}_{a,b} \circ \overrightarrow{add}_{b,a}$ and we obtain an irredundant term t' for G', so that the previous automaton can be used on t'. However, a loop counts for two edges. Hence, if G has loops, we can do the same but we need to modify the last two types of transitions into:

$(\mathbf{a}^\ell, 01) \to (\mathbf{0}, 2^a)$ and $(\mathbf{a}^\ell, 11) \to (1^a, 2^a)$.

An automaton similar to $\mathcal{B}_{InDeg_{\leq d}(X_1,X_2),k}$ can be constructed for the property $InDeg_d(X, Y)$ meaning that every vertex in X is the head of exactly d edges with tail in Y. It suffices to replace $\alpha_G(X, Y)(a)$ by:

$\varepsilon_G(X, Y)(a) := d$ if $d = \partial_G(x, Y)$ for all x in $X \cap \pi_G^{-1}(a) \neq \emptyset$,
$:= \perp$ if $X \cap \pi_G^{-1}(a) = \emptyset$ ($\perp$ means "undefined"),
$:= \top$ if $\partial_G(x, Y) \neq \partial_G(y, Y)$ for some x, y in $X \cap \pi_G^{-1}(a)$ ($\top$ yields an error).

It is easy to modify $\mathcal{B}_{InDeg_{\leq d}(X_1,X_2),k}$ accordingly. This idea can be used to construct the automata $\mathcal{B}_{Deg_2(X_1,X_2),k}$ with $2^{O(k)}$ states that are useful to check the perfectness of loop-free undirected graphs (cf. Example 22).

Having cycles is an important property for which we give two constructions of automata. This property is useful to express the notions of forests and trees.

5.2.7. Undirected cycles

We consider loop-free, undirected (simple) graphs. The property that a graph has cycles, denoted by $Cycle$, is expressed by the sentence $\exists X.P[X]$ where P expresses that the graph is not empty and all its vertices have degree at least 2. If X is minimal for inclusion with this property, then it is the vertex set of an induced cycle. The minimal cardinality of such a set, called the *girth* of the graph, can thus be computed by finite automata solving optimization problems (cf. Chapter 6 of [9]).

An easy adaptation of the construction of Section 5.2.6 yields a deterministic automaton $\mathcal{A}_{P,k}$ with 9^k states, hence a deterministic automaton $\mathcal{A}_{Cycle,k}$ with 2^{9^k} states. These automata are intended to run on irredundant terms. By the remarks of Section 4.2.4, they can be transformed into automata with same number of states intended to run on annotated terms.

The states of $\mathcal{A}_{P,k}$ are the 4-tuples (A_1, A_2, D_0, D_1) of subsets of $[k]$ such that $A_1 \cap A_2 = \emptyset$ and $D_0 \cup D_1 \subseteq A_1 \cup A_2$. The characteristic property of (A_1, A_2, D_0, D_1) is :

$P_{(A_1,A_2,D_0,D_1)}(t)$ if and only if :

A_1 is the set of port labels having a unique occurrence in $cval(t)$,

A_2 is the set of port labels having at least two occurrences (hence $A_1 \cup A_2 = \pi(cval(t))$),

D_0 is the set of port labels of the isolated vertices,

D_1 is the set of port labels of the degree 1 vertices.

The accepting states are those such that $A_1 \cup A_2 \neq \emptyset$ and $D_0 \cup D_1 = \emptyset$. The transitions (for processing an irredundant term) are easy to write from this description.

As for connectivity, if we want to verify the absence of cycles in graphs of degree at most d, we can use use a smaller automaton with less than $2^{4d \cdot k^2}$ states[12].

[12]Its construction is similar to that for connectedness (cf. Section 6.3) and is based on that of a deterministic automaton for $Cycle$ that has 3^{3^k} states. We can send it to anybody interested.

5.2.8. Directed cycles

The automata for checking the existence of directed cycles are surprisingly smaller than those for undirected cycles. The construction is similar to that for *Paths* in Section 5.2.5 and we will use notation from it. We only consider loop-free graphs.

If G is directed, the notation $x{\to}^{+}_{G}y$ means that there is a directed path from x to y with at least one edge. We may have $x = y$. If G has ports in C, we define: $\beta^{+}(G) := \{(\pi_G(x), \pi_G(y)) \,|\, x{\to}^{+}_{G}y\} \subseteq C \times C$. This function can be computed inductively on t such that $G = cval(t)$. If R is a binary relation on C, we let $g(R, a, b) := (R \cup \{(a, a)\}) \cdot \{(a, b)\} \cdot (R \cup \{(b, b)\})$.

Lemma 31. *For disjoint p-graphs G and H, we have:*

(1) $\beta^{+}(G \oplus H) = \beta^{+}(G) \cup \beta^{+}(H)$.

(2) $\beta^{+}(add_{a,b}(G)) = \begin{cases} g(\beta^{+}(G), a, b) & \text{if } a, b \in \pi(G), \\ \beta^{+}(G) & \text{otherwise.} \end{cases}$

(3) $\beta^{+}(relab_h(G)) = h(\beta^{+}(G))$.

(4) $\beta^{+}(\mathbf{a}) = \beta^{+}(\varnothing) = \emptyset$.

Proof. Similar to that of Lemma 29 in Section 5.2.5. □

State q	Property $P_q(t)$
(A, R)	$A = \pi(cval(t)), R = \beta^{+}(cval(t))$ and there is no directed cycle in $cval(t)$
Success	there is a directed cycle in $cval(t)$

Table 13: Meaning of the states of $\mathcal{A}_{DirCycle,k}$

We now construct a deterministic automaton $\mathcal{A}_{DirCycle,k}$ with set of states $Q = \{(A, R) \,|\, A \subseteq C, R \subseteq A \times A\} \cup \{Success\}$. The input terms need not be irredundant. There are less than 2^{k^2+k} accessible states whose meaning is in Table 13. The sink *Success* is the unique accepting state. In Table 14, all transitions not listed yield *Success*, and since we only consider loop-free graphs, there is no transition for $\mathbf{a}^{\ell}$.

Transitions	Conditions
$\varnothing \to (\emptyset, \emptyset)$ $\mathbf{a} \to (\{a\}, \emptyset)$	
$relab_h[(A, R)] \to (h(A), h(R))$	
$add_{a,b}[(A, R)] \to Success$	$(b, a) \in R$
$add_{a,b}[(A, R)] \to (A, R)$	a and b are not in A
$add_{a,b}[(A, R)] \to g(R, a, b)$	otherwise
$\oplus[(A, R), (A', R')] \to (A \cup A', R \cup R')$	

Table 14: The transitions of $\mathcal{A}_{DirCycle,k}$

The property *DirCycle* is expressed by the sentence $\exists X\ [X \neq \emptyset \wedge Dom(X, X)]$. From this expression, we get a non-deterministic automaton with $O(2^{2k})$ states, and a deterministic one with $2^{O(2^{2k})}$ states. Our direct construction is thus better.

6. Connectedness

Connectedness is an important graph property that is used in the expression of several other properties such as minor inclusion (cf. Example 21) or being a tree. We present in detail several constructions of automata for it. Connectedness does not depend on edge directions: we construct F_k^{u}-automata that can be adapted to directed graphs. (See the beginning of Section 5.2.5).

6.1. "Large" deterministic automata.

We first observe that a graph is *not connected* if and only if it satisfies the property:

$$\exists X\,(X \neq \emptyset \wedge \overline{X} \neq \emptyset \wedge \neg Link(X, \overline{X})).$$

Hence, the associated *nondeterministic* F_k^{u}-automaton, constructed with the tools of Sections 4.2 and 5.2.3 has $O(2^{2k})$ states, which gives a (deterministic) automaton with $2^{O(2^{2k})}$ states. We will do better by constructing a deterministic automaton F_k^{u}-automaton $\mathcal{A}_{Conn,k}$ with less than 2^{2^k} states.

We will use the following notions and notation. If E is a set, we denote by $\mathbb{M}(E)$ the set of finite multisets of elements of E. We denote by $P \uplus P'$ the union of two multisets P and P', by $|P|$ the cardinality of P, so that $|P \uplus P'| = |P| + |P'|$. If $P \subseteq Q$, i.e., if $Q = P \uplus P'$ for some P', we denote this unique P' by $Q - P$. We let $Set(P) \subseteq P$ be the set of elements of E having an occurrence in P, hence, $Set(P) = P$ if and only if P is a set. Finally, we define:

$$Set^\dagger(P) := \texttt{if } Set(P) = \{d\} \texttt{ and } |P| \geq 2 \texttt{ then } \{d, d\} \texttt{ else } Set(P).$$

For all multisets P and P' :

$$Set^\dagger(P \uplus P') = Set^\dagger(Set^\dagger(P) \uplus Set^\dagger(P')). \tag{3}$$

Every mapping $f : E \to E'$ extends into a mapping : $\mathbb{M}(E) \to \mathbb{M}(E')$ and we have, for all multisets P :

$$Set^\dagger(f(P)) = Set^\dagger(f(Set^\dagger(P))). \tag{4}$$

We fix C. If G is a p-graph of type included in C, then $CC(G)$ is its set of connected components and $\pi CC(G)$ is the multiset of the types $\pi(H)$ for $H \in CC(G)$. It is clear that G is connected if and only if $\left|Set^\dagger(\pi CC(G))\right| \leq 1$ (the empty graph is connected, but a connected component is defined as nonempty).

We define the *support* of a multiset $M \in \mathbb{M}(\mathcal{P}(C))$ as the set union of the sets forming M, hence as the set of elements of C having at least one occurrence in an element of M. We denote it by $Support(M)$. This set is empty if and only if M is empty or is $\{\emptyset, ..., \emptyset\}$. It is clear that $Support(\pi CC(G)) = \pi(G)$ for G as above.

We now define the set of states $Q := Q_{\mathcal{A}_{Conn,k}}$ of $\mathcal{A}_{Conn,k}$ as the set of multisets of the form $Set^\dagger(M)$ where $M \in \mathbb{M}(\mathcal{P}_+(C))$ and $\mathcal{P}_+(C)$ is the set of nonempty subsets of $C := [k]$. Their characteristic property is, for $N \in Q$:

$$P_N(t) \text{ if and only if } N = Set^\dagger(\pi CC(cval(t))).$$

Examples of states are $\{\{a,b\},\{a,b\}\}$ and $\{\{a\},\{a,b\},\{b,c,d,f\}\}$, that we will denote respectively by $\{ab,ab\}$ and $\{a,ab,bcdf\}$. (We will do the same in the sequel in our examples: we will replace sets and multisets of labels by words, where letters are ordered in the alphabetical order. The original notation will be kept in definitions and proofs.) The state $\{ab,ab\}$ corresponds to a graph $cval(t)$ that has at least two connected components, all of type $\{a,b\}$. The state $\{a,ab,bcdf\}$ corresponds to a graph that has at least three connected components including at least one of each type $\{a\}$, $\{a,b\}$ and $\{b,c,d,f\}$, and none of other types.

The number of states is thus $2^{|\mathcal{P}_+(C)|} + |\mathcal{P}_+(C)| = 2^{2^k-1} + 2^k - 1 < 2^{2^k}$. The accepting states are $\emptyset$ and the singletons. We will detail the transitions and prove the correctness of the construction.

For every term t in $T(F_k^{\mathrm{u}})$, we let $r(t) := Set^\dagger(\pi CC(cval(t)))$. We will define $\mathcal{A}_{Conn,k}$ in such a way that $r(t)$ is the state reached at the root of t.

Lemma 32. *The function $r : T(F_k) \to \mathbb{M}(\mathcal{P}(C))$ is computable inductively.*

Proof. We consider each operation of F_k^{u} in turn.

Claim 32.1 : $Set^\dagger(\pi CC(G_1 \oplus G_2)) = Set^\dagger(Set^\dagger(\pi CC(G_1)) \uplus Set^\dagger(\pi CC(G_2)))$.

Proof. Follows from Equality (3) above because $CC(G_1 \oplus G_2) = CC(G_1) \cup CC(G_2)$ and $\pi CC(G_1 \oplus G_2) = \pi CC(G_1) \uplus \pi CC(G_2)$. □

By using this claim for a term $t = t_1 \oplus t_2$ with $G_1 = cval(t)/u_1$ and $G_2 = cval(t)/u_2$ (where u_1 and u_2 are the two sons of the root of t), we get: $r(t) = r(t_1 \oplus t_2) = Set^\dagger(r(t_1) \uplus r(t_2))$.

Claim 32.2: $Set^\dagger(\pi CC(relab_h(G))) = Set^\dagger(h(Set^\dagger(\pi CC(G))))$.

Proof. We have $CC(relab_h(G)) = \{relab_h(H) \mid H \in CC(G)\}$ hence, $\pi CC(relab_h(G)) = h(\pi CC(G))$. The result follows from Equality (4) about $Set^\dagger$. □

By applying this claim to $G = cval(t_1)$, we get $r(relab_h(t_1)) = Set^\dagger(relab_h(r(t_1)))$.

For handling the operations $add_{a,b}$, we define, for $a, b \in C$, $a \neq b$ a mapping $f_{a,b} : \mathbb{M}(\mathcal{P}(C)) \to \mathbb{M}(\mathcal{P}(C))$ by[13]:

$f_{a,b}(P) := P$ if a and b are not both in $Support(P)$,

and otherwise:

$f_{a,b}(P) := P' \uplus \{Support(P - P')\}$

where P' is the multiset $\{\alpha \in P \mid \alpha \cap \{a,b\} = \emptyset\}$. Note that $f_{a,b}(P) = \{Support(P)\}$ if each set in P contains a or b (or both).

[13] We will also use this definition in the case where $a = b$ in Section 6.4 below.

Claim 32.3: $Set^{\dagger}(\pi CC(add_{a,b}(G))) = f_{a,b}(Set^{\dagger}(\pi CC(G)))$.

Proof. If a and b are not both in $Support(Set^{\dagger}(\pi CC(G))) = \pi(G)$, then $add_{a,b}(G) = G$ and $f_{a,b}(Set^{\dagger}(\pi CC(G))) = Set^{\dagger}(\pi CC(G))$, so the result holds.

Otherwise, we enumerate $CC(G)$ as $\{G_1, ..., G_q\}$ in such a way that $\pi(G_i) \cap \{a, b\} = \emptyset$ for all $i = 1, ..., p$ and $\pi(G_i) \cap \{a, b\} \neq \emptyset$ for all $i = p+1, ..., q$. Then, since $\{a, b\} \subseteq \pi(G_{p+1}) \cup ... \cup \pi(G_q)$, we have $CC(add_{a,b}(G)) = \{G_1, ..., G_p, H\}$ where H consists of $G_{p+1}, ..., G_q$ linked by the edges added by $add_{a,b}$. It follows that:

$$\begin{aligned}\pi CC(add_{a,b}(G)) &= \{\pi(G_1), ..., \pi(G_p)\} \uplus \{\pi(G_{p+1}) \cup ... \cup \pi(G_q)\} \\ &= f_{a,b}(\pi CC(G))\end{aligned}$$

Hence, $Set^{\dagger}(\pi CC(add_{a,b}(G))) = Set^{\dagger}(f_{a,b}(\pi CC(G)))$. We now want to prove that :

$$Set^{\dagger}(f_{a,b}(\pi CC(G))) = f_{a,b}(Set^{\dagger}(\pi CC(G))),$$

but this follows from the observation that we have $Set^{\dagger}(f_{a,b}(P)) = f_{a,b}(Set^{\dagger}(P))$ for every $P \in \mathbb{M}(\mathcal{P}(C))$. □

By applying this claim to $G = cval(t_1)$, we get $r(add_{a,b}(t_1)) = f_{a,b}(r(t_1))$, and this observation completes the proof of the lemma. □

Here is an example illustrating the last claim. Let G be such that $\pi CC(G) = \{b, b, c, c, ab, ac, ad, cd\}$. We have $\pi CC(add_{a,b}(G)) = \{c, c, abcd, cd\}$ and $Set^{\dagger}(\pi CC(add_{a,b}(G))) = \{c, abcd,\ cd\}$. On the other hand, $Set^{\dagger}(\pi CC(G)) = \{b, c, ab, ac, ad, cd\}$ and $f_{a,b}(Set^{\dagger}(\pi CC(G))) = \{c, abcd, cd\}$, hence $f_{a,b}(Set^{\dagger}(\pi CC(G))) = Set^{\dagger}(\pi CC(add_{a,b}(G)))$ as stated by the claim.

The transitions are shown in Table 15.

Transitions	Conditions
$\varnothing \to \emptyset$ $\mathbf{c} \to \{\{a\}\}$	 $\mathbf{c}$ is $\mathbf{a}$ or $\mathbf{a}^{\ell}$
$relab_h[P] \to N$	$N = Set^{\dagger}(h(P))$
$add_{a,b}[P] \to N$	$N = f_{a,b}(P)$
$\oplus[P_1, P_2] \to N$	$N = Set^{\dagger}(P_1 \uplus P_2)$

Table 15: The transitions of $\mathcal{A}_{Conn,k}$

Remarks. 1. This automaton is not minimal. The states $\{ab, ac\}$ and $\{ab, ac, bc\}$ are equivalent as one can check easily. Characterizing the corresponding minimal automaton is both difficult and uninteresting, because of the next fact.

2. We know from [CouEng, Example 4.54.4] that there exists no automaton with less than $2^{2^{\lfloor k-1 \rfloor/2}}$ states that checks the connectivity of graphs of clique-width at most k (with $k \geq 3$).

3. Let us define the *size* of a multiset of words P (here, words represent subsets of C) as $\| P \| := |P| + \Sigma_{\alpha \in P} |\alpha|$. Each letter has size 1. For example, the size of $\{\emptyset, abc, ac, bcd\}$ is

$4+3+2+3=12$, which is the length of a possible coding by the word $\{, abc, ac, bcd$. We obtain a notion of size for the states of $\mathcal{A}_{Conn,k}$ (another one, based on a different syntax will be given in Example 42). Each state occurring in a run of this automaton on a term that defines a graph with n vertices has a size bounded by $\min\{2n, (k+1)\cdot 2^k\}$: if P is such a state, its number of elements (as a multiset) is at most $\min\{n, 2^k\}$; the total number of occurrences of letters in P is at most n which gives the bound $n+n$; each set in P has at most k elements, which gives the other bound $2^k + k\cdot 2^k$. This shows that, even if $k=30$, these states are manageable whenever n is not too large. This observation will be used in Section 7.

6.2. Using annotated terms

We now show that if the given term t is annotated by ADD_t (cf. Definition 6(b)), then the automaton can be made to run with states of smaller sizes, hence faster if it is used as a fly-automaton (cf. Section 7) because the transitions will be easier to compute.

First a notation: if R is a set of unordered pairs of port labels, then we denote by f_R the composition (in any order) of the unary functions $f_{a,b}$ for all $\{a,b\} \in R$ (cf. Claim 32.3).We now observe that if the graph $cval(t)/u$ (where t and u are as above) has two connected components of respective types $\{a,b\}$ and $\{a,c\}$, and furthermore $\{b,c\} \in ADD_t(u)$, then these two components will be part of a unique one in $cval(t)$. Hence, anticipating that, the automaton can replace $\{a,b\}$ and $\{a,c\}$ by $\{a,b,c\}$.

Table 16 shows the transitions of a modified automaton of $\mathcal{A}'_{Conn,k}$. The annotation is used only at the occurrences of $\oplus$ (cf. in Section 5.1.2, the automaton for $edg(X_1,X_2)$ for the notation $(\oplus, R)$ that puts the annotation with the function symbol). It contains all the necessary information regarding edge additions, hence, the transitions for $add_{a,b}$ are just identity.

Transitions	Conditions
$\varnothing \to \emptyset$ $\mathbf{c} \to \{\{a\}\}$	 $\mathbf{c}$ is $\mathbf{a}$ or $\mathbf{a}^\ell$
$relab_h[P] \to N$	$N = Set^\dagger(h(P))$
$add_{a,b}[P] \to P$	
$(\oplus, R)[P_1, P_2] \to N$	$N = f_R(Set^\dagger(P_1 \uplus P_2))$

Table 16: The transitions of $\mathcal{A}'_{Conn,k}$

The states of $\mathcal{A}'_{Conn,k}$ are the same as those of $\mathcal{A}_{Conn,k}$. If we let r denote the run of $\mathcal{A}_{Conn,k}$ on a term t, and r' denote the run of $\mathcal{A}'_{Conn,k}$ on the annotated term (t, ADD_t). We have $\| r'(u) \| \leq \| r(u) \|$ for every u in $Pos(t)$ (cf. the end of the previous section for the size $\| P \|$ of a state P).

6.3. Graphs of degree at most d

We now define an automaton smaller than $\mathcal{A}_{Conn,k}$ for verifying the connectedness of graphs that we *know to be of degree* at most d: we replace Q by the set Q_d of states N in Q such that each label of C belongs to at most d sets of N. The accepting states are the singletons and the empty set, as in $\mathcal{A}_{Conn,k}$. We denote this automaton by $\mathcal{A}^{\leq d}_{Conn,k}$.

Here is the idea. If a port label a of a p-graph G belongs to d' sets in $Set^\dagger(\pi CC(G))$ and $d' > d$, this means that there are a-ports in at least d' connected components of G. If $add_{a,b}$ is applied to G and $\pi_G(x) = b$, then x has degree at least d' in $add_{a,b}(G)$. If such a p-graph G is defined as $cval(t)/u$ for a term t such that $cval(t)$ has degree at most d, then no edge addition on the path in t between the root and u can create edges between a vertex x and one in $\pi_G^{-1}(a)$ (because otherwise, this operation would create at least d' edges incident with x, and this vertex would have in $cval(t)$ a degree larger than d). So the state at u for an automaton like $\mathcal{A}_{Conn,k}$ running on t need not store the label a any longer. This label is somehow "dead". Hence, we can delete a from $Set^\dagger(\pi CC(G))$ and then, we apply $Set^\dagger$ again to remove the double occurrences of elements that may have been created.

Formally, if $N \in \mathbb{M}(\mathcal{P}(C))$, we define $Trim_d(N)$ by removing from all the sets forming N every label that occurs in more than d elements of N. Here is an example with $d = 2$. Let $N = \{a, b, ab, ac, adef, bcg, def\}$. The mapping $Trim_2$ removes a and b, so that $Trim_2(N) = \{\emptyset, \emptyset, \emptyset, c, cg, def, def\}$ and by removing duplicates, we get $Set^\dagger(Trim_2(N)) = \{\emptyset, c, cg, def\}$.

We define the set of states of $\mathcal{A}^{\leq d}_{Conn,k}$ as Q_d, the set of multisets in $\mathbb{M}(\mathcal{P}(C))$ of the form $Set^\dagger(Trim_d(N))$ for N in $\mathbb{M}(\mathcal{P}_+(C))$. Table 17 shows the transitions (to be compared with those of Table 15). (We can also add an *Error* state that replaces every state N such that $\emptyset \in N$ and $|N| \geq 2$. Some terms are thus rejected faster. We cannot detail all possible optimizations.)

Transitions	Conditions
$\varnothing \to \emptyset$ $\mathbf{c} \to \{\{a\}\}$	 $\mathbf{c}$ is $\mathbf{a}$ or $\mathbf{a}^\ell$
$relab_h[P] \to N$	$N = Set^\dagger(Trim_d(h(P)))$
$add_{a,b}[P] \to N$	$N = f_{a,b}(P)$
$\oplus[P_1, P_2] \to N$	$N = Set^\dagger(Trim_d(P_1 \uplus P_2))$

Table 17: The transitions of $\mathcal{A}^{\leq d}_{Conn,k}$

Each multiset P in Q_d has at most kd elements belonging to $\mathcal{P}(C)$. Hence, $|Q_d| \leq 2^{d \cdot k^2}$. We will prove that:

$$L(\mathcal{A}_{Conn,k}) \cap L_{Deg \leq d,k} \subseteq L(\mathcal{A}^{\leq d}_{Conn,k}) \subseteq L(\mathcal{A}_{Conn,k}) \tag{5}$$

where $Deg \leq d$ is the property that a graph has maximal degree at most d. The automaton $\mathcal{A}^{\leq d}_{Conn,k}$ may reject a term that defines a connected graph of maximal degree larger than d. It does not check whether G has maximal degree at most d.

Let us give an example with $d = 3$. Let $t := add_{a,c}(\mathbf{a} \oplus s)$ where

$$s := add_{a,b}(\mathbf{a} \oplus \mathbf{b}) \oplus add_{a,c}(\mathbf{a} \oplus \mathbf{c}) \oplus add_{a,b}(\mathbf{a} \oplus add_{b,c}(\mathbf{b} \oplus \mathbf{c})).$$

The term $\mathbf{a} \oplus s$ defines the graph :

$$a \qquad a - b \qquad a - c \qquad a - b - c$$

and t defines a connected graph of maximal degree 5 (where the two c-ports are adjacent to four a-ports). The runs of $\mathcal{A}_{Conn,k}$ and $\mathcal{A}^{\leq d}_{Conn,k}$ on the term s yield the same state : $\{ab, ac, abc\}$. On

the term $\mathbf{a} \oplus s$, the automaton $\mathcal{A}_{Conn,k}$ yields $\{a, ab, ac, abc\}$ while $\mathcal{A}^{\leq d}_{Conn,k}$ removes a and yields $\{\emptyset, b, c, bc\}$. Hence t is rejected by $\mathcal{A}^{\leq d}_{Conn,k}$ because we have a state of the form $\{\emptyset, ...\}$. Since the label a has been removed from the state, the operation $add_{a,c}$ applied to $\mathbf{a} \oplus s$ is "considered" by the automaton has having no effect although it makes $cval(t)$ connected.

For proving the correctness, we define a variant of p-graphs.

Definition 33. *p*-graphs*

(a) A p*-*graph* G is a p-graph, some vertices of which may have no port label. In other words, π_G is a partial function : $V_G \to C$. Every p-graph is a p*-graph. So is every graph, without using a default port label (cf. Definition 3, Section 2.1). Without being empty, it has an empty type. The operations $\oplus$, $add_{a,b}$ and $relab_h$ (Definition 4) extend to p*-graphs in the obvious way.

(b) Let $B \subseteq C$. We define $del_B(G)$ as the p*-graph obtained by deleting all labels belonging to B. In particular, $G^\circ = del_C(G)$.

For a multiset $N \in \mathbb{M}(\mathcal{P}(C))$, we define $Del_B(N)$ as the multiset in $\mathbb{M}(\mathcal{P}(C - B))$ obtained by removing from the sets forming N every label of B. For every p*-graph G, we have :

$$Del_B(\pi CC(G)) = \pi CC(del_B(G)). \tag{6}$$

(c) If G and H are p*-graphs, we say that G is a *sub-p*-graph* of H, written $G \subseteq H$, if $G^\circ \subseteq H^\circ$ and $\pi_G^{-1}(a) \subseteq \pi_H^{-1}(a)$ for every $a \in C$. In particular, $\pi(G) \subseteq \pi(H)$.

Lemma 34. *Let $t \in T(F^{\mathrm{u}}_k)$.*

(1) If $t \in L(\mathcal{A}^{\leq d}_{Conn,k})$, then $cval(t)$ is connected.

(2) If t defines a connected graph of maximal degree at most d, then $t \in L(\mathcal{A}^{\leq d}_{Conn,k})$.

Proof. The proof will use two claims. Let $t \in T(F^{\mathrm{u}}_k)$, r be the run of $\mathcal{A}_{Conn,k}$ and r' be the run of $\mathcal{A}^{\leq d}_{Conn,k}$ on t. We will use r' to define a p*-graph G with vertex set $V_{cval(t)}$. We will define it as $G(root_t)$ where, for every node u of t, $G(u)$ is a p*-graph with vertex set $V_{cval(t)/u}$ such that $G(u) \subseteq cval(t)/u$. We define $G(u)$ by bottom-up induction on u:

If u is a leaf, then $G(u) := cval(t)/u$. Note that $V_{G(u)} = \{u\}$.

If u is an occurrence of $add_{a,b}$ with son u_1, then $G(u) := add_{a,b}(G(u_1))$.

If u is an occurrence of $relab_h$ with son u_1, then $G(u) := del_B(relab_h(G(u_1)))$ where B is the set of labels that occur in more than d elements of the multiset $h(r'(u_1))$ (see an example below).

If u is an occurrence of $\oplus$ with sons u_1 and u_2, then $G(u) := del_B(G(u_1) \oplus G(u_2))$ where B is the set of labels that occur in more than d elements of the multiset $r'(u_1) \uplus r'(u_2)$ (see an example below).

Here are the examples. Let $d = 3$, let u be an occurrence of $relab_h$ with son u_1 such that h relabels a into b; assume that $r'(u_1) = \{ac, bd, abc, bcd\}$. Then $h(r'(u_1)) = \{bc, bc, bd, bcd\}$, and since b belongs to four sets in $h(r'(u_1))$, it must be deleted. We get $r'(u) = Set^\dagger(\{c, c, d, cd\}) = \{c, d, cd\}$. Let us now assume that u is an occurrence of $\oplus$ with sons u_1 and u_2, that $r'(u_1)$ is as

above and that $r'(u_2) = \{ac, acd\}$. Then $r'(u_1) \uplus r'(u_2) = \{ac, ac, bd, abc, acd, bcd\}$ and we must delete a and c, so that $r'(u) = Set^\dagger(\{\emptyset, \emptyset, b, d, bd, bd\}) = \{\emptyset, b, d, bd\}$.

We prove two claims showing the meaning of $G(u)$.

Claim 34.1:
For every $t \in T(F_k^{\mathrm{u}})$ and $u \in Pos(t)$, we have $r'(u) = Set^\dagger(\pi CC(G(u)))$.

Proof. By bottom-up induction on u. The fact is clear if u is a leaf.

Let u be an occurrence of $add_{a,b}$ with son u_1. Then,

$r'(u) = f_{a,b}(r'(u_1))$

$= f_{a,b}(Set^\dagger(\pi CC(G(u_1))))$ (by induction),

$= Set^\dagger(\pi CC(add_{a,b}(G(u_1))))$ (by Claim 31.3)

$= Set^\dagger(\pi CC(G(u)))$ (by the definition of $G(u)$.

Let now u be an occurrence of $\oplus$ with sons u_1 and u_2. We have :

$r'(u) = Set^\dagger(Trim_d(r'(u_1) \uplus r'(u_2)))$

$= Set^\dagger(Del_B(r'(u_1) \uplus r'(u_2)))$ where B is the set of labels a that occur in more than d elements of $r'(u_1) \uplus r'(u_2)$,

$= Set^\dagger(Del_B(Set^\dagger(\pi CC(G(u_1))) \uplus Set^\dagger(\pi CC(G(u_2)))))$ (by induction),

$= Set^\dagger(Del_B(\pi CC(G(u_1)) \uplus \pi CC(G(u_2))))$ as we can check easily[14]

$= Set^\dagger(Del_B(\pi CC(G(u_1) \oplus G(u_2))))$

$= Set^\dagger(\pi CC(del_B(G(u_1) \oplus G(u_2))))$ (by Equality (6)),

$= Set^\dagger(\pi CC(G(u)))$ (by the definition of $G(u)$).

If u is an occurrence of $relab_h$, the proof is similar, by using the fact that, for every multiset M, set B and mapping h we have:

$$Set^\dagger(Del_B(h(Set^\dagger(M)))) = Set^\dagger(Del_B(h(M))).$$

□

Claim 34.2: For every $t \in T(F_k^{\mathrm{u}})$ and $u \in Pos(t)$:

(i) $G(u) \subseteq cval(t)/u$ and these two p*-graphs have the same vertex set.

(ii) For every $a \in C$, if $0 \neq \left|\pi^{-1}_{cval(t)/u}(a)\right| \leq d$, then $\pi^{-1}_{cval(t)/u}(a) = \pi^{-1}_{G(u)}(a)$.

Furthermore, if $cval(t)$ has maximum degree at most d, then:

(iii) $G(u)^\circ = (cval(t)/u)^\circ$.

[14] Using the fact that, for every two multisets M and N and every set B, we have $Set^\dagger(Del_B(Set^\dagger(M) \uplus Set^\dagger(N)))$ $= Set^\dagger(Del_B(M \uplus N))$.

Proof. For fixed t, we use bottom-up induction on u.

(i) This is clear from the definition of $G(u)$ and $cval(t)/u$. (Because of the removal of certain port labels, some edges created in $cval(t)$ by the operations $add_{a,b}$ are no longer created in $G(u)$.)

(ii) Let $0 \neq \left|\pi^{-1}_{cval(t)/u}(a)\right| \leq d$. We distinguish several cases.

(a) If u is a leaf, then $G(u) = cval(t)/u$, hence $\pi^{-1}_{cval(t)/u}(a) = \pi^{-1}_{G(u)}(a) = \{u\}$.

(b) If u is an occurrence of $add_{b,c}$ with son u_1, then $\left|\pi^{-1}_{cval(t)/u_1}(a)\right| = \left|\pi^{-1}_{cval(t)/u}(a)\right| \leq d$, hence $\pi^{-1}_{cval(t)/u_1}(a) = \pi^{-1}_{G(u_1)}(a)$. We get the desired equality because $G(u) = add_{b,c}(G(u_1))$.

(c) If u is an occurrence of $\oplus$ with sons u_1 and u_2, then $\pi^{-1}_{cval(t)/u}(a)$ is the union of the disjoint sets $\pi^{-1}_{cval(t)/u_1}(a)$ and $\pi^{-1}_{cval(t)/u_2}(a)$, hence $d_1 + d_2 \leq d$ where $d_i := \left|\pi^{-1}_{cval(t)/u_i}(a)\right|$ for $i = 1, 2$. By induction, every a-port of $cval(t)/u_i$ is also one of $G(u_i)$. The number of elements of $Set^{\dagger}(\pi CC(G(u_i)))$ that contain a is at most $\left|\pi^{-1}_{G(u_i)}(a)\right| = d_i$. It follows that at most d elements of $r'(u_1) \uplus r'(u_2)$ which is equal to $Set^{\dagger}(\pi CC(G(u_1))) \uplus Set^{\dagger}(\pi CC(G(u_2)))$, (this equality follows from Claim 33.1) contain a. Hence a is still in $\pi(G(u))$. We have thus $\pi^{-1}_{cval(t)/u}(a) \subseteq \pi^{-1}_{G(u)}(a)$ and the equality by (i).

(d) If u is an occurrence of $relab_h$ with son u_1, we let $\{b_1, ..., b_p\}$ enumerate the set $h^{-1}(a) \cap \pi(cval(t)/u_1)$. Hence :

$$\pi^{-1}_{cval(t)/u}(a) = \pi^{-1}_{cval(t)/u_1}(b_1) \cup ... \cup \pi^{-1}_{cval(t)/u_1}(b_p).$$

The sets of this union are disjoint, hence $d_1 + ... + d_p \leq d$ where $d_i := \left|\pi^{-1}_{cval(t)/u_1}(b_i)\right|$. By induction $\pi^{-1}_{cval(t)/u_1}(b_i) = \pi^{-1}_{G(u_1)}(b_i)$ for each i. The number q_i of elements of $Set^{\dagger}(\pi CC(G(u_1)))$ that contain b_i is at most d_i. The number of elements of $h(Set^{\dagger}(\pi CC(G(u_1)))) = h(r'(u_1))$ (by Claim 33.1) that contain a is at most $q_1 + ... + q_p$ hence, at most d, and a is in $\pi(G(u))$. We have $\pi^{-1}_{cval(t)/u}(a) \subseteq \pi^{-1}_{G(u)}(a)$, hence the equality by (i).

(iii) Assume now that $cval(t)$ has maximum degree at most d. Let $x-y$ be an edge of $cval(t)/u$. It is created by an operation $add_{a,b}$ at some occurrence w above the leaves x and y and below or equal to u. Since t defines a graph of degree at most d, we have $0 \neq \left|\pi^{-1}_{cval(t)/w}(a)\right| \leq d$ and $0 \neq \left|\pi^{-1}_{cval(t)/w}(b)\right| \leq d$. Hence by (ii), letting w_1 be the son of w, we have $\pi_{G(w)}(x) = \pi_{G(w_1)}(x) = a$ and $\pi_{G(w)}(y) = \pi_{G(w_1)}(y) = b$, so this edge is also created in $G(w)$ hence is an edge of $G(u)$. We have $(cval(t)/u)^{\circ} \subseteq G(u)^{\circ}$ and the desired equality by (i). □

We now prove the lemma.

(1) If t is accepted by $\mathcal{A}^{\leq d}_{Conn,k}$, then $r(root_t)$ is empty or singleton, hence, by Claim 33.1, $G = G(root_t)$ is connected. Since, by (i) of Claim 33.2, G° is a spanning subgraph of $cval(t)^\circ$, the p-graph $cval(t)$ is connected.

(2) Conversely, let $t \in T(F^{\mathrm{u}}_k)$ define a connected graph of maximal degree d. Then, by (iii), $G^\circ = cval(t)^\circ$ is connected. By Claim 33.1, $|r'(root_t)| = \left|Set^\dagger(\pi CC(G))\right|$ hence,

$\left|Set^\dagger(\pi CC(G))\right| \leq 1$ and t is accepted by $\mathcal{A}^{\leq d}_{Conn,k}$.

□

This lemma establishes the inclusions (5) and the correctness of the construction. Annotated terms can also be used as in 6.1. With the same definitions and notation as at the end of Section 6.1, we can evaluate the maximal size of a state as $\min\{2n, (k+1)\cdot k \cdot d\}$.

6.4. "Small" nondeterministic automata

In Section 6.1, we have constructed deterministic F^{u}_k-automata for connectedness and non-connectedness with less than 2^{2^k} states, and shown the existence of a nondeterministic automaton for nonconnectedness with $O(2^{2k})$ states. Here, we construct a nondeterministic automaton for connectedness with $2^{O(k\log(k))}$ states. This is interesting for testing minor inclusion (cf. Example 21, Section 4.1). The sentence of Example 21 yields (by means of Corollary 28) a nondeterministic automaton with $2^{O(k\log(k))}$ states, whereas, the construction derived from the sentence of Section 6.1 yields a much larger automaton.

We will use the annotation ADD_t of Definition 6(b)[15] (in Section 2.3) together with another one, denoted by π'. In a few words, ADD_t describes edges (and we used it already in Section 6.2 to accelerate computations) whereas π' describes similarly paths of length 2. The major difference is that $ADD_t(u)$ depends only on the operation symbols above u in t, whereas $\pi'(u)$ depends in a more complicated way on the symbols of the context of u in t. This explains why we need nondeterminism.

Definition 35. *The annotation π'*

Let $t \in T(F^{\mathrm{u}}_k)$. For $u \in Pos(t)$, we define $\pi(u) := \pi(cval(t)/u)$ and $\pi'(u)$ as the set of port labels a in $\pi(u)$ such that $(a,u) \rightarrow_t^+ (b,w_1) \rightarrow_t (b,w)$ where w is an occurrence of $add_{b,c}$ or $add_{c,b}$ such that $cval(t)/w_1$ has a c-port x that is not in $cval(t)/u$ (cf. Definition 6(b) for the relation $\rightarrow_t$). Hence, $a \in \pi'(u)$ if and only if the operations of the context of u in t create edges between all a-ports of $cval(t)/u$ and at least one vertex not in $cval(t)/u$. In other words, any two vertices of $cval(t)/u$ with same port label belonging to $\pi'(u)$ are linked in $cval(t)$ by a path of length 2. Our use of π' will be based on the fact that, if $cval(t)/u$ has connected components whose types $\alpha_1, ..., \alpha_p$ contain all some $a \in \pi'(u)$, then these components are included in a single connected component of $cval(t)$. Hence, we can anticipate and merge $\alpha_1, ..., \alpha_p$ into a single set at node u during a bottom-up computation on t. We have used ADD_t in Section 6.2 in a similar way.

If $R \subseteq \mathcal{P}_2(C)$ and $A \subseteq C$, we define $R \circ A$ as the set $\{a \in C \mid \{a,b\} \in R$ for some $b \in A\}$. The mapping π' satisfies the following conditions which offer the possibility of a top-down computation of π' using $\pi : Pos(t) \rightarrow \mathcal{P}(C)$ that can be computed during a previous

[15] We recall that for $t \in T(F^{\mathrm{u}}_k)$, $ADD_t(u)$ is a subset of $\mathcal{P}_2(C)$, the set of 2-element subsets of $C := [k]$.

bottom-up traversal (cf. Section 5.2.1) and $ADD_t : Pos(t) \to \mathcal{P}_2(C)$ that can be computed top-down (Definition 6(b)) simultaneously with π':

$$\pi'(root_t) = \emptyset. \tag{7a}$$

If u is an occurrence of $\oplus$ with sons u_1 and u_2, then

$$\pi'(u_1) = \pi(u_1) \cap (\pi'(u) \cup ADD_t(u) \circ \pi(u_2)) \tag{7b}$$

and

$$\pi'(u_2) = \pi(u_2) \cap (\pi'(u) \cup ADD_t(u) \circ \pi(u_1)). \tag{7c}$$

If u is an occurrence of $relab_h$ with son u_1, then

$$\pi'(u_1) = \pi(u_1) \cap h^{-1}(\pi'(u)). \tag{7d}$$

If u is an occurrence of $add_{a,b}$ with son u_1, then

$$\pi'(u_1) = \pi'(u). \tag{7e}$$

Furthermore, if u is an occurrence of $\oplus$ with sons u_1 and u_2, we also have:

$$\pi'(u_1) = \pi(u_1) \cap (\pi'(u) \cup ADD_t(u) \circ \pi'(u_2)), \tag{8a}$$

$$\pi'(u_2) = \pi(u_2) \cap (\pi'(u) \cup ADD_t(u) \circ \pi'(u_1)), \tag{8b}$$

$$\pi'(u) \subseteq \pi'(u_1) \cup \pi'(u_2). \tag{8c}$$

These facts are easy to check from the definitions and Equalities (7b) and (7c). We illustrate them with the term t of Figure 2. Some of its positions are designated by $w, w_1, \cdots, w_7$. The bottom-up computation of π yields :
$\pi(w_5) = \{a, c, d\}$, $\pi(w_6) = \{b, e\}$, $\pi(w_7) = \{d\}$,
$\pi(w_4) = \{a, b, c, d, e\} = \pi(w_3) = \pi(w_2) = \pi(w_1) = \pi(w)$.

The top-down computation of ADD_t yields :
$ADD_t(w) = \emptyset$, $ADD_t(w_1) = \{\{c, d\}\} = ADD_t(w_2)$,
$ADD_t(w_3) = \{\{a, b\}, \{c, d\}\}$,
$ADD_t(w_4) = \{\{a, b\}, \{c, d\}, \{c, e\}\} = ADD_t(w_5) = ADD_t(w_6)$.

The top-down computation of π' yields :
$\pi'(w) = \emptyset = \pi'(w_1)$.
Equalities (7b) and (7c) give respectively (since $\pi'(w_1) = \emptyset$) :
$\pi'(w_2) = \{a, b, c, d, e\} \cap (\{\{c, d\}\} \circ \{d\}) = \{c\}$, and
$\pi'(w_7) = \{d\} \cap (\{\{c, d\}\} \circ \{a, b, c, d, e\}) = \{d\}$.
Equality (8a) holds because $\pi'(w_7) = \pi(w_7)$ and Equality (8b) because $\pi'(w_2) = \{c\}$ and we have :
$\pi'(w_7) = \{d\} \cap (\{\{c, d\}\} \circ \{c\}) = \{d\}$.
Inequality (8c) holds since $\pi'(w) = \emptyset$.
We have :
$\pi'(w_3) = \pi'(w_2) = \pi'(w_4)$.
Again by (7b) and (7c) we have:

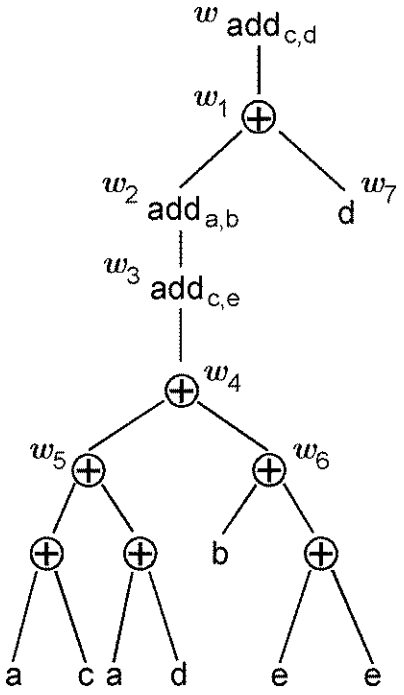

Figure 2: A term t.

$\pi'(w_5) = \{a, c, d\} \cap (\{c\} \cup \{\{a, b\}, \{c, d\}, \{c, e\}\} \circ \{b, e\}) = \{a, c\}$, and
$\pi'(w_6) = \{b, e\} \cap (\{c\} \cup \{\{a, b\}, \{c, d\}, \{c, e\}\} \circ \{a, c, d\}) = \{b, e\}$.
It is easy to verify that Properties (8a), (8b) and (8c) also hold.

Informal presentation

We can construct a deterministic (bottom-up) automaton for checking the connectedness[16] of a graph given by a term annotated by ADD_t and π', but we cannot apply Lemma 15 to get from this automaton another one to check $Conn[X_1]$ (the connectedness of the induced subgraph with vertex set X_1) because this lemma is based on a transformation of terms that replaces some nullary symbols specifying vertices by the symbol $\varnothing$ whose value is the empty graph. If a term t is transformed in this way into t', the annotation $ADD_{t'}$ is the same as ADD_t, but the annotation π' for t' is not the same as the corresponding one for t because it depends on the nullary symbols. Hence, we will not use π' as a "fixed annotation" like ADD_t. We will compute it "in the states of the automaton" as we can do for π (cf. Section 5.2.1) (that also depends on the nullary symbols). A difficulty comes from the fact that π' is computable deterministically in a *top-down* way (by using ADD_t and π) whereas we want a *bottom-up* automaton. To handle this, we will construct a *non-deterministic bottom-up* automaton that will *guess* for each u a possible value of $\pi'(u)$ and check simultaneously the consistency of the

[16]There are simpler linear-time algorithms for doing that. Our objective is not to check the global connectedness of a graph, but rather, the connectedness of its induced subgraphs, cf. Example 21.

guessed value with the previously guessed values. The correctness of all the guesses made during a run will be ascertained if the state reached at the root is accepting.

Non-deterministic computation of π'

We first define a complete *non-deterministic automaton* $\mathcal{B}_k$ intended to compute π' on terms $t \in T(F_k^{\mathrm{u}})$ annotated by ADD_t. Its states are the pairs (A, B) such that $B \subseteq A \subseteq C$ ($:= [k]$, as usual). That a state (A, B) is reached at a node u means that $A = \pi(u)$ and that B is a potential value of $\pi'(u)$. The transitions are in Table 18. As in Section 5.1.2, the annotation ADD_t is only used for the transitions relative to $\oplus$ (cf. Table 5 for the notation $(\oplus, R)$). The accepting states are $(A, \emptyset)$ for all $A \subseteq C$. The transitions implement the characterization of π' by Properties (7a)-(7e) so that $\mathcal{B}_k$ has a unique accepting run on each term t. The transitions for $(\oplus, R)$ also use Property (8c) in order to narrow *a priori* the set of possible sets B', hence to limit the degree of nondeterminism.

This automaton is non-ambiguous: on each accepted term t, it has a unique accepting run. The unique possible accepting root state is $(\pi(cval(t)), \emptyset)$; the transitions yield the top-down computation of π' and at each position u, the state is $(\pi(u), \pi'(u))$.

Transitions	Conditions
$\varnothing \to (\emptyset, \emptyset)$	
$\mathbf{c} \to \{\{a\}, B\}$	$\mathbf{c} \in \{\mathbf{a}, \mathbf{a}^{\ell}\}$ and, either $B = \emptyset$ or $B = \{a\}$
$relab_h[(A, B)] \to (A', B')$	$A' = h(A), B' \subseteq A', B = A \cap h^{-1}(B')$
$add_{a,b}[(A, B)] \to (A, B)$	
$(\oplus, R)[(A_1, B_1), (A_2, B_2)] \to (A', B')$	$A' = A_1 \cup A_2, B' \subseteq B_1 \cup B_2$ $B_1 = A_1 \cap (B' \cup (R \circ A_2))$ $B_2 = A_2 \cap (B' \cup (R \circ A_1))$

Table 18: The transitions of $\mathcal{B}_k$

The non-deterministic automaton $\mathcal{C}_k$ for connectedness

We will construct $\mathcal{C}_k$ by "enriching" $\mathcal{B}_k$. Its states are the triples (A, B, Π) such that $B \subseteq A \subseteq C$, Π is a partition in nonempty sets of B if B is not empty and is $\{\emptyset\}$ or $\emptyset$ if $B = \emptyset$. (We could remove B from these triples because it can be determined from Π but the description is more clear in this way). We denote by Q this set of states (for fixed k). Its cardinality is bounded by $2^k.B(k)$ where $B(k)$ is the number of partitions in nonempty sets of $[k]$. We have $B(k) \leq k!$ if $k \geq 8$, hence, $|Q| = 2^{O(k \log(k))}$. ($B(k)$ is a Bell number, see Wikipedia or any textbook in Combinatorics). In order to describe the meaning of a state (A, B, Π) at node u in t and to define the transitions of $\mathcal{C}_k$, we need more technical definitions.

Definition 36. *More operations on multisets and on p*-graphs.*

(a) For each $a \in C$, we define $f_a : \mathbb{M}(\mathcal{P}(C)) \to \mathbb{M}(\mathcal{P}(C))$ as the mapping[17] such that :

$f_a(P) := P' \uplus \{Support(P - P')\}$ where P' is the multiset $\{\alpha \in P \mid a \notin \alpha\}$.

[17] In Claim 32.3, Section 6.1, we defined a similar mapping $f_{a,b}$.

If $a \notin Support(P)$, then $P' = P$ and $f_a(P) := P$. If a belongs to each set of P, then $P' = \emptyset$ and $f_a(P) = \{Support(P)\}$.

If $B \subseteq C$, we define f_B as the composition in any order of the mappings f_a for $a \in B$ (the resulting mapping does not depend on the order of composition).

(b) If G is a p*-graph and $B \subseteq C$, we let $add_B(G)$ be G augmented with edges between every two distinct vertices x and y such that $\pi(x) = \pi(y) \in B$ (an edge is added between x and y only if there does not already exist one). We will use the following obvious fact :

$$\pi CC(add_B(G)) = f_B(\pi CC(G)), \text{ for } B \subseteq C^{18} \tag{9}$$

(c) For a term $t \in T(F_k^{\mathrm{u}})$ and $u \in Pos(t)$, we define

$H_u := del_{C-\pi'(u)}(add_{\pi'(u)}(add_{ADD_t(u)}(cval(t)/u)))$, and
$\Pi_u := \pi CC(H_u)$.

The p*-graph H_u is obtained from $cval(t)/u$ in three steps :
1) by adding the edges that are created by the context of u in t, hence, that are in $cval(t)$ but not in $cval(t)/u$;
2) by adding an edge between any vertices x and $y \neq x$ (unless there exists one already) such that $\pi_{cval(t)/u}(x) = \pi_{cval(t)/u}(y) \in \pi'(u)$: in $cval(t)$ such vertices x and y are linked by a path of length 2, hence are in the same connected component; the vertex sets of the connected components of $cval(t)$ and H_{root_t} are the same.
3) Finally, the port labels not in $\pi'(u)$ are removed.

The multiset Π_u is obtained from $\pi CC(cval(t)/u)$ by merging any two sets containing, one a label a and the other a label b such that $\{a, b\} \in ADD_t(u)$, then by merging any two sets containing a same label from $\pi'(u)$ (by (9)) and finally by removing the port labels not in $\pi'(u)$. It is a partition of $\pi'(u)$ if this set is not empty. We also have, $\Pi_{root_t} := Del_C(\pi CC(cval(t)))$, hence this multiset is empty if $cval(t)$ is the empty graph, and it consists of n times $\emptyset$ if $cval(t)$ has n connected components.

Lemma 37. *For every term $t \in T(F_k^{\mathrm{u}})$ the following are equivalent:*

(i) $cval(t)$ is connected,

(ii) Π_{root_t} is empty or $\{\emptyset\}$,

(iii) for every $u \in Pos(t)$, if $\emptyset \in \Pi_u$, then $\Pi_u = \{\emptyset\}$.

Proof. The equivalence of (i) and (ii) and the implication (iii) $\Longrightarrow$ (ii) are clear from the previous remarks (the empty graph is connected). We prove (i) $\Longrightarrow$ (iii) by contradiction. If some multiset Π_u contains $\emptyset$ and another set, then the p-graph $cval(t)/u$ has at least two connected components, one of which, say H, is such that $\pi(H) \cap \pi'(u) = \emptyset$. This implies that there is no

[18] This equality also holds for $B \subseteq \mathcal{P}_2(C)$: add_B is defined in the proof of Proposition 9 and we define f_B as the composition of the mappings $f_{a,b}$ for $\{a, b\}$ in B.

edge in $cval(t)$ that links a vertex of H and one not in $V_{cval(t)/u}$. Hence, H is also a connected component of $cval(t)$. Hence $cval(t)$ is not connected.

□

Lemma 38. *On every term $t \in T(F_k^{\mathrm{u}})$ annotated by $(ADD_t(u), \pi(u), \pi'(u))$ at each node u, the mapping $u \mapsto \Pi_u$ is computable bottom-up.*

Proof. If u is an occurrence of $\varnothing$, then $\Pi_u = \emptyset$.

If u is an occurrence of $\mathbf{a}$ or $\mathbf{a}^\ell$, then Π_u is $\{\emptyset\}$ if $\pi'(u) = \emptyset$ and it is $\{\{a\}\}$ if $\pi'(u) = \{a\}$.

If u is an occurrence of $add_{a,b}$ with son u_1, then $add_{ADD_t(u)}(cval(t)/u) = add_{ADD_t(u_1)}(cval(t)/u_1)$, $\pi'(u) = \pi'(u_1)$ (by Property (7e)), and so $\Pi_u = \Pi_{u_1}$.

If u is an occurrence of $relab_h$ with son u_1, then we have

Claim 38.1: $\Pi_u = f_{\pi'(u)}(h(\Pi_{u_1}))$.

Proof. By the definitions, H_u is obtained from $h(H_{u_1})$ by adding an edge $x - y$ whenever x is an a-port and y is a b-port of H_{u_1} such that $a \neq b$ and $h(a) = h(b)$ (so that $h(a) \in \pi'(u)$). Consider two connected components A and B of H_{u_1}. Their types α and β belong to $\Pi_{u_1} = \pi CC(H_{u_1})$, are included in $\pi'(u_1)$ and are disjoint. Clearly, A and B are connected components of $relab_h(H_{u_1})$ of respective types $h(\alpha)$ and $h(\beta)$ included in $\pi'(u)$. If $h(\alpha) \cap h(\beta) \neq \emptyset$, then A and B are linked in H_u by an edge created by $add_{\pi'(u)}$ applied to $add_{ADD_t(u)}(cval(t)/u) = relab_h(add_{ADD_t(u_1)}(cval(t)/u_1))$ and $h(\alpha)$ and $h(\beta)$ get merged in $\Pi_u = \pi CC(H_u)$. The multiset Π_u is obtained from $h(\Pi_{u_1})$ by such merges. It follows that $\Pi_u = f_{\pi'(u)}(h(\Pi_{u_1}))$. □

The last case to consider is when u is an occurrence of $\oplus$ with sons u_1 and u_2.

Claim 38.2: $\Pi_u = Del_{C-\pi'(u)}(f_{\pi'(u)}(f_{ADD_t(u)}(\Pi_{u_1} \uplus \Pi_{u_2})))$.

Proof. By the definitions, we have $\Pi_u = \pi CC(H_u)$ where

$H_u = del_{C-\pi'(u)}(add_{\pi'(u)}(add_{ADD_t(u)}(cval(t)/u_1 \oplus cval(t)/u_2)))$. We define

$K := del_{C-\pi'(u)}(add_{\pi'(u)}(add_{ADD_t(u)}(H_{u_1} \oplus H_{u_2})))$ and we compare it to H_u.

It is clear that H_u and K have the same vertices. Their vertices have the same port labels: every a-port of K is an a-port of H_u. Conversely, let x be an a-port of H_u. Then $a \in \pi'(u) \cap \pi(u_i) \subseteq \pi'(u_i)$ ($i = 1, 2$, by 7b, 7c), hence x is an a-port of H_{u_i}, hence of K.

We now compare their edges, and prove first that every edge of H_u is in K. Let $x - y$ be an edge of H_u. If it is an edge of $cval(t)/u_i$, it is one of H_{u_i}, hence of K. If it is added to $cval(t)/u_i$ by $add_{ADD_t(u)}$, then it is in H_{u_i}, hence in K. If it is added to $cval(t)/u_i$ by $add_{\pi'(u)}$, then x and y have a same port label $a \in \pi'(u_i)$ (by the previous argument), hence, it is in H_{u_i}, whence in K.

Assume now that x is in $cval(t)/u_1$ with label a and y is in $cval(t)/u_2$ with label b. If $x - y$ is created by $add_{ADD_t(u)}$, then $a \in \pi'(u_1)$ and $b \in \pi'(u_2)$ by the definition

of π'. Hence, x and y have the same respective labels in H_{u_1} and in H_{u_2} and the edge is also created in K by the same operation. If $x-y$ is created by $add_{\pi'(u)}$, then $a=b$, and we have $a \in \pi'(u_1) \cap \pi'(u_2)$, hence x and y are a-ports of H_{u_1} and H_{u_2}, and the edge is also created in K by $add_{\pi'(u)}$.

Conversely, consider an edge $x-y$ of K. It is an edge of H_u, except possibly if it is an edge of H_{u_i} not in $cval(t)/u_i$. Assume this with $i=1$. If this edge is added to $cval(t)/u_1$ by $add_{ADD_t(u_1)}$, then it is also added to $cval(t)/u_1$ (in H_u) by $add_{ADD_t(u)}$, because $ADD_t(u_1) = ADD_t(u)$ (by Definition 6 in Section 2.3). If it is added to $cval(t)/u_1$ by $add_{\pi'(u_1)}$, then, either it is added to $cval(t)/u_1$ in H_u by $add_{\pi'(u)}$, or not. The latter case happens if x and y are a-ports of $cval(t)/u_1$ with a in $\pi'(u_1) - \pi'(u)$. In this case, there is in $cval(t)/u_2$ a b-port such that $\{a,b\} \in ADD_t(u)$ (by (8a)). Hence, the operation $add_{ADD_t(u)}$ creates in H_u two edges $x-z$ and $y-z$. These two edges form a path in H_u linking x and y.

This analysis shows that the p*-graphs H_u and K do not have exactly the same edges (because of the very last subcase) but that the vertex sets of their connected components are the same. Since the port labels are the same, we have $\pi CC(H_u) = \pi CC(K)$. Hence, we have :

$\Pi_u = \pi CC(H_u) = \pi CC(K)$

$= \pi CC(del_{C-\pi'(u)}(add_{\pi'(u)}(add_{ADD_t(u)}(H_{u_1} \oplus H_{u_2}))))$,

$= Del_{C-\pi'(u)}(f_{\pi'(u)}(\pi CC(add_{ADD_t(u)}(H_{u_1} \oplus H_{u_2})))))$ (by (6), Definition 33 and (9), Definition 36(b))

$= Del_{C-\pi'(u)}(f_{\pi'(u)}(f_{ADD_t(u)}(\pi CC(H_{u_1}) \uplus \pi CC(H_{u_2})))))$ (by (9))

$= Del_{C-\pi'(u)}(f_{\pi'(u)}(f_{ADD_t(u)}(\Pi_{u_1} \uplus \Pi_{u_2})))$ as was to be proved.

□

This concludes the proof of the claim and that of the lemma. □

Finally, the transitions of $\mathcal{C}_k$

We let $\mathcal{C}_k$ be the non-deterministic automaton with set of states Q, transitions defined by Table 19 and accepting states of the form $(A, \emptyset, \emptyset)$ or $(A, \emptyset, \{\emptyset\})$. The conditions about (A,B) in Table 19 are the same as in Table 18.

Proposition 39. *For every term t in $T(F_k)$ annotated by ADD_t:*
(1) if r is an accepting run of $\mathcal{C}_k$, then for every node u, if $(A,B,\Pi) = r(u)$, we have:

i) $A = \pi(u)$,

ii) $B = \pi'(u)$,

iii) $\Pi = \Pi_u$ *and, if* $\emptyset \in \Pi_u$ *then* $\Pi_u = \{\emptyset\}$,

iv) $cval(t)$ *is connected.*

(2) Conversely, if $cval(t)$ is connected, then $\mathcal{C}_k$ has a unique accepting run on t.

Transitions	Conditions
$\varnothing \to (\emptyset, \emptyset, \emptyset)$ $\mathbf{c} \to (\{a\}, B, \Pi)$	 $\mathbf{c} \in \{\mathbf{a}, \mathbf{a}^{\ell}\}, B = \emptyset$ and $\Pi = \{\emptyset\}$, or $B = \{a\}$ and $\Pi = \{\{a\}\}$.
$relab_h[(A, B, \Pi)] \to (A', B', \Pi')$	$A' = h(A), B' \subseteq A'$, $B = A \cap h^{-1}(B')$, $\Pi' = f_{B'}(h(\Pi))$, if $\emptyset \in \Pi'$ then $\Pi' = \{\emptyset\}$.
$add_{a,b}[(A, B, \Pi)] \to (A, B, \Pi)$	
$(\oplus, R)[(A_1, B_1, \Pi_1), (A_2, B_2, \Pi_2)] \to (A, B, \Pi)$	$A = A_1 \cup A_2, B \subseteq B_1 \cup B_2$ $B_1 = A_1 \cap (B \cup (R \circ A_2))$, $B_2 = A_2 \cap (B \cup (R \circ A_1))$, $\Pi = g_B(f_R(\Pi_1 \uplus \Pi_2))$, if $\emptyset \in \Pi$ then $\Pi = \{\emptyset\}$.

Table 19: The transitions of $\mathcal{C}_k$

Proof. (1) Let r be an accepting run of $\mathcal{C}_k$ on t. The first two components of each state define the unique accepting run of $\mathcal{B}_k$. This fact implies the equalities i) and ii) for each u.

We now prove iii) by bottom-up induction on u. This is actually a consequence of the facts proved in Lemma 38. Since r is defined as accepting, the conditions that $\Pi = \{\emptyset\}$ if $\emptyset \in \Pi$ (cf. Table 19) is satisfied for each Π in an accessible state (A, B, Π). Hence, Condition (ii) of Lemma 37 holds and $cval(t)$ is connected by this lemma.

(2) By Lemma 37, if $cval(t)$ is connected and u is a position in t, then $\Pi_u = \{\emptyset\}$ if $\emptyset \in \Pi_u$. It follows that the mapping $r : Pos(t) \to Q$ such that $r(u) := (\pi(u), \pi'(u), \Pi_u)$ is an accepting run of $\mathcal{C}_k$ on t. □

This proposition establishes the correctness of the construction of $\mathcal{C}_k$.

The maximal size of a state is $2 \min\{n, k\}$ for an appropriate encoding similar to that of Section 6.1. This is not much but remember that this automaton is non-deterministic.

7. Fly automata

Table 20 collects results of Sections 5-6 and shows an upper-bound to the number of states $N(k, P)$ of the constructed automaton for Property P. These values come from constructions of complete and deterministic automata that are not necessarily minimal. The mark (*) indicates that the automata must take irredundant terms as input. The use of Θ indicates that we know a lower bound for the minimal automaton. $N_{ann}(k, P)$ and $N_{ndet}(k, P)$ are the numbers of states of a deterministic and, respectively, a nondeterministic automaton on annotated terms. The large number of states in many cases motivates the introduction of fly-automata.

A *fly-automaton* is an automaton whose transitions are defined by computable functions. Each time a transition is needed, it is computed. To take an example, the automaton $\mathcal{A}_{Conn,4}$ of Section 6.1 has more than 2^{15} states. Its transitions described in Table 15 in a concise way

Property P	$N(k,P)$	$N_{ann}(k,P)$	$N_{ndet}(k,P)$
$edg(X_1,X_2)$	k^2+k+3	$2k+3$	
St	2^k+1		
$Link(X_1,X_2)$	$2^{2k}+1$		
$Dom(X_1,X_2)$	$2^{2k}+1$		
$InDeg_{\leq d}(X_1,X_2)$	$< 2^{2k\log(d+2)}$ (*)		
$Path(X_1,X_2)$	$< 2^{k^2+2k}$		
$DirCycle$	$< 2^{k^2+k}$		$O(2^{2k}$
$Clique$	$< 2^{k^2+k}, 2^{\Theta(k^2)}$	2^k+1	
$ConnIfDeg_d$	$< 2^{d\cdot k^2}$		
$Conn$	$2^{2^{\Theta(k)}}$		$2^{O(k\log(k))}$
$\neg Conn$	$2^{2^{\Theta(k)}}$		$O(2^{2k})$
$Cycle$	2^{9^k} (*)		

Table 20: Some basic graph properties

can be expressed by programs but cannot be stored in a table. However, for checking a term of size 100, only 100 transitions need to be fired. They can be computed on the fly.

Since we need not list its states and transitions, a fly-automaton can be infinite. For example, the automata $\mathcal{A}_{Conn,k}$ for all values of k can be merged into a single infinite automaton. This infinite fly-automaton can run on any term in $T(F^{\mathrm{u}}_{\infty})$ where F^{u}_{∞} is the union of all signatures F^{u}_{k}. Hence,we need not use a particular automaton $\mathcal{A}_{Conn,k}$ for each k.

7.1. Definitions and general properties

Definition 40. *Fly-automaton*

An F-automaton $\mathcal{A} = \langle F, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, Acc_{\mathcal{A}}\rangle$ is a *fly-automaton* (a *fly-F-automaton* if F must be specified) if it satisfies the following conditions:

(a) The signature F is finite or countably infinite. In the latter case, F must be *effectively given*, that is, defined with a bijection to a recursive (i.e., decidable) set of integers or of words over a finite alphabet such that its arity mapping is computable via this bijection. (Effectively given sets and computable functions over them are defined in detail in Chapter 2 of [9].) Each state has thus a size defined as the length of the corresponding word (cf. the end of Section 6.1 and Example 42 below). An integer is also handled as a word.

(b) The set of states $Q_{\mathcal{A}}$ is finite or countably infinite and effectively given. In the latter case (and without loss of generality), we assume that it is a recursive set of words over a finite alphabet Z. The set $Acc_{\mathcal{A}}$ must be recursive.

(c) The transition relation $\delta_{\mathcal{A}}$ is defined by a computable function $\gamma_{\mathcal{A}}$ that maps any tuple $(f, q_1, \ldots, q_m)$ where $f \in F$ has arity m and $q_1, \ldots, q_m \in Q_{\mathcal{A}}$ to a finite sequence of states that enumerates in increasing order the set $\{q \mid f[q_1, \ldots, q_m] \rightarrow_{\mathcal{A}} q\}$ for some fixed (say lexicographic) linear order on Z^*. This condition implies that, if $\mathcal{A}$ is not deterministic, then each transition yields only finitely many states.

Tables 4,5,11,14-18 describe computable functions $\gamma_{\mathcal{A}}$ in concise ways: we call *meta-transitions* such descriptions.

(d) All definitions given for automata in Definition 17 are applicable to fly-automata.

(e) A finite automaton $\mathcal{A}$ whose sets of states $Q_{\mathcal{A}}$ and $Acc_{\mathcal{A}}$ are enumerated and whose transitions are listed in a table is called a *table-automaton*. Fly- and table-automata will be compared in Section 8.

Example 41. We let $F = \{a, f\}$ with $\rho(a) = 0$ and $\rho(f) = 2$. We define $\mathcal{A} = \langle F, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, Acc_{\mathcal{A}} \rangle$ with $Z = \{0, 1\}$, $Acc_{\mathcal{A}} = 1\{0,1\}^*$, $Q_{\mathcal{A}} = \{0\} \cup Acc_{\mathcal{A}}$. Each state is the binary notation of a nonnegative integer. With this convention, the transitions are specified by the meta-rules $a \to_{\mathcal{A}} 1$, $f[i, i] \to_{\mathcal{A}} i + 1$ if $i \neq 0$ and $f[i, j] \to_{\mathcal{A}} 0$ if $i = 0$ or $j \neq i$. Hence $\mathcal{A}$ is a complete and deterministic fly-automaton. It recognizes the set of terms in $T(F)$ whose syntactic tree has all branches of the same length. This set is not regular.

Example 42. We consider the finite F_k^{u}-automata $\mathcal{A}_{Conn,k}$ for connectedness constructed in Section 6.1. It is clear that $\mathcal{A}_{Conn,k}$ is a subautomaton of $\mathcal{A}_{Conn,k'}$ for every $k' > k$. Hence we can define $\mathcal{A}_{Conn}$ as the union of the automata $\mathcal{A}_{Conn,k}$ (cf. Definition 17(i)). Its signature F_∞^{u} is the union of the signatures F_k^{u}; it is effectively given. For making the union of the automata $\mathcal{A}_{Conn,k}$ into a fly-F_∞^{u}-automaton $\mathcal{A}_{Conn}$, we must specify their states by words, in a computable way.

The method used in Section 6.1 only works for fixed k. We let Z_0 the alphabet consisting of[19] 0,1, (,) and ,. The states of $\mathcal{A}_{Conn,k}$ are either $\{\alpha, \alpha\}$ or $\{\alpha, \beta, ..., \delta\}$ for pairwise distinct nonempty subsets $\alpha, \beta, ..., \delta$ of $[k]$. We encode an integer $i \in [k]$ by its binary writing $Bin(i) \in 1\{0,1\}^*$, a set $\alpha \subseteq [k]$ by the word $b(\alpha) = (Bin(i_1),...,Bin(i_p))$ where $i_1, ..., i_p$ are the elements of α in increasing order. Then, we encode $\{\alpha, \alpha\}$ by $((bin(\alpha)))$ (we double parentheses) and $\{\alpha, \beta, ..., \delta\}$ by $(bin(\alpha),bin(\beta),...,bin(\delta))$ where the words $bin(\alpha)$, $bin(\beta)$, ... , $bin(\delta)$ are ordered by increasing order. This syntax specifies the states of the automata $\mathcal{A}_{Conn,k}$ in a unique way. A state $\{\alpha, \beta, ..., \delta\}$ with m elements is thus represented by a word of length at most $4m \cdot k \cdot \log(k)$. It is then straightforward to see that the transitions of Table 15 are computable. The set of accepting states is recursive.

Example 43. We now define a fly-F_∞^{u}-automaton $\mathcal{C}$ that counts the number of vertices of the graph defined by an input term. Its set of states is $\mathbb{N}$ (integers are encoded in binary as words in $\{0\} \cup 1\{0,1\}^*$), and its transitions are specified by the following meta-transitions :

$\mathbf{a} \to_{\mathcal{C}} 1$, $\mathbf{a}^{\ell} \to_{\mathcal{C}} 1$,$\varnothing \to_{\mathcal{C}} 0$,

$add_{a,b}[i] \to_{\mathcal{C}} i$, $ren_h[i] \to_{\mathcal{C}} i$ and

$\oplus[i, j] \to_{\mathcal{C}} i + j$.

For each state i, $L(\mathcal{C}, \{i\})$ is the set of terms that define graphs having i vertices. If we let $Acc_{\mathcal{C}}$ be a recursive set of integers, then $L(\mathcal{C})$ is the set of terms that define graphs (of any clique-width) whose number of vertices is in $Acc_{\mathcal{C}}$. The corresponding set of graphs is not monadic second-order definable if $Acc_{\mathcal{C}}$ is, for example, the set of prime numbers.

[19] Note the use of the boldface symbols (,) and , to distinguish them from the corresponding symbols of the meta-language.

Proposition 44. *Let $\mathcal{A}$ be a fly-F-automaton. The membership in $L(\mathcal{A})$ of any term $t \in T(F)$ is decidable. The emptiness of $L(\mathcal{A})$ is not decidable in general.*

Proof. Let $\mathcal{A}$ be given. For every term $t \in T(F)$ and every position u in t, the set $run^*_{\mathcal{A},t}(u)$ is finite. One can compute these sets for all u by bottom-up induction. Then $t \in L(\mathcal{A})$ if and only if the set $run^*_{\mathcal{A},t}(root_t)$ contains an accepting state.

For proving the undecidability, we associate an automaton $\mathcal{A}_h$ with every primitive recursive mapping $h : \mathbb{N} \to \mathbb{N}$. We let $F = \{a, f, g\}$ with $\rho(a) = 0$ and $\rho(f) = \rho(g) = 1$. We define $\mathcal{A}_h = \langle F, Q_\mathcal{A}, \delta_{\mathcal{A}_h}, Acc_{\mathcal{A}_h}\rangle$ with $Q_\mathcal{A}$ as in Example 41, $Acc_{\mathcal{A}_h} = \{0\}$ and the meta-transitions $a \to_{\mathcal{A}_h} 1$, $f[i] \to_{\mathcal{A}_h} i+1$ if $i > 0$ and $g[i] \to_{\mathcal{A}_h} 0$ if $h(i) = 0$. Hence $\mathcal{A}_h$ is a deterministic fly-automaton. It recognizes a term in $T(F)$ if and only if $h(i) = 0$ for some $i > 0$. This fact is undecidable, hence, the emptiness of $L(\mathcal{A}_h)$ is undecidable. □

We say that a relabelling $h : F \to F'$ between two effectively given signatures F and F' is *computable* if the integer or the word representing $h(f)$ can be computed from the one representing f. We say that its *inverse is computable* if each set $h^{-1}(f')$ is finite and one can compute from the integer or the word representing any $f' \in F'$ the set of those representing the elements of $h^{-1}(f')$.

Proposition 45. *Let $\mathcal{A}$ and $\mathcal{B}$ be two fly-F-automata and F' be another effectively given signature.*

(1) There are fly-automata $\mathcal{A} \cup \mathcal{B}$ and $\mathcal{A} \cap \mathcal{B}$ that define respectively $L(\mathcal{A}) \cup L(\mathcal{B})$ and $L(\mathcal{A}) \cap L(\mathcal{B})$.

(2) There exists a complete and deterministic fly-F-automaton equivalent to $\mathcal{A}$. The language $T(F) - L(\mathcal{A})$ is thus recognized by a fly-automaton.

(3) If $h : F \to F'$ is a relabelling whose inverse is computable, then the image automaton $h(\mathcal{A})$ is a fly-automaton.

(4) If $h : F' \to F$ is a computable relabelling, then the inverse-image $h^{-1}(\mathcal{A})$ of $\mathcal{A}$ is a fly-automaton.

Proof. (1) Without loss of generality, we can assume that $Q_\mathcal{A} \cup Q_\mathcal{B} \subseteq Z^*$ where Z contains the alphabet Z_0 of Example 42. We take for $Q_{\mathcal{A}\cup\mathcal{B}}$ the language $(0,Q_\mathcal{A}) \cup (1,Q_\mathcal{B}) \subseteq Z^*$. The standard construction of the union of two automata with disjoint sets of states yields the result. We take for $Q_{\mathcal{A}\cap\mathcal{B}}$ the language $(Q_\mathcal{A},Q_\mathcal{B}) \subseteq Z^*$. The standard construction of the product of two automata yields the result.

(2) Given $\mathcal{A} = \langle F, Q_\mathcal{A}, \delta_\mathcal{A}, Acc_\mathcal{A}\rangle$, we define as follows a determinized fly-automaton $\mathcal{D} = det(\mathcal{A})$.

We let $Q_\mathcal{D}$ be the set of finite strictly increasing sequences of states of $Q_\mathcal{A}$ (increasing with respect to some fixed linear order on Z^*, cf. Definition 40) and including the empty sequence. We define $Acc_\mathcal{D}$ as the set of those that contain a state in $Acc_\mathcal{A}$. If $f \in F$ has arity m and $\sigma_1, \ldots, \sigma_m \in Q_\mathcal{D}$, we let $\gamma_\mathcal{D}(f, \sigma_1, \ldots, \sigma_m)$ be the finite sequence[20] that enumerates in increasing order the set $\{q \in Q_\mathcal{A} \mid f[q_1, \ldots, q_m] \to_\mathcal{A} q \text{ for some } q_1 \in \sigma_1, \ldots, q_m \in \sigma_m\}$.

[20] See Definition 17(f) in Section 3 for the notation $\gamma_\mathcal{D}$.

It is clear that $\mathcal{D}$ is a complete and deterministic fly-automaton, and that for every term $t \in T(F)$, the sequence $run^*_{\mathcal{D},t}(root_t)$ enumerates in increasing order the finite set $run^*_{\mathcal{A},t}(root_t)$, cf. Definition 17(g) (Section 3). It follows that $L(\mathcal{D}) = L(\mathcal{A})$.

The language $T(F) - L(\mathcal{A})$ is recognized by the fly-automaton $\langle F, Q_\mathcal{D}, \gamma_\mathcal{D}, Q_\mathcal{D} - Acc_\mathcal{D}\rangle$.

(3) and (4): these assertions are straightforward to prove from the definitions and the constructions of Definition 17(h). □

7.2. Bounding space and time

We now examine the space and time used to check whether a term is recognized by a fly-automaton. In view of our applications, we will only consider binary signatures, i.e., with symbols of arity at most 2.

Definition 46. *The Strahler number of a term.*

Let F be a binary signature.

(a) The *Strahler number* of $t \in T(F)$ is the positive integer such that :

$Strh(t) = 1$ if $t = a \in F$ (hence a has arity 0),

$Strh(t) = Strh(t_1)$ if $t = f(t_1)$,

$Strh(t) = Strh(t_1) + 1$ if $t = f(t_1, t_2)$ and $Strh(t_1) = Strh(t_2)$,

$Strh(t) = \max\{Strh(t_1), Strh(t_2)\}$ if $t = f(t_1, t_2)$ and $Strh(t_1) \neq Strh(t_2)$.

The Strahler number of a slim k-expression (cf. Definition 5(c)) is 1 or 2. Strahler numbers are studied in [15].

(b) An *m-register program over* F is a sequence P of assignments of the form

$R_i := a$ for $a \in F$ of arity 0, or

$R_i := f(R_j)$ for $f \in F$ of arity 1, or

$R_i := f(R_j, R_\ell)$ for $f \in F$ of arity 2, or

$R_i := R_j$,

where $1 \leq i, j, \ell \leq m$ and $R_1, \cdots, R_m$ are the *registers* that can hold values of any relevant type. By evaluating in sequence these assignments in $T(F)$, we obtain, either a term $val(P)$ defined as the one computed in the last assignment or nothing if some register in a right-hand side of an assignment has undefined value. For an example, if P is the sequence $R_1 := a; R_2 := R_1; R_2 := g(R_1, R_2); R_3 := f(R_1, R_2); R_1 := g(R_1, R_3)$, then $val(P) = g(g(a, a), f(a, g(a, a)))$. If P is $R_1 := a; R_2 := g(R_1, R_1); R_1 := f(R_3, R_2); R_1 := g(R_2, R_2)$, then $val(P)$ is undefined because R_3 has no value in $R_1 := f(R_3, R_2)$.

If $t = val(P)$ is defined, then P can be used for computing the value of t under any interpretation of the function symbols of F by total functions. The following fact is easy to prove by induction on the structure of t.

Fact [30]: Every term $t \in T(F)$ is $val(P)$ for some $Strh(t)$-register program P over F that consists of $|t|$ assignments.

The integer $Strh(t)$ is an easy to compute upper bound to the minimal number of registers of a program P that is necessary to get $t = val(P)$, but it does not give the minimal value (just take $t = f(a, a)$ for a counter-example).

Definition 47. *Parameters for measuring computations.*

Let $\mathcal{A}$ be a complete and deterministic fly-automaton over a finite binary signature F. Each state q has a size by Definition 40.

(a) Let $\tau_{\mathcal{A}}$ be a mapping: $\mathbb{N} \to \mathbb{N}$ such that $\tau_{\mathcal{A}}(0)$ bounds the time for computing $\gamma_{\mathcal{A}}(a)$ for any $a \in F$ of arity 0 and $\tau_{\mathcal{A}}(m)$ bounds the time for computing $\gamma_{\mathcal{A}}(f, q)$ and $\gamma_{\mathcal{A}}(g, q, q')$ for any $f \in F$ of arity 1, any $g \in F$ of arity 2 and any q, q' of size at most m. We assume also that $\tau_{\mathcal{A}}(m)$ bounds the time for checking if a state q of size at most m is accepting.

(b) For $t \in T(F)$, we denote by $b_{\mathcal{A}}(t)$ the maximal size of a state occurring in the unique run of $\mathcal{A}$ on t.

(c) For a fly-automaton that is not complete and deterministic, we use these parameters relative to the associated complete and deterministic fly-automaton constructed in Proposition 45(2).

Table 23 collects some maximal sizes resulting from our constructions of deterministic automata (n is the number of vertices of the checked graph). States are encoded by words over appropriate alphabets, typically Z_0 of Example 42. We have $\log(k)$ factors because each label of $[k]$ is encoded by a binary word. The $O(.)$ notation does not hide large constants.

Property	Maximum size of a state
$Sgl, Card_p, edg$	constant
$Partition$, $Disjoint$	constant
$St, Link, Dom$	$O(\min\{n, k\}.\log(k))$
$Path, DirCycle, Clique$	$O(\min\{n^2, k^2\}.\log(k))$
$ConnIfDeg_d$	$O(\min\{n, d.k^2\}.\log(k))$
$Conn, Cycle$	$O(\min\{n, k.2^{O(k)}\}.\log(k))$

Table 21: Sizes of states

Proposition 48. *Let $\mathcal{A}$ be a complete and deterministic fly-automaton over a finite binary signature F. The time and space required to check if a term $t \in T(F)$ belongs to $L(\mathcal{A})$ are respectively bounded by $(|t| + 1).\tau_{\mathcal{A}}(b_{\mathcal{A}}(t))$ and by $Strh(t).b_{\mathcal{A}}(t)$.*

Proof. The time bound is clear from the definitions (the term +1 is for checking that the state found at the root is accepting). For the space bound, we consider a term t and a $Strh(t)$-register program over F that computes this term (cf. Definition 46(b)). We run this program by letting its registers take values in $Q_{\mathcal{A}}$. A nullary symbol a evaluates to the state $\gamma_{\mathcal{A}}(a)$, a function symbol f of arity 1 denotes the mapping $q \longmapsto \gamma_{\mathcal{A}}(f, q)$ and a function symbol g of arity 2 denotes the mapping $q, q' \longmapsto \gamma_{\mathcal{A}}(g, q, q')$.

This program computes $run_{\mathcal{A},t}(root_t)$ by maintaining simultaneously at most $Strh(t)$ states in its registers. We neglect the fact that, for evaluating $R_i := f(R_j)$ or $R_i := g(R_j, R_\ell)$, some auxiliary space may be necessary. We have never seen cases where the space bound $Strh(t).b_{\mathcal{A}}(t)$ should be increased in a significant way for this reason. □

Remarks. (1) If we check properties of graphs defined by slim k-expressions t, then the space bound $Strh(t) \cdot b_{\mathcal{A}}(t)$ can be replaced by $2 \cdot b_{\mathcal{A}}(t)$.

(2) In applications to graphs, it may happen that for two terms t and t' defining the same graph, the values $b_{\mathcal{A}}(t)$ and $b_{\mathcal{A}}(t')$ are very different. This shows that the bounds of Proposition 48 depend strongly on the given term t.

Here is an example where $\mathcal{A}$ is $\mathcal{A}_{Conn}$, the fly-automaton for checking connectedness defined as the union (cf. Definition 17(i) in Section 3) of the automata $\mathcal{A}_{Conn,k}$ (we use the simple encoding of states defined at the end of Section 6.1). Let $k > 0$. For every nonempty subset of $[k]$, we let P_A be a path with $|A|$ vertices and of type $\pi(P_A) = A$. If $B \subseteq \mathcal{P}_+([k])$, we let G_B be the graph $\boldsymbol{c} \oplus \bigoplus_{A \in B} P_A$ where $\boldsymbol{c}$ is an isolated vertex labelled by $k+1$, and we denote this graph by a slim term t_B. Finally we let H_B be defined by the term $t'_B = add_{k+1,[k]}(t_B) \in T(F^{\mathrm{u}}_{k+1})$, where $add_{k+1,[k]}$ is the composition of the operations $add_{k+1,i}$ for all $i \in [k]$. It is clear that G_B has $|B| + 1$ connected components of pairwise different types, whereas H_B is connected. The maximal size of a state of $\mathcal{A}$ occurring in a run on t'_B is thus larger than $|B|$ and may be more than 2^k. However, H_B can also be defined by a slim term $s_B \in T(F^{\mathrm{u}}_{k+2})$ any subterm of which defines a graph with at most 2 connected components: the vertices are added one by one and each time a vertex is added, it is immediately linked to a $(k+1)$-port. The maximal size of a state of $\mathcal{A}$ running on s_B is thus $O(k)$.

However, if we use the annotation of t'_B defined in Section 6.2, the states are as in the run on s_B hence of size $O(k \cdot \log(k))$. This example shows the usefulness of annotations and also that the choice of a term t yielding a small value $b_{\mathcal{A}}(t)$ depends strongly on the property checked by $\mathcal{A}$. We do not see how to make a general statement (based of the syntax of a defining formula) about this choice. □

7.3. Fly-automata constructed from MS formulas

We let F_∞ (resp. $F^{(n)}_\infty$) be the union of the signatures F_k (resp. $F^{(n)}_k$) for all k (cf. Definitions 4 and 11). For every MS formula $\varphi(X_1, ..., X_n)$, we let $L_{\varphi(X_1,...,X_n)}$ be the set of terms $t * (V_1, ..., V_n)$ in $T(F^{(n)}_\infty)$ such that $val(t) \models \varphi(V_1, ..., V_n)$, and similarly for a property $P(X_1, ..., X_n)$.

Proposition 49. *For every MS formula $\varphi(X_1, ..., X_n)$, one can define a fly-$F^{(n)}_\infty$-automaton $\mathcal{A}_{\varphi(X_1,...,X_n)}$ that recognizes $L_{\varphi(X_1,...,X_n)}$.*

Proof. For every formula $\varphi(X_1, ..., X_n)$, either atomic or that defines one of the basic properties of Section 4.1 and for every $k < k'$, we have $\mathcal{A}_{\varphi(X_1,...,X_n),k} \subseteq \mathcal{A}_{\varphi(X_1,...,X_n),k'}$. This is clear from the definitions and constructions of Sections 5 and 6. Hence, we take for $\mathcal{A}_{\varphi(X_1,...,X_n)}$ the union of the automata $\mathcal{A}_{\varphi(X_1,...,X_n),k}$ for all k. It is clear that it is a fly-$F^{(n)}_\infty$-automaton.

By Proposition 45, the constructions of Section 4.2 extend to fly-automata. In particular, the relabellings used for existential quantifications (Lemma 26) are computable and have computable inverses. Those used for variable substitutions and relativization (Lemmas 13 and 15) are computable. So all our previous constructions of finite automata extend to fly-automata. □

7.3.1. One automaton for all clique-widths

The observation that $\mathcal{A}_{\varphi(X_1,...,X_n),k} \subseteq \mathcal{A}_{\varphi(X_1,...,X_n),k'}$ if $k < k'$ also holds for all properties of Section 5 except for maximal indegree at most d : the states of the automata constructed in

Section 5.2.6 are pairs (α, β) where $\alpha : [k] \to [0, d]$ and $\beta : [k] \to [0, d+1]$. However, if k is replaced by $k' > k$, then a function $\alpha : [k] \to [0, d]$ is replaced by the function $\alpha' : [k'] \to [0, d]$ such that $\alpha'(i) := \texttt{if } i \leq k \texttt{ then } \alpha(i) \texttt{ else } 0$, as one checks easily and similarly for β. In the description of a state, the function α can be defined by the set of pairs $(i, \alpha(i))$ such that $i > 0$ and $\alpha(i) \neq 0$ with the convention that $\alpha(i) = 0$ if there is no pair (i, p) in the set. With this variant of the definition given in Section 5.2.6 and the similar one for β, we have inclusions of the sets of states and so, $\mathcal{A}_{InDeg_d(X_1,X_2),k} \subseteq \mathcal{A}_{InDeg_d(X_1,X_2),k'}$.

7.3.2. Existential quantifications

We examine the bounds of Proposition 48 for the automata of a sentence φ of the form $\exists X_1, ..., X_p.\theta(X_1, ..., X_p)$. We let $N(k, \theta)$ be the number of states of a deterministic fly-$F_k^{(p)}$-automaton $\mathcal{A} = \mathcal{A}_{\theta(X_1,...,X_p),k}$. We let $\mathcal{A}'$ be the corresponding nondeterministic fly-F_k-automaton constructed by Lemma 26 and $\mathcal{B} = det(\mathcal{A}')$ (cf. Proposition 45(2)) be the determinized fly-F_k-automaton recognizing $L_{\varphi,k}$.

Our objective is to bound the time and space needed for running $\mathcal{B}$ on $t \in T(F_k)$. To do that, we will use $ndeg_{\mathcal{A}'}(t)$, the degree of nondeterminism of $\mathcal{A}'$ on t (cf. Definition 17(g)). It is clear that $b_{\mathcal{B}}(t) \leq ndeg_{\mathcal{A}'}(t) \cdot \overline{b_{\mathcal{A}}}(t)$ where $\overline{b_{\mathcal{A}}}(t) := \max\{b_{\mathcal{A}}(t') \mid t' \in T(F_k^{(p)}), pr^{(p)}(t') = t\}$ and so the memory space is bounded by $Strh(t) \cdot ndeg_{\mathcal{A}'}(t) \cdot \overline{b_{\mathcal{A}}}(t)$.

We now bound the time necessary to fire a transition of $\mathcal{B}$ at a position u of t.

We consider first an occurrence u of a nullary symbol (not $\varnothing$). The automaton $\mathcal{A}'$ has 2^p transitions at u yielding a multiset of 2^p states. This multiset must be made into a sorted set. This step takes time $O(p \cdot 2^p)$: we consider that the lexicographical comparison of two states obtained from nullary symbols takes time $\tau_{\mathcal{A}}(0)$, a (small) constant depending on k. We get the time bound $O(\tau_{\mathcal{A}}(0) \cdot p \cdot 2^p)$ that does not depend on t.

Let now u be an occurrence of a binary symbol. The transitions of $\mathcal{A}'$ on binary symbols are those of $\mathcal{A}$, hence are deterministic. There are thus at most $ndeg_{\mathcal{A}'}(t)^2$ transitions of $\mathcal{A}'$ at u. The corresponding multiset can be computed in time at most $ndeg_{\mathcal{A}'}(t)^2 \cdot \tau_{\mathcal{A}}(\overline{b_{\mathcal{A}}}(t))$ and transformed into a sorted set in time $O(ndeg_{\mathcal{A}'}(t)^2 \cdot \log(ndeg_{\mathcal{A}'}(t)) \cdot \overline{b_{\mathcal{A}}}(t))$ (the lexicographical comparison of two states of $\mathcal{A}$ takes time at most $\overline{b_{\mathcal{A}}}(t)$). The same bound can be used if u is an occurrence of a unary symbol.

A graph with n vertices of clique-width at most k can be defined by a term of size at most $n(k^2 - k + 4)$ as proved in Section 2.5.3 of [9]. Hence, running $\mathcal{B}$ on such term t takes time bounded by :

$$O(n \cdot [p \cdot 2^p \cdot \tau_{\mathcal{A}}(0) + k^2 \cdot ndeg_{\mathcal{A}'}(t)^2.\Big(\tau_{\mathcal{A}}(\overline{b_{\mathcal{A}}}(t)) + \log(ndeg_{\mathcal{A}'}(t)) \cdot \overline{b_{\mathcal{A}}}(t)\Big)]).$$

The fly-automaton $\mathcal{A}$ is over a finite signature F. It can be the restriction of an F_∞-automaton to F_k. In this case, the values $\tau_{\mathcal{A}}(0), ndeg_{\mathcal{A}'}(t)$ etc. depend on k.

7.3.3. Improvements

A slight improvement is possible for sentences φ of the form $\exists X_1, ..., X_p.(Partition(X_1, ..., X_p) \wedge \theta(X_1, ..., X_p))$. The sentences expressing vertex coloring problems and minor inclusion (cf. Examples 20 and 21) are of this form. Instead of replacing a nullary symbol $\mathbf{c}$ by $(\mathbf{c}, w)$

with $w \in \{0,1\}^p$, (cf. Lemma 26), we can replace it by $(\mathbf{c}, i)$ with $i \in [p]$, to mean that the corresponding vertex belongs to X_i. The condition $Partition(X_1, ..., X_p)$ is ensured by this choice (the automaton need not check it). In the above evaluation, we can replace $p.2^p$ by $p.\log(p)$. This technique is applicable to $\exists X_1, ..., X_p(Disjoint(X_1, ..., X_p) \wedge \theta(X_1, ..., X_p))$ because it is equivalent to $\exists X_1, ..., X_{p+1}.(Partition(X_1, ..., X_{p+1}) \wedge \theta(X_1, ..., X_p))$.

We now apply it to the p-coloring problem (cf. Example 20) expressed by the sentence

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

The states of the automaton $\mathcal{C}$ for $St(X_i)$ are $Error$ and the subsets of $[k]$. Those of the automaton $\mathcal{A}$ for $St(X_1) \wedge ... \wedge St(X_p)$ are $Error$ and the p-tuples of subsets of $[k]$. It follows that $b_{\mathcal{A}}(t) \leq pk$ and that $\tau_{\mathcal{A}}(pk) = O(pk)$.

If t of size $O(n.k^2)$ defines a graph with n vertices, then we get the time bound

$$O(n \cdot [p \cdot \log(p) + k^2 \cdot (2^{2kp} \cdot p \cdot k + (k \cdot p)^2)]) = O(n \cdot p \cdot k^3 \cdot 2^{2kp}).$$

8. Experiments

Many constructions of automata described in the previous sections have been implemented in a system written in Common Lisp. We describe some aspects of this implementation and we report some experiments.

8.1. Scratch and composed fly-automata

We call *scratch* fly-automata those that are built directly from meta-transitions in order to distinguish them from the ones that are constructed by using Proposition 45 as combinations of previously defined or computed fly-automata. We call the later *composed* fly-automata.

In order to implement a scratch fly-automaton, we must specify the structure of the states and transform the meta-transitions into procedures that compute the specific transitions. We consider for example the counting automaton $\mathcal{C}$ of Example 41(3). Its Lisp implementation is shown in Figure 3.

The software `Autowrite`[21] implements table- and fly-automata on terms. As in Definitions 17 and 40, the symbols have fixed arities, but the most recent version admits also *unranked symbols* denoting associative and commutative binary operations. On top of it, we have developed a software called `Autograph` (also written in Common Lisp) to compute the fly-automata that verify graph properties (in particular monadic second-order ones, but not only, cf. Example 43). This software is intended for the signature F_∞, hence for graphs of bounded clique-width. Its extension to graphs of bounded tree-width is not difficult and will be done in the next future.

The disjoint union operation can be handled in `Autowrite` either as an ordinary binary operation or as an unranked associative and commutative one. The annotation ADD_t of Definition 6 (computable on t in a top-down way) has been implemented and has proved useful in some cases.

[21] `Autowrite` is written in Common Lisp (see [13]) and still under development `http://dept-info.labri.u-bordeaux.fr/~idurand/autowrite`

```
(defclass counting-state (state)
  ((num :initarg :num :reader num)))

;; add and rel operations do npt change the count
(defmethod graph-add-target (a b (s state)) s)
(defmethod graph-ren-target (a b (s state)) s)

;; the count of the disjoint union is the sum of the counts
(defmethod graph-oplus-target
    ((co1 counting-state) (co2 counting-state))
  (make-counting-state (+ (num co1) (num co2))))

;; a constant yields a state with count 1
(defmethod cardinality-transitions-fun
    ((root constant-symbol) (arg (eql nil)))
  (make-counting-state 1))
```

Figure 3: Lisp implementation for an automaton counting vertices

8.2. Fly- versus table-automata

When a fly-automaton $\mathcal{A}$ is finite, it can be *compiled* into a table-automaton, *provided the resulting transition table is not too large.* The accessible states and the corresponding transitions can be computed from the transition function $\gamma_{\mathcal{A}}$ (cf. Definition 40) in the following way: starting from the transitions relative to the nullary symbols, we can compute them by using a standard saturation algorithm. This algorithm determines actually the *accessible subautomaton* $\mathcal{B}$ of $\mathcal{A}$. It can be used even if $\mathcal{A}$ is infinite but has a finite signature, and it terminates if and only if $\mathcal{B}$ is finite. Provided $\mathcal{B}$ fits in the main memory, it is faster for recognizing a term than the original fly-automaton $\mathcal{A}$. If this is not the case, the access time is no longer constant and the fly-automaton $\mathcal{A}$ is *a priori* preferable.

An infinite fly-automaton can be stored in a finite memory space. A finite fly-automaton uses in general a much smaller space to encode the transition function than the corresponding table-automaton but it is slower for term recognition, especially if the transition function is complex (but we never met cases where the computation of transitions is difficult). We have discussed in Section 7 the space needed to recognize a term.

We now examine for which properties scratch table-automata can be built.

Property	**Compilation**
St	up to $cwd = 13$
$edg(X_1, X_2)$	up to $cwd = 90$
$Card_{\leq p}(X_1)$ or $Card_{=p}(X_1)$	$p + 2$ states, any cwd
$Col(X_1, \ldots, X_p)$	up to $p = 3$ for $cwd = 4$
$Conn$	up to $cwd = 3$
$Path(X_1, X_2)$	up to $cwd = 4$
$Cycle$	fails

Table 22: Direct constructions

Table 22 shows some positive cases. See Example 20(1) for the definition of $Col(X_1, \ldots, X_p)$ and p-Col.

With these properties and by using relabellings and Boolean operations (cf. Lemmas 25 and 26), we can obtain automata for properties like p-colorability, p-AC-colorability, p-VertexCover among others. Some results appear in Figure 23.

Property	**Compilation**
p-Col	up to $p = 3$ for $cwd = 2$, $p = 2$ for $cwd = 3$.
p-AC-Col	failed
p-$Chord - Free - Cycle$	up to $p = 4$ for $cwd = 4$
p-$MaxDegree$	up to $p = 1$ ($cwd = 3$), $p = 5$ ($cwd = 2$)
p-$VertexCover$	up to $p = 800$ for $cwd = 3$, $p = 100$ for $cwd = 6$

Table 23: Derived constructions

We recall that a set of vertices X of a graph G is a *vertex cover* if every edge has an end in X, i.e., if $V_G - X$ is stable. We let p-*VertexCover* mean that the considered graph has a vertex cover of cardinality p. This property is thus expressed by $\exists X_1.(Card_{=p}(X_1) \wedge St(\overline{X_1}))$; ($\overline{X_1}$ is a set term denoting the set of vertices not in X_1). The corresponding Lisp code is shown in Figure 4. The property p-*Chord-Free-Cycle* for $p \geq 4$ means that every cycle with at least p vertices has a chord.

```
;; Vertex-Cover(X1) = Stable(V-X1)
(defun fly-vertex-cover (cwd)
  (x1-to-cx1 ; Stable(V-X1)
   (fly-subgraph-stable-automaton
     cwd 1 1))) ; Stable(X1)

;; E. X1 | vertex-cover(X1) & card(X1) = k
(defun fly-k-vertex-cover (k cwd)
  (vprojection
   (intersection-automaton
    ;; Vertex-Cover(X1)
    (fly-vertex-cover cwd)
    ;; Card(X1) = k
    (fly-subgraph-cardinality-automaton
      k cwd 1 1))))
```

Figure 4: Lisp code for Vertex-Cover

8.3. Running time comparisons

We now report comparisons of the running times of a fly-automaton and that of the corresponding table-automaton. The implementation has been done by using SBCL (Steel Bank Common Lisp) on a MacBook Pro laptop equipped with a processor 2.53 GHz Intel Core Duo and a 4 GB memory.

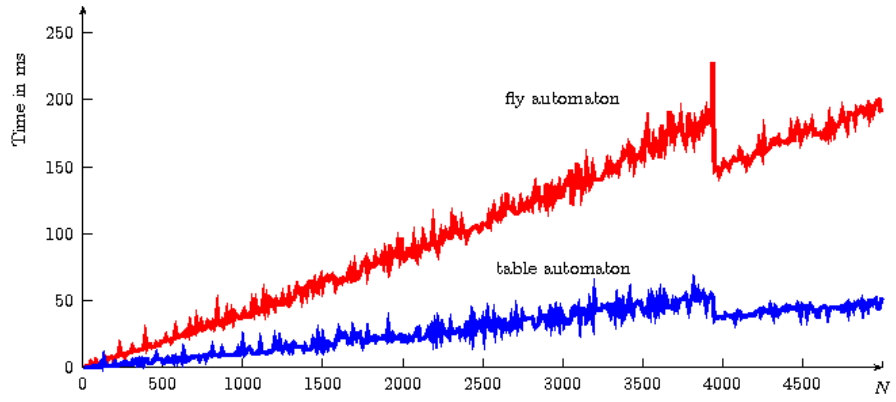

Figure 5: Fly- versus table-automata for connectedness

The running times are usually averaged over 10 runs[22] except when they are exceptionally large (more than hundreds of minutes).

8.3.1. Connectedness

We have chosen the property *Conn* for which the deterministic automata are described in Section 6.1. For a bound k on clique-width, the table-automaton has $2^{2^k-1} + 2^k - 2$ states. It can be compiled up to $k = 3$. (For $k = 4$, it has 32782 states; the number of transitions is quadratic in the number of states, see Definition 17).

Each graph P_N (an undirected path with N vertices) has clique-width 3 if $N \geq 4$. We will compare the computation times of the fly-automaton and the table-automaton for increasing (and large) values of N. The size of a term representing P_N is $5N + 1$ and its depth is $4N - 3$.

Figure 5 shows that the computation time is roughly linear with respect to n and that the slope of the line is steeper for the fly-automaton. The up and down variations around the line could possibly be explained by the launching of the automatic garbage collector. However, the global shape of the curve is linearly ascending. The sudden decrease which appears around N=3900 in both curves is probably due to memory cache organization.

8.3.2. Coloring problems

Table 24 shows some results concerning two coloring problems that are NP-complete for fixed numbers of colors (at least 3). We made some tests for three classical graphs defined by Grünbaum, Petersen and McGee. They are on figures 6,7 and 8 respectively. The 24 vertices of McGee's graph are on the external cycle.

Using a term in $T(F_3)$ of size 15 that defines this graph, its non 4-AC-colorability has been verified in less than 0.3 seconds and its 5-AC-colorability in 1.3 seconds; the last time is reduced to 0.9 seconds when using the annotation ADD_t.

[22]The times may vary because of the garbage collector.

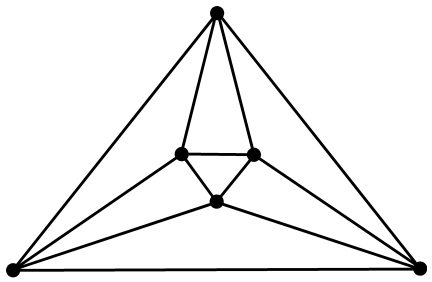

Figure 6: A graph by Grünbaum

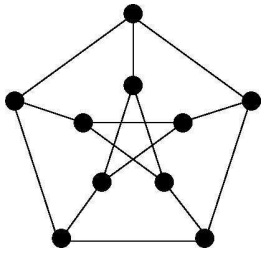

Figure 7: Petersen's Graph

For defining Grünbaum's graph, we have used a term in $T(F_3)$ of size 15 and a term in $T(F_5)$ of size 21. For Petersen's graph the automata are all impossible to construct and fly-automata have been used. For McGee's graph (24 vertices, 36 edges), we have found a term in $T(F_{10})$ of size 99 and depth 76. Using the annotation ADD_t, the verification took around 11 hours which is not that bad.

We have also checked the 3-colorability of grids of moderate clique-width. Grids are trivially 2-colorable, but our point was to use them for tests. A square grid $G_{N\times N}$ has clique-width $N+1$ ([21]). It was difficult to verify its 3-colorability for $N = 8$ and impossible for $N > 8$. See Figure 10.

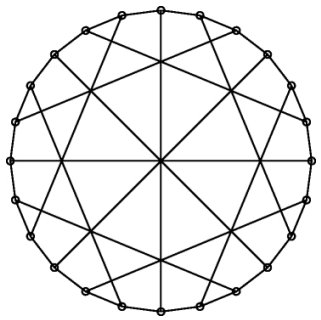

Figure 8: McGee's graph

Graph	term in	Verified property	Time without/with annotations
Grünbaum	$T(F_3)$	3-col., non 4-AC-col., 5-AC-col.	0.01/0.01, 0.27/0.18, 1.3/0.9
Grünbaum	$T(F_5)$	non 4-AC-col., 5-AC-col.	0.44/0.44, 2.6/0.9, 13.1/5.6
Petersen	$T(F_7)$	3-col., not 3-AC-col. 4-AC-col	1.1/1.1, 6/1.6, 8mn/4mn
McGee	$T(F_{10})$	3-AC-col.	21h/11h

Table 24: Some results for coloring problems

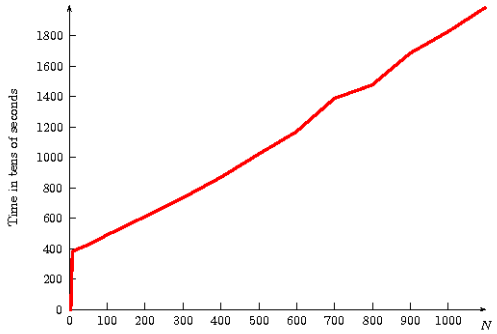

Figure 9: 3-colorability of grids $6 \times N$

For the grids $G_{6\times N}$ (they are denoted by terms in $T(F_8)$), we could reach $N > 1000$ by using a fly-automaton[23] Figure 9 shows the results for these grids.

9. 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.

These constructions extend to *counting problems* (e.g., how many p-colorings or p-AC-colorings has a given graph?) and *optimization problems* (e.g., what is the minimum number of vertices that must get color 1 for a p-coloring of a given graph?): the theoretical results of Chapter 6 of [9] should be implemented. These constructions also extend to graphs of bounded tree-width and MS properties written with edge quantifications by using the results of [7].

[23] A table-automaton can be obtained only for clique-width 2.

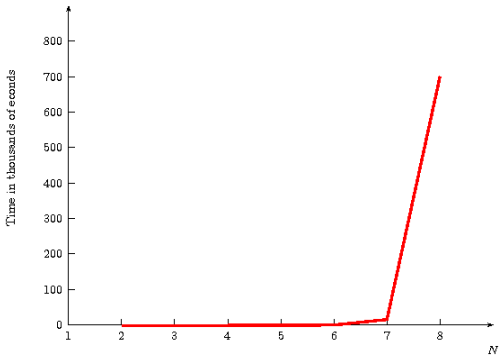

Figure 10: 3-colorability of square grids $N \times N$

References

[1] B.-M. Bui-Xuan, J. A. Telle and M. Vatshelle, H-join decomposable graphs and algorithms with runtime single exponential in rankwidth, *Discrete Applied Mathematic* **158** (2010) 809-819.

[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*, http://tata.gforge.inria.fr/

[5] B. Courcelle, The monadic second-order logic of graphs I: Recognizable sets of graphs, *Information and Computation* **85** (1990) 12-75.

[6] B. Courcelle, The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues, *Inform. Théor. et Applications* **26** (1992) 257-286.

[7] B. Courcelle, On the model-checking of monadic second-order formulas with edge set quantifications, *Discrete Applied Mathematics* **160** (2012) 866-887.

[8] B. Courcelle and I. Durand, Verifying monadic second-order graph properties with tree automata, *3rd European Lisp Symposium*, May 2010, Lisbon, Proceedings edited by C. Rhodes, pp. 7-21.

[9] B. Courcelle and J. Engelfriet, *Graph structure and monadic second-order logic, A language-theoretic approach*, *Cambridge University Press*, 2012.

[10] B. Courcelle, J. A. Makowsky and U. Rotics, Linear-Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory Comput. Syst.* **33** (2000) 125-150.

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

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

[13] I. Durand, *Autowrite: A Tool for Term Rewrite Systems and Tree Automata*, *ENTCS* **124** (2005) 29-49.

[14] M. Fellows, *et al.*, Clique-width is NP-Complete. *SIAM J. Discrete Math.* **23** (2009) 909-939.

[15] P. Flajolet, J.-C. Raoult and J. Vuillemin, The number of registers required for evaluating arithmetic expressions, Theoretical Computer Science **9** (1979) 99–125.

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

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

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

[19] R. Ganian, P. Hlinený and J. Obdrzálek, Better Algorithms for Satisfiability Problems for Formulas of Bounded Rank-width, FSTTCS, 2010, LIPIcs **8** (2010) pp. 73-83.

[20] F. Gécseg and M. Steinby, Tree languages, Chapter 1 of *Handbook of Formal Languages, Vol. 3: Beyond Words*, G. Rozenberg and A. Salomaa eds., Springer, 1997, pp. 1-68.

[21] M. Golumbic and U. Rotics, On the Clique-Width of Some Perfect Graph Classes. *Int. J. Found. Comput. Sci.* **11** (2000) 423-443.

[22] G. Gottlob, R. Pichler and F. Wei: Tractable database design and datalog abduction through bounded treewidth. *Inf. Syst.* **35** (2010) 278-298.

[23] G. Gottlob, R. Pichler and F. Wei: Monadic datalog over finite structures of bounded treewidth. ACM Trans. Comput. Log. **12**(1)(2010) 3.

[24] M. Kanté and M. Rao : F-Rank-Width of (Edge-Colored) Graphs, CAI 2011, Lecture Notes in Computer Science, 6742, (2011), pp. 158-173. Full version to appear in Discrete Applied Maths.

[25] J. Henriksen *et al.*, MONA: monadic second-order logic in practice, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science **1019** Springer, 1995, pp. 89-110.

[26] P. Hlinený and S. Oum: Finding Branch-Decompositions and Rank-Decompo-sitions. *SIAM J. Comput.* **38** (2008) 1012-1032.

[27] S. Oum and P. Seymour: Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B* **96** (2006) 514-528.

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

[29] K. Reinhardt, The complexity of translating logic to finite automata. in *Automata, Logics, and Infinite Games: A Guide to Current Research*. Lecture Notes in Computer Science **2500,** Springer, 2002, pp. 231-238.

[30] R. Sethi and J. Ullman: The Generation of Optimal Code for Arithmetic Expressions. *J. ACM* **17** (1970) 715-728.

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