Computations by fly-automata beyond monadic second-order logic

Bruno Courcelle and Irène Durand
LaBRI, Bordeaux University and CNRS,
351 Cours de la Libération, 33405 Talence, France
courcell@labri.fr ; idurand@labri.fr

May 30, 2013

Abstract

We present logically based methods for constructing XP and FPT graph algorithms, parametrized by tree-width or clique-width. We will use *fly-automata* introduced in a previous article. They make possible to check properties that are *not* monadic second-order expressible because their states may include counters, so that their sets of states may be infinite. We equip these automata with *output functions*, so that they can compute values associated with terms or graphs. Rather than new algorithmic results we present tools for constructing easily certain *dynamic programming algorithms* by combining predefined automata for basic functions and properties.

Keywords : monadic second-order logic, graph algorithm, fly-automaton, FPT algorithm, XP algorithm, clique-width.

1 Introduction

We provide logic based methods for constructing FPT and XP *dynamic programming algorithms* for terms and graphs, based on automata on terms. The system AUTOGRAPH, currently under development, implements the presented constructions.

Finite automata on terms that denote graphs of bounded tree-width or clique-width can be used to check monadic second-order properties of the denoted graphs. However, these automata have in most cases so many states that their transition tables cannot be built. In a previous article ([BCID]) we have introduced automata called *fly-automata* whose states are described (but not

listed) and whose transitions are computed on the fly (and not tabulated)[1]. Fly-automata can have infinite sets of states. For example, a state can record, among other things, the (unbounded) number of occurrences of a particular symbol. We exploit this feature for constructing fly-automata that check properties that are *not monadic second-order (MS) expressible*. Furthermore, we equip automata with *output functions*, that map accepting states to some effective domain $\mathcal{D}$ (e.g., the set of integers, or of pairs of integers, or the set of words over a fixed alphabet). Hence, a fly-automaton $\mathcal{A}$ defines a mapping from $T(F)$ (the set of terms over its signature F) to $\mathcal{D}$, and we construct automata that yield polynomial-time algorithms for these mappings. The height $ht(t)$ of a term t and the number of occurrences of a given symbol in this term are computable in this way. The *uniformity* of a term, *i.e.*, the property that all leaves of its syntactic tree are at the same depth, can be checked by a polynomial-time fly-automaton. It cannot be by a finite automaton.

Our main interest is actually for the case where F is the signature F_∞ of graph operations upon which the definition of clique-width is based and for fly-automata that define mappings from the graphs defined by terms in $T(F_\infty)$ to $\mathcal{D}$. The graphs of clique-width at most k are those denoted by the terms in $T(F_k)$ where F_k is a finite subset of F_∞. We will construct fly-automata that yield FPT and XP algorithms (definitions are in Section 2.5) for clique-width as parameter and that check graph properties that are *not expressible* in monadic second-order logic. Since the clique-width $cwd(G)$ of a simple graph G is bounded in terms of its tree-width $twd(G)$ (we have $cwd(G) \leq 2^{2twd(G)+2}+1$, [CouEng], Proposition 2.114), all our FPT and XP algorithms parameterized by clique-width will be also FPT and XP respectively for tree-width.

As in [BCID], we will construct fly-automata for basic functions and properties, e.g., the degree of a vertex or the regularity of graph. Then, we will consider more complex functions and properties written with these functions and properties (and the basic MS properties considered in [BCID]) and functional and logical constructors. An example is the possibility of partitioning the vertex set in two sets inducing regular subgraphs of the considered graph, expressed by $\exists X, Y.(Partition(X,Y) \wedge Reg[X] \wedge Reg[Y])$. The atomic formula $Reg[X]$ expresses that the induced subgraph $G[X]$ of G is regular, which is not MS expressible. We will combine basic fly-automata by means of products, direct and inverse images as in [BCID] to obtain fly-automata for the considered functions and properties. This method is extremely flexible because a slight change in the defining formula is quickly reflected in the construction of a new automaton, performed by the system AUTOGRAPH.

[1] Fly-automata are useful when the number of states is large compared to the number of function symbols and the size of input terms. *Symbolic automata* [VeaBjo] are defined for the case when the number of states is manageable but the set of function symbols is very large, and possibly infinite. In these automata, states are listed but function symbols and transitions are described by logical formulas..

Here are some typical examples of properties and functions that we can handle in this way:

(1) Is it possible to partition the vertex set in s components such that each of them induces a regular subgraph? Or to cover the edges of a graph with those of s cliques?

(2) Does there exist an equitable s-coloring ? *Equitable* means that the sizes of any two color classes differ by at most 1 ([Fell]). We can express this property by:

$$\exists X_1, \ldots, X_s.(Partition(X_1, \ldots, X_s) \wedge St[X_1] \wedge ... \wedge St[X_s] \\ \wedge |X_1| = ... = |X_{i-1}| \geq |X_i| = ... = |X_s| \geq |X_i| - 1)$$

where $St[X]$ means that $G[X]$ is *stable*, *i.e.,* has no edge.

(3) Assuming the graph s-colorable, what is the minimum size of the first color class of an s-coloring?

(4) What is the minimum number of edges between X and Y for a partition (X, Y) of the vertex set such that $G[X]$ and $G[Y]$ are connected ?

More generally, let $P(X_1, ..., X_s)$ be a property of sets of vertices $X_1, ..., X_s$ or of positions of a term; we will use $\overline{X}$ to denote $(X_1, ..., X_s)$ and $t \models P(\overline{X})$ to mean that $\overline{X}$ satisfies P in t (or in the graph $G(t)$ defined by t); this writing does not assume that P is written in any particular logical language. We are interested, not only to check the validity of $\exists \overline{X}.P(\overline{X})$ in some term t, but also to compute from it the following objects associated with t:

$\#\overline{X}.P(\overline{X})$, defined as the number of assignments $\overline{X}$ such that $t \models P(\overline{X})$,

$\mathrm{Sp}\overline{X}.P(\overline{X})$, the *spectrum* of $P(\overline{X})$, defined as the set of tuples of the form $(|X_1|, \ldots, |X_s|)$ such that $t \models P(\overline{X})$,

$\mathrm{MSp}\overline{X}.P(\overline{X})$, the *multispectrum* of $P(\overline{X})$, defined as the multiset of tuples $(|X_1|, \ldots, |X_s|)$ such that $t \models P(\overline{X})$,

$\mathrm{Sat}\overline{X}.P(\overline{X})$ as the set of assignments $\overline{X}$ such that $t \models P(\overline{X})$.

We will say that $\#\overline{X}.P(\overline{X})$, $\mathrm{Sp}\overline{X}.P(\overline{X})$, $\mathrm{MSp}\overline{X}.P(\overline{X})$ and $\mathrm{Sat}\overline{X}.P(\overline{X})$ are *MS expressible* functions on terms if $P(\overline{X})$ is an MS property. Their values are in the first three cases numbers or sets of tuples of numbers. The value of $\mathrm{Sat}\overline{X}.P(\overline{X})(t)$ is a set of s-tuples of subsets of $Pos(t)$. It can thus be described by a finite word since the positions of any term are so (they can be described by Dewey words, see Section 2.1), but this word may be of length $2^{s.|t|}$, hence, not computable by a polynomial-time algorithm. In most cases, our automata will give XP algorithms, and in some cases FPT algorithms.

Our main results

By continuing the work initiated in [BCID], we develop a novel vision of automata for the construction of model-checking algorithms that take finite terms

as inputs or, in other words, a *theory of automaton-based dynamic programing*. Here are the four main ideas and achievements.

First, a well-known fact: the state of a deterministic bottom-up automaton collects, at each position u of an input term, some information about the subterm issued from u. This information should be of size as small as possible so that the computation of a run be efficient. The best situation is when the set of states is finite, but finiteness alone does not guarantee efficient algorithms. This is well-known for monadic second-order definable sets of terms (over a finite signature): the number of states of an automaton that implements a monadic second-order formula has a number of states that cannot be bounded by an elementary function (an iterated exponentiation of bounded depth) in the size of the formula, see [FriGro, Rei]. The notion of fly-automaton permits to construct usable algorithms based on finite automata whose transitions cannot be compiled in manageable tables.

Second, as we do not insist on compiling transitions in tables, we have no reason to insist on finiteness for the set of states. So we use fly-automata over finite signatures, whose states are integers, or pairs of integers or some information representable by a finite word over a fixed finite alphabet. These automata yield p*olynomial-time dynamic programing algorithms* if the computation of each transition takes polynomial time in the size of the input term.

Third, fly-automata can take as input terms over countably infinite signatures (encoded in effective ways). This extension is motivated by the use of automata for checking properties of graphs defined by terms. In this article, we use the terms with which clique-width is defined. As no finite set of operations can generate all graphs, the use of an infinite signature is necessary[2]. By analyzing how these automata are constructed from logical descriptions and basic automata, we can understand (in part) why some algorithms constructed from automata are FPT whereas others are only XP.

Fourth, we go beyond monadic second-order logic in two ways. We adapt the (classical) construction of finite automata from monadic second-order formulas exposed in detail in Chapter 6 of [CouEng] and in [BCID] to properties of terms and graphs that are not MS expressible (for example we can handle the regularity of a graph) and we build automata that compute functions (for example, the largest size of an induced subgraph that is regular). Such properties and functions are defined by formulas using new atomic formulas, such as $Reg[X]$, and new constructions, such as $\#\overline{X}.P(\overline{X})$ or $\mathrm{Sp}\overline{X}.P(\overline{X})$, that can be seen as generalized quantifications (they bind the variables of $\overline{X}$ and deliver more information than $\exists\overline{X}.P(\overline{X})$). From the usual case of MS logic, we keep the inductive construction of an automaton based on the structure of the defining formula.

We generalize results from [ALS], [CouMos] and [CMR] that build algorithms for properties of terms or of graphs of bounded tree-width or clique-width that are of the form

[2] The corresponding constructed algorithms that are FPT (or XP) for clique-width are immediately FPT (or XP) for tree-width.

$$\exists X_1, \ldots, X_s.(\varphi(X_1, \ldots, X_s) \wedge R(|X_1|, \ldots, |X_s|))$$

where $\varphi(X_1, \ldots, X_s)$ is an MS formula and R is an s-ary arithmetic relation that can be decided in polynomial time can be checked in polynomial time. However, we cannot allow atomic formulas of the form $R(|X_1|, \ldots, |X_s|)$ (with R as above) to occur everywhere in formulas (and not, as above, just below external existential quantifiers). We discuss this issue in Section 5.3.

Summary of article : Section 2 reviews notation and definitions relative to terms and graphs, and to computability. Section 3 recalls from [BCID] (and extends) the definitions concerning fly-automata. Section 4 gives the main algorithms that operate on fly-automata. Section 5 details some applications to terms and graphs. (We also give "direct constructions" of smaller automata than those obtained by the general results). Section 6 gives an overview of the organization of AUTOGRAPH and reports about experiments. Section 7 is a conclusion. An appendix reviews definitions concerning monadic second-order logic and establishes a technical lemma about terms that define graphs.

Short versions of the theoretical aspects and of the implementation are in the proceedings, respectively, of CAI 2013 [BCID13] and of ELS 2013 [BCID13a].

2 Review of definitions

We assume that the reader is familiar with [BCID]. We review quickly definitions and notation, but the reader will find more details and comments in [BCID]. We also give some new definitions. Monadic second-order logic on terms and graphs is reviewed in the appendix.

2.1 General notation

We denote by $\mathbb{N}$ the set of natural numbers, by $\mathbb{N}_+$ the set of positive ones, by $[n, m]$ the interval $\{i \mid n \leq i \leq m\}$ and by $[n]$ the interval $[1, n]$. We denote by $w[i]$ the i-th element of a sequence or a word w. As usual, $\log(x)$ stands for $\max\{1, \log(x)\}$ and logarithms are in base 2.

The cardinality of a set A is denoted by $|A|$. An encoding of this set may be larger. For example, a set of m integers in $[n]$ can be encoded in size $O(m.\log(n) + 1)$ by a word over a fixed finite alphabet. We denote by $\|A\|$ the size of such an encoding. If A is any set, then $\mathcal{P}(A)$, $\mathcal{P}_f(A)$, $\mathcal{P}_n(A)$, $\mathcal{P}_{\leq n}(A)$ denote respectively its set of subsets, of finite subsets, of subsets of cardinality n and of subsets of cardinality at most n.

We denote by $A - B$ the difference of two sets A, B. We denote it by B^c (to mean complementation) if $B \subseteq A$ and A is clear from the context. (We will use the notation $\overline{B}$ for another purpose).

We denote by $[A \to B]$ the set of mappings (total functions) : $A \to B$. If $C \subseteq A$ and $f \in [A \to B]$, we denote by $f \restriction C$ the restriction of f to C. We can consider that $[C \to C]$ is a subset of $[A \to A]$ by identifying $h : C \to C$ with its extension h' to A such that $h'(a) := a$ if $a \in A - C$. However, no such identification is needed if we formalize (unambiguously) h as the set of pairs $(a, h(a))$ such that $a \in C$ and $h(a) \neq a$, because in this case the set of pairs corresponding to h' is exactly the same. This observation yields a way to implement h if C is finite and A is infinite.

We say that a mapping $h : A \to \mathbb{N}$ is *finite* if the set $h^{-1}(\mathbb{N}_+)$ is finite. It can be seen as a *finite multiset* over A. We denote by $[A \to \mathbb{N}]_f$ the set of finite mappings $h : A \to \mathbb{N}$. Even if A is infinite, such a mapping can be seen (and implemented) as the finite set of pairs $(a, h(a))$ such that $h(a) \neq 0$. If $A \subseteq B$ we can consider that $[A \to \mathbb{N}]_f$ is a subset of $[B \to \mathbb{N}]_f$: the mapping h in $[A \to \mathbb{N}]_f$ is identified with h' in $[B \to \mathbb{N}]_f$ that extends it by taking $h'(a) := 0$ if $a \in B - A$. But the set of pairs $(a, h(a))$ such that $a \in A$ and $h(a) \neq 0$ also describes h'.

Let $f \in [A \to B]$ and $X \subseteq A$. We denote by $\mathbf{\{}f(x) \mid x \in X\mathbf{\}}$ with boldface braces, the multiset of elements $f(x)$ of B and by $\{f(x) \mid x \in X\}$ the corresponding set. If $B = \mathbb{N}$, then $\Sigma\mathbf{\{}f(x) \mid x \in X\mathbf{\}} = \Sigma_{x \in X} f(x)$. We use in a similar way the notation $\oplus\mathbf{\{}f(x) \mid x \in X\mathbf{\}}$ if $\oplus$ is an associative and commutative binary operation on B.

We denote by $\sqcup$ the union of two multisets. We denote by $A_1 \uplus ... \uplus A_k$ the union of sets $A_1, ..., A_k$ if they are pairwise disjoint (otherwise $A_1 \uplus ... \uplus A_k$ is undefined).

Terms and their syntactic trees

A *signature* F is a finite or countable set of *function symbols*, each being given together with a natural number called its *arity*: we denote by $\rho(f)$ the arity of the symbol f and by $\rho(F)$ the maximal arity of a symbol of F, provided its symbols have bounded arity. We denote by $T(F)$ the set of finite *terms over* F and by $Pos(t)$ the set of positions of a term t. Each position is an occurrence of some symbol. Positions are defined as Dewey words. For example, the 7 positions of the term $f(g(a, b), g(b, c))$ are denoted by the Dewey words $\varepsilon, 1, 11, 12, 2, 21, 22$. For a term $t = h(t_1, t_2, t_3)$, we have $Pos(t) = \{\varepsilon\} \cup 1.Pos(t_1) \cup 2.Pos(t_2) \cup 3.Pos(t_3)$ where . denotes concatenation. (If function symbols have arity at most 9, we can omit the concatenation marks in Dewey words, as done in the above example of $f(g(a, b), g(b, c))$). We denote by $Sig(t)$ the finite subsignature of F consisting of the symbols that have occurrences in t.

The *syntactic tree* of a term t is a rooted, labelled and ordered tree with set of nodes $Pos(t)$. Each node u is labelled by a symbol f and has an ordered sequence of $\rho(f)$ sons. The root ($root_t$) is the first position (relative to some

linear writing of t), and the occurrences of the nullary symbols are the leaves. We denote by t/u the *subterm of t issued from a node u* and by $Pos(t)/u$ the set of positions of t below u or equal to it. In terms of Dewey words, we have $Pos(t)/u = u.Pos(t/u)$. Note that $Pos(t)/u \neq Pos(t/u)$ unless $u = \varepsilon$ i.e., is the root. If X is a set of positions of t, then X/u denotes $X \cap (Pos(t)/u)$, hence is the set of elements of X below or equal to u. The *height* $ht(t)$ of a term t is 1 if t is a nullary symbol and $1 + \max\{ht(t_1), ..., ht(t_r)\}$ if $t = h(t_1, ..., t_r)$.

Let H be another 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*.

We denote by $|t|$ the number of positions of a term t. In order to discuss algorithms taking terms as input, we must define the size of t. If F is finite, we can take $|t|$ as its size. If F is infinite, its symbols must be encoded by words of variable length. We define the size $\|t\|$ of t as the sum of lengths of the words that encode its symbols[3]. In both cases, we denote the size of t by $\|t\|$. We have $|Sig(t)| \leq |t| < \|t\|$. We say that an algorithm takes time poly($\|t\|$) if its computation time is bounded by $p(\|t\|)$ for some polynomial p that we need not specify.

The set $Pos(t)$ of all positions of $t \in T(F)$ can be encoded by a sequence of length bounded by $|t|.ht(t).\log(r) \leq \|t\|^2$ where $r = \rho(Sig(t))$ is the maximal arity of the symbols of t. (This sequence encodes the arity of the function symbol occurring at each position, but not the symbol itself.)

A *language* is either a set of words or a set of terms.

2.2 Graphs and clique-width

Notation and definitions are as in the book [CouEng]. More details are given in the appendix.

Graphs

All graphs are finite and simple (without parallel edges) and without loops. A graph G is identified with the relational structure $\langle V_G, edg_G \rangle$ where edg_G is a binary relation representing the directed or undirected adjacency (edg_G is symmetric if G is undirected). If $X \subseteq V_G$, we denote by $G[X]$ the induced subgraph of G with vertex set X, i.e., $G[X] := \langle X, edg_G \cap (X \times X) \rangle$. If E is a set of edges, $E \subseteq edg_G$, then $G[E] := \langle V_G, E \rangle$.

In order to build graphs by means of graph operations, we use labels attached to vertices. We let L be a fixed countable set of *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: $V_G \to L$.

[3] If w_x encodes a symbol x, then $\|f(g(a,b), g(b,c))\| = |w_f| + 2.|w_g| + |w_a| + 2.|w_b| + |w_c|$. The length of a LISP list implementing a term t is between $\|t\|$ and $3 \cdot \|t\|$. (We use LISP to implement fly-automata, see Section 6.)

So, $\pi_G(x)$ is the label of x and if $\pi_G(x) = a$, we say that x is an *a-port*. 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. A p-graph G is identified with the relational structure $\langle V_G, edg_G, (lab_{a\,G})_{a\in L}\rangle$ where lab_a is a unary relation and $lab_{a\,G}$ is the set of a-ports of G (see the appendix for more details). 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 port labels in $L := \mathbb{N}_+$.

We denote by $G \approx G'$ the fact that p-graphs G and G' are isomorphic and by $G \simeq G'$ that they are isomorphic up to port labels.

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 the set of port labels $C := [k]$:

- the binary symbol $\oplus$ denotes the union of two *disjoint* p-graphs (i.e., $G \oplus H := \langle V_G \cup V_H, edg_G \cup edg_H, (lab_{a\,G} \cup lab_{a\,H})_{a\in C}\rangle$ with $V_G \cap V_H = \emptyset$),

- 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 defined as a subset[4] of $C \times C$, as explained in Section 2.1);

- 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),

- the nullary symbol $\mathbf{a}$, for $a \in C$, denotes an isolated a-port, and the nullary symbol $\varnothing$ denotes the empty graph $\varnothing$.

The set $\{\mathbf{a} \mid a \in C\}$ is denoted by $\mathbf{C}$. For constructing undirected graphs, we 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 use the operations $add_{a,b}$ instead of $\overrightarrow{add}_{a,b}$, which yields the signature F_k^{u}.

Every operation of F_k (resp. F_k^{u}) is an operation of $F_{k'}$ (resp. $F_{k'}^{\mathrm{u}}$) if $k < k'$ (by our convention on mappings h in $relab_h$). We let F_∞ (resp. F_∞^{u}) be the union of the signatures F_k (resp. F_k^{u}). Hence, F_k (resp. F_k^{u}) is the restriction of F_∞ (resp. F_∞^{u}) to the operations involving labels in $[k]$.

Let $t \in T(F_\infty)$. We say that a port label a *occurs in* t if either $\mathbf{a}$, $\overrightarrow{add}_{a,b}$, $\overrightarrow{add}_{b,a}$ or $relab_h$ such that $h(a) \neq a$ or $h(b) = a \neq b$ has an occurrence in t. We

[4] If $k = 3$, then $relab_{\{(1,2),(3,1)\}} = relab_h$ where $h(1) = 2, h(2) = 2, h(3) = 1$. We denote also $relab_{\{(a,b)\}}$ by $relab_{a\to b}$. Each operation $relab_h$ can be expressed as a composition of operations $relab_{a\to b}$. See [CouEng] Proposition 2.118 for the proof.

denote by $\mu(t)$ the set of port labels that occur in t and by $\max \mu(t)$ its maximal element. We also denote by $\pi(t)$ the set of port labels $\pi(G(t))$ and by $\max \pi(t)$ its maximal element. Clearly, $\pi(t) \subseteq \mu(t)$.

Clique-width

Every term t in $T(F_k)$ (or in $T(F_k^{\mathrm{u}})$) denotes a p-graph, $G(t)$ that we now define formally. We let $Pos_0(t)$ be the set occurrences in t of the symbols from $\mathbf{C}$. For each $u \in Pos(t)$, we define a p-graph $G(t)/u$, whose vertex set is $Pos_0(t)/u$, i.e., the set of leaves of t below u that are not occurrences of $\varnothing$. The definition of $G(t)/u$ is by bottom-up induction on u.

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

if u is an occurrence of $\mathbf{a}$, then $G(t)/u$ has the unique vertex u that is a a-port,

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

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

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

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

Finally, $G(t) := G(t)/root_t$. Its vertex set is thus $Pos_0(t)$. Note the following facts :

(1) up to port labels, $G(t)/u$ is a subgraph of $G(t)$ (a port label of a vertex of $G(t)/u$ can be modified by a relabelling occurring on the path from u to the root of t);

(2) if u and w are incomparable under the ancestor relation, then $G(t)/u$ and $G(t)/w$ are disjoint graphs.

If $t \in T(F_k) \cup T(F_k^{\mathrm{u}})$, $X \subseteq Pos_0(t)$ and t' is the term obtained by replacing, for each $u \in X$, the symbol occurring there by $\varnothing$, then $G(t')$ is the induced subgraph $G(t)[Pos_0(t) - X]$ of $G(t)$.

The *clique-width* of a graph G, denoted by $cwd(G)$, is the least integer k such that $G \simeq G(t)$ for some t in $T(F_k)$ (in $T(F_k^{\mathrm{u}})$ if G is undirected). A term t in $T(F_k)$ (or in $T(F_k^{\mathrm{u}})$) is *optimal* if $k = cwd(G)$. Every graph G has clique-width at most $|V_G|$. Two terms t and t' are *equivalent*, denoted by $t \simeq t'$, if $G(t) \simeq G(t')$.

All definitions and results stated below for F_k and F_∞ apply also to F_k^{u} and F_∞^{u}. Let $t \in T(F_k)$. Each of its symbols can be encoded by a word of length $O(k.\log(k))$ (only $O(\log(k))$ for $\mathbf{a}$, $\overrightarrow{add}_{a,b}$). Hence, its size is $O(k.\log(k).|t|)$. Clearly, $|V_{G(t)}| \leq |t| < \|t\|$. The size $\|t\|$ is not bounded by a function of $|V_{G(t)}|$ because a graph can be denoted by arbitrary large terms (in particular because

$\overrightarrow{add}_{a,b}$ is idempotent). To avoid this, we define a term $t \in T(F_\infty)$ as *good* if, for some k, we have $t \in T(F_k)$, $k \leq |V_{G(t)}|$ and $|t| \leq (k+1)^2.|V_{G(t)}| + 1$. These conditions do not imply that t is optimal. We denote by $T_{\text{good}}(F_\infty)$ the set of good terms. In the appendix (Proposition 44), we give an algorithm that transforms a term $t \in T(F_k)$ into an equivalent good term in $T(F_{k'})$ for some $k' \leq k$. (Its proof will construct a kind of normal form that justifies the bound $(k+1)^2.|V_{G(t)}| + 1$ in the definition). For example the term $relab_{5\to 1}(add_{1,9}(add_{1,8}(\mathbf{1} \oplus \mathbf{5} \oplus \mathbf{8})))$ is not good and can be replaced by the good term $relab_{2\to 1}(add_{1,3}(\mathbf{1} \oplus \mathbf{2} \oplus \mathbf{3}))$. This preprocessing takes time poly($\|t\|$).

If t is good we have $\|t\| = O(k.\log(k).|t|) = O(k^3.\log(k).|V_{G(t)}|)$. It follows that a computation time is bounded by a polynomial in $\|t\|$ if and only if it is by a polynomial in $\max \mu(t) + |V_{G(t)}|$. In Section 2.5 below, we discuss complexity issue about clique-width.

2.3 Encoding sets of positions of terms and sets of vertices

A property of terms in $T(F)$ can be seen as a subset of this set. We recall from [BCID] that, similarly, a property $P(X_1, ..., X_s)$ of sets of positions $X_1, ..., X_s$ of a term can be seen as a set of terms over a set of function symbols constructed from F in a canonical way. We can thus handle these properties by automata.

Sets of positions of terms.

Let F be a signature and s be a positive integer. We let $F^{(s)}$ be the set $F \times \{0,1\}^s$ made into the signature by letting $\rho((f,w)) := \rho(f)$ (here $w \in \{0,1\}^s$ is a sequence of Booleans considered as a word over $\{0,1\}$). We let $pr_s : F^{(s)} \to F$ be the relabelling that deletes the second component of a symbol (f,w). (We denote it by pr if s need not be specified.)

To every term $t \in T(F^{(s)})$ corresponds the term $pr_s(t)$ in $T(F)$ and the s-tuple $\nu(t) = (X_1, ..., X_s)$ of subsets of $Pos(t) = Pos(pr_s(t))$ such that $u \in X_i$ if and only if $w[i] = 1$ where the symbol at position u in t is (f,w). Conversely, if $t \in T(F)$ and $(X_1, ..., X_s)$ is an s-tuple of sets of positions of t, then there is a unique term $t' \in T(F^{(s)})$ such that $pr_s(t') = t$ and $\nu(t') = (X_1, ..., X_s)$.We will denote this term by $t*(X_1, ..., X_s)$ or by $t*\overline{X}$ where $\overline{X}$ abbreviates $(X_1, ..., X_s)$.

A property $P(X_1, ..., X_s)$ of sets of positions of terms over a signature F is characterized by the language $T_{P(X_1,...,X_s)}$ over $F^{(s)}$ defined as $\{t * \overline{X} \mid t \models P(\overline{X})\}$[5]. Such a property can also be considered as the property $\overline{P}$ of terms $t*\overline{X}$ in $T(F^{(s)})$ such that $t*\overline{X} \models \overline{P}$ if and only if $t \models P(\overline{X})$. Conversely, every subset of $T(F^{(s)})$ is $T_{P(\overline{X})}$ for some property $P(\overline{X})$.

A key fact about the relabelling pr_s is that $T_{\exists \overline{X}.P(\overline{X})} = pr_s(T_{P(\overline{X})})$.

More generally (because every property is a Boolean-valued function) a function α whose arguments are $t \in T(F)$ and s-tuples $\overline{X}$ of positions of t, and whose values are in a set $\mathcal{D}$, corresponds to the function $\overline{\alpha} : T(F^{(s)}) \to \mathcal{D}$ such that $\overline{\alpha}(t * \overline{X}) = \alpha(t, \overline{X})$.

[5] Obviously, $P(\overline{X})$ stands for $P(X_1, ..., X_s)$ and not for $P((X_1, ..., X_s))$.

Sets of vertices.

The same technique applies to sets of vertices of graphs defined by terms in $T(F_\infty)$.We define $F_\infty^{(s)}$ from F_∞ by replacing each nullary symbol $\mathbf{a}$ by the nullary symbols $(\mathbf{a}, w)$ for all $w \in \{0,1\}^s$. We define $pr_s : F_\infty^{(s)} \to F_\infty$ as the mapping that deletes the sequences w from these nullary symbols. It extends into a relabelling $pr_s : T(F_\infty^{(s)}) \to T(F_\infty)$. A term t in $T(F_\infty^{(s)})$ defines thus the graph $G(pr_s(t))$ and the s-tuple $(X_1, ..., X_s)$ such that X_i is the set of vertices that are occurrences of nullary symbols $(\mathbf{a}, w)$ such that $w[i] = 1$. As for sets of positions in terms, we use the notation $t * (X_1, ..., X_s)$ or more shortly $t * \overline{X}$. Hence, a property $P(X_1, ..., X_s)$ of sets of vertices is characterized by the language $L_{P(X_1,...,X_s)}$ over $F_\infty^{(s)}$ defined as $\{t * \overline{X} \mid G(t) \models P(\overline{X})\}$. Such a property can also be considered as the property $\overline{P}$ of terms in $T(F_\infty^{(s)})$ such that $t * \overline{X} \models \overline{P}$ if and only if $G(t) \models P(\overline{X})$.

As for terms, the same holds for functions on graphs taking sets of vertices as auxiliary arguments. For example, the function $e(X_1, X_2)$ that we define as the number of undirected edges between disjoint sets X_1 and X_2 (with value $\perp$ meaning "undefined" if X_1 and X_2 are not disjoint) can be handled as a mapping $\overline{e} : T(F_\infty^{\mathrm{u}(2)}) \to \{\perp\} \cup \mathbb{N}$. (It will be used in Section 5.2.1).

Set terms and substitutions of variables.

The following definitions and facts apply to terms and to graphs. 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 ^{c} for complementation. Hence, $\varnothing^c$ denotes the set of all positions of the considered term or of all vertices of the considered graph. An example is $(X_1 \cup X_3^c) \cap (X_2 \cup X_5)^c$. We now extend to functions Lemma 13 of [BCID] which is formulated for the special case of properties. Let $\alpha(Y_1, ..., Y_m)$ be a function on terms in $T(F)$ with set arguments $Y_1, ..., Y_m$ and values in a set $\mathcal{D}$. Let $S_1, ..., S_m$ be set terms over $\{X_1, ..., X_n\}$. We let $\beta(X_1, ..., X_n) := \alpha(S_1, ..., S_m)$. Hence $\overline{\alpha} : T(F^{(m)}) \to \mathcal{D}$ and $\overline{\beta} : T(F^{(n)}) \to \mathcal{D}$. There exists a relabelling h: $T(F^{(n)}) \to T(F^{(m)})$ that replaces each symbol (f, w) by (f, w') for some $w' \in \{0,1\}^m$ in such a way that $\overline{\beta} = \overline{\alpha} \circ h$.

We give two examples. First, we let $n := 4$, $S_1 := X_1 \cup X_3^c$ and α be unary ($m = 1$). Then $\beta(X_1, X_2, X_3, X_4)$ defined as $\alpha(X_1 \cup X_3^c)$ satisfies the equality $\overline{\beta} = \overline{\alpha} \circ h$ with h defined as follows, for all $x, y \in \{0,1\}$ and[6] $f \in F$:

$$h((f, 1x0y)) = h((f, 1x1y)) = h((f, 0x0y)) = (f, 1) \text{ and}$$
$$h((f, 0x1y)) = (f, 0),$$

i.e., $h((f, x_1x_2x_3x_4)) = (f, x_1 \vee \neg x_3)$. Hence h encodes the set term S_1 in a natural way. For another example, consider $\alpha(Y_1, Y_2, Y_3)$ and $\beta(X_1)$ defined as $\alpha(X_1, \varnothing, \varnothing^c)$. Then we have $\overline{\beta} = \overline{\alpha} \circ h$ with h such that :

$h((f, 0)) = (f, 001)$ and $h((f, 1)) = (f, 101)$, hence, $h((f, x)) = (f, x01)$ for all $x \in \{0,1\}$.

[6] h is defined in the very same way for all f in F.

This technique 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.

First-order variables

If $P(X, Y, Z)$ is a property, we will denote by $P(X, y, Z)$ the property $P(X, \{y\}, Z)$ where y denotes an element of the considered domain. This is actually a derived notation. Our basic syntax uses only set variables. Accordingly, $\exists y.P(X,\ y, Z)$ abbreviates $\exists Y.(P(X, Y, Z) \wedge Sgl(Y))$ where $Sgl(Y)$ means that Y is singleton. If α is a ternary function, we let similarly $\alpha(X, y, Z)$ abbreviate $\alpha(X, \{y\}, Z)$.

2.4 Effectively given sets

A set is *effectively given* if, either it is finite and the list of its elements is known, or it is countable and its elements are bijectively encoded by words over $\{0, 1\}$ (or over another finite alphabet) that form a decidable set. Such elements can be the symbols of a signature, the Boolean constants $True$, $False$, an integer, a tuple of integers, or even a set of positions of a term t (a position is defined as a Dewey sequence, cf. Section 2.2). Through the encodings, one can define *computable functions* on effectively given sets. If $\mathcal{D}$ is effectively given, then so are $\mathcal{D}^s$ and $\mathcal{P}_f(\mathcal{D})$.

We let $\mathbb{B} := \{False, True\}$ or $\{0, 1\}$ with 0 for $False$. A property of terms or graphs is a function from terms or graphs to $\mathbb{B}$. The general definitions for functions apply thus to properties.

In many cases, an effectively given set $\mathcal{D}$ has a special element $\perp$ that can stand for an undefined value, or be 0 if $\mathcal{D} = \mathbb{N}$, or the empty set if $\mathcal{D} = \mathcal{P}_f(\mathcal{E})$. Then, a mapping $f : \mathcal{D}' \to \mathcal{D}$ is *finite* if the set of elements d of $\mathcal{D}'$ such that $f(d) \neq \perp$ is finite. Such a mapping can be identified with the finite set $\{(d, f(d)) \mid f(d) \neq \perp\}$. Hence, if $\mathcal{D}'$ is effectively given, the set $[\mathcal{D}' \to \mathcal{D}]_f$ of finite mappings : $\mathcal{D}' \to \mathcal{D}$ is effectively given.

We will consider terms over *finite* or *countably infinite signatures* F that satisfy the following conditions (that we will not repeat in definitions and statements):

(a) F is effectively given,

(b) its symbols have bounded arity ($\rho(F)$ denotes the maximal arity),

(c) the arity of a symbol can be computed in constant time.

To insure (c), we can begin the word encoding a symbol by its arity. It follows that one can determine in linear time if a labelled tree is actually the syntactic tree of a "well-formed" term in $T(F)$. We will only use relabellings: $F \to F'$ that are computable in linear time. Their extensions : $T(F) \to T(F')$ are also computable in linear time (by our definition of the size of a term in Section 2.1).

Functions from $T(F^{(s)})$ to $\mathcal{D}$ are always total: the symbol $\perp$ will stand for an undefined value.

2.5 Parameterization

We give definitions relative to parameterized complexity. Let F be a signature. A function $h : T(F) \to \mathbb{N}$ is P-*bounded* if there exists a polynomial p such that $h(t) \leq p(\|t\|)$ for every term t in $T(F)$. It is FPT-*bounded* if, similarly, $h(t) \leq f(Sig(t)).\|t\|^a$ for some fixed function f and constant a. It is XP-*bounded* if $h(t) \leq f(Sig(t)).\|t\|^{g(Sig(t))}$ for some fixed functions f and g. Since $|t| < \|t\| \leq |t|.\ell(Sig(t))$ for some function ℓ, $\|t\|$ can be replaced by $|t|$ in the last two cases.

A function $\alpha : T(F) \to \mathcal{D}$ is **P**-*computable* (resp. **FPT**-*computable*, resp. **XP**-*computable*) if it has an algorithm whose computation time is P-bounded (resp. FPT-bounded, resp. XP-bounded). Hence, we use $Sig(t)$ as a parameter in the sense of *parameterized complexity* (see the books [DF], [FG]). If F is finite, the three notions are equivalent. If α is a property, we say that it is **P**-, **FPT**- or **XP**-*decidable*.

We will consider graph algorithms that take as input terms t over F_∞ that define input graphs $G(t)$. By constructing automata, we will obtain algorithms that are polynomial-time, FPT or XP for $Sig(t)$ as parameter. The size of the input is $\|t\|$. What do we get if the graph is given without any defining term t, so that we must construct a term defining it? We get algorithms with same parameterized time complexity for the following reasons.

First we observe that every graph with n vertices is defined by a (trivial) term in $T(F_n)$ where each vertex has a distinct label and no relabelling is made. Such a term can be constructed in polynomial time in n so as to be good (in the technical sense of Section 2.2) and irredundant, whence of size $O(n^2.\log(n))$. Hence, if a function α on graphs whose input is given by a term is P-computable, then it is also P-computable if the graph is given without any defining term.

The situation is more complicated for FPT- and XP-computability. The problem of deciding if a graph has clique-width at most k is **NP**-complete (with k part of the input, [Fell]). However, finding an optimal term that defines the input graph is not essential. There is an algorithm that computes, for every directed or undirected graph G, a term in $T(F_{h(cwd(G))})$ defining this graph; this algorithm takes time $g(cwd(G)).|V_G|^3$ where g and h are fixed functions and can produce a good (not necessarily optimal) term (see [CouEng], Proposition 6.8). It follows that an FPT or XP graph algorithm taking as input a term in $T(F_\infty)$ yields an FPT or XP graph algorithm (for clique-width as parameter) taking a graph as input.

3 Fly-automata

3.1 Fly-automata : basic definitions

We review the main definitions of [BCID] and we extend them by equipping automata with output functions. All automata are bottom-up (or frontier-to-root) without ε-transition.

Definitions 1 : *Fly-automata that recognize languages.*

(a) Let F be a finite or infinite (effectively given) signature. A *fly-automaton* over F (in short, a FA over F) 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 infinite, effectively given set called the set of *states*, $Acc_\mathcal{A}$ is a computable mapping $Q_\mathcal{A} \to \{True, False\}$ so that $Acc_\mathcal{A}^{-1}(True)$ is the (usual) set of *accepting states*, and $\delta_\mathcal{A}$ is a computable function such that for each tuple $(f, q_1, \ldots, q_m)$ with $q_1, \ldots, q_m \in Q_\mathcal{A}$, $f \in F$, $\rho(f) = m \geq 0$, $\delta_\mathcal{A}(f, q_1, \ldots, q_m)$ is a finite set of states. The *transitions* are such that $f[q_1, \ldots, q_m] \to_\mathcal{A} q$ (and $f \to_\mathcal{A} q$ if f is nullary) if and only if $q \in \delta_\mathcal{A}(f, q_1, \ldots, q_m)$. We say that $f[q_1, \ldots, q_m] \to_\mathcal{A} q$ is a transition that *yields* q. We assume that each set $\delta_\mathcal{A}(f, q_1, \ldots, q_m)$ is linearly ordered for some fixed (say lexicographic) linear order on Z^*, where Z is the alphabet used to denote the states.

Each state is a word over a fixed alphabet and so has a size defined as the length of that word. An integer can be a state (written in binary notation, see Example 5 below).

We say that $\mathcal{A}$ is *finite* if F and $Q_\mathcal{A}$ are finite[7]. If furthermore, $Q_\mathcal{A}$, its accepting states and its transitions are listed in tables, it is called a *table-automaton*.

(b) A *run* of a FA $\mathcal{A}$ 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 $f \to_\mathcal{A} r(u)$).

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 the state $r(root_t)$ is accepting. We let $L(\mathcal{A}) := \bigcup\{L(\mathcal{A}, q) \mid Acc_\mathcal{A}(q) = True\} \subseteq T(F)$. It is the *language accepted* (or *recognized*) by $\mathcal{A}$. A state q is *accessible* if $L(\mathcal{A}, q) \neq \emptyset$. We denote by $Q_\mathcal{A} \restriction t$ the set of states that occur on the runs on t and on its subterms, and by $Q_\mathcal{A} \restriction L$ the union of the sets $Q_\mathcal{A} \restriction t$ for t in $L \subseteq T(F)$.

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,

[7] Every MS definable set of terms over a finite signature F is recognizable by a finite automaton. The case of an infinite signature is discussed in the appendix.

then an automaton can have at most one sink. A state named *Error* will be a nonaccepting sink.

Unless $\mathcal{A}$ is finite, one cannot decide if a state is accessible. Hence, we cannot perform on FA the classical trimming operation that removes the inaccessible states, which raises no problem as we will see next.

(c) *Deterministic automata.* A FA $\mathcal{A}$ is *deterministic* if all sets $\delta_{\mathcal{A}}(f, q_1, \ldots, q_m)$ have cardinality 1. Hence, to simplify the terminology, deterministic will mean deterministic *and complete* (cf. [BCID]; by adding a nonaccepting sink to a (usual) deterministic automaton, one makes it complete while preserving determinism). A deterministic FA $\mathcal{A}$ has, on each term t, a unique run denoted by $run_{\mathcal{A},t}$; we let also $q_{\mathcal{A}}(t) := run_{\mathcal{A},t}(root_t)$. The mapping $q_{\mathcal{A}}$ is computable and the membership in $L(\mathcal{A})$ of a term t is decidable.

Every fly-automaton $\mathcal{A}$ over F can be *determinized* as follows. For every term $t \in T(F)$, we denote by $run^*_{\mathcal{A},t}$ the mapping: $Pos(t) \to \mathcal{P}_f(Q_{\mathcal{A}})$ that associates with every position u the finite set of states of the form $r(root_{t/u})$ for some run r on the subterm t/u of t. If $\mathcal{A}$ is finite, then $run^*_{\mathcal{A},t} = run_{\mathcal{B},t}$ where $\mathcal{B}$ is its (classical) determinized automaton, denoted by $\det(\mathcal{A})$, with set of states included in $\mathcal{P}_f(Q_{\mathcal{A}})$. If $\mathcal{A}$ is an infinite fly-automaton, then we have the same equality where $\mathcal{B}$ is a deterministic fly-automaton with set of states $\mathcal{P}_f(Q_{\mathcal{A}})$ that we denote also by $\det(\mathcal{A})$. (See [BCID], Proposition 45(2)). In both cases, a run of $\det(\mathcal{A})$ is called *the determinized run* of $\mathcal{A}$. The mapping $run^*_{\mathcal{A},t}$ is computable and the membership in $L(\mathcal{A})$ of a term in $T(F)$ is decidable because $t \in L(\mathcal{A})$ if and only if the set $run^*_{\mathcal{A},t}(root_t)$ contains an accepting state. We define $ndeg_{\mathcal{A}}(t)$, the *nondeterminism degree of* $\mathcal{A}$ *on* t, as the maximal cardinality of $run^*_{\mathcal{A},t}(u)$ for u in $Pos(t)$. We have $ndeg_{\mathcal{A}}(t) \leq |Q_{\mathcal{A}} \restriction t|$.

It is not decidable whether a FA $\mathcal{A}$ is deterministic. If it is, then $\det(\mathcal{A})$ is not identical to $\mathcal{A}$: its states are the singletons $\{q\}$ such that $q \in Q_{\mathcal{A}}$. Clearly, $\mathcal{A}$ and $\det(\mathcal{A})$ have isomorphic runs and they recognize the same languages. In practice, we know from its construction when a FA is deterministic.

Whether all states of a FA are accessible or not does not affect the membership algorithm: the inaccessible states will simply never appear in any run. There is no need to try to remove them (which is actually impossible in general) as this is the case for a table-automaton, to get small transition tables.

Definitions 2 : *Fly-automata that compute functions.*

A fly-automaton *with output* is a 4-tuple $\mathcal{A} = \langle F, Q_{\mathcal{A}}, \delta_{\mathcal{A}}, Out_{\mathcal{A}} \rangle$ as in Definition 1 except that the mapping $Acc_{\mathcal{A}}$ is replaced by a total and computable *output function* $Out_{\mathcal{A}}$: $Q_{\mathcal{A}} \to \mathcal{D}$ where $\mathcal{D}$ is an effectively given domain. If $Error \in Q_{\mathcal{A}}$, we also require that $Out_{\mathcal{A}}(Error) = \bot$.

If $\mathcal{A}$ is deterministic, the *function computed by* $\mathcal{A}$ is $Comp(\mathcal{A}) : T(F) \to \mathcal{D}$ such that $Comp(\mathcal{A})(t) := Out_{\mathcal{A}}(q_{\mathcal{A}}(t))$. It can have value $\bot$ on a state different from $Error$.

If $\mathcal{A}$ is not deterministic, we let $\mathcal{B}$ be $\det(\mathcal{A})$ equipped with the output function $Out_{\mathcal{B}}{:}\mathcal{P}_f(Q_{\mathcal{A}}) \to \mathcal{P}_f(\mathcal{D})$ such that $Out_{\mathcal{B}}(R) := \{Out_{\mathcal{A}}(q) \mid q \in R\}$. Then, we define $Comp(\det(\mathcal{A}))$ as $Comp(\mathcal{B}) : T(F) \to \mathcal{P}_f(\mathcal{D})$. In some cases, we may equip $\det(\mathcal{A})$ with an output function $f : \mathcal{P}_f(Q_{\mathcal{A}}) \to \mathcal{D}'$ such that $\mathcal{D}'$ is another effectively given domain, so that the corresponding function maps $T(F)$ into $\mathcal{D}'$. In this article, most FA with output will be deterministic.

Examples 3:
(a) The height $ht(t)$ of a term t is obviously computable by a FA.

(b) Let F be a signature and $r = \rho(F)$ be the maximal arity of its operations. Let $f \in F$. For every term t over F, we let $Pos_f(t)$ be the set of positions of t that are occurrences of f. The function Pos_f is computed by a FA $\mathcal{A}_f$ that we construct as follows: its states are the finite sets of words over $[r]$. These sets are intended to denote sets of positions of terms in $T(F)$. The transitions are defined as follows, for $q_1, ... \subseteq [r]^*$:

$$f[q_1, ..., q_s] \to \{\varepsilon\} \cup 1.q_1 \cup ... \cup s.q_s,$$
$$f'[q_1, ..., q_{s'}] \to 1.q_1 \cup ... \cup s'.q_{s'} \quad \text{if } f' \neq f.$$

This automaton is deterministic. At each position u of t, we have $run_{\mathcal{A}_f,t}(u) = Pos_f(t/u)$, hence, we have $Comp(\mathcal{A}_f) = Pos_f$ if we take the identity as output function.

(c) For $t \in T(F_\infty)$, the finite sets of port labels $\pi(t)$ and $\mu(t)$ (cf. Section 2.2) can be computed by FA whose states on t are finite subsets of $\mu(t)$. Hence, the set $T_{\text{good}}(F_\infty)$ is recognizable by a fly-automaton.

Definitions 4 : *Subautomata; products and other transformations of automata .*

(1) *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 a fly-automaton $\mathcal{B}$ over H is a *subautomaton* of a fly-automaton $\mathcal{A}$ over F, which we denote by $\mathcal{B} \subseteq \mathcal{A}$, if:

$H \subseteq F$, $Q_{\mathcal{B}} \subseteq Q_{\mathcal{A}}$,
$\delta_{\mathcal{B}}$ is the restriction of $\delta_{\mathcal{A}}$ to tuples $(f, q_1, \ldots, q_r)$ such that $f \in H$ and $q_1, \ldots, q_r \in Q_{\mathcal{B}}$,
and, either $Acc_{\mathcal{B}} = Acc_{\mathcal{A}} \restriction Q_{\mathcal{B}}$ or $Out_{\mathcal{B}} = Out_{\mathcal{A}} \restriction Q_{\mathcal{B}}$.

These definitions imply that $L(\mathcal{B}) = L(\mathcal{A}) \cap T(H)$ in the first case and $Comp(\mathcal{B}) = Comp(\mathcal{A}) \restriction T(H)$ in the second. If $\mathcal{A}$ is fly-automaton over F and $H \subseteq F$, we define $\mathcal{A} \restriction H := \langle H, Q_{\mathcal{A}}, \delta_{\mathcal{A}\restriction H}, Out_{\mathcal{A}}\rangle$ where $\delta_{\mathcal{A}\restriction H}$ is the restriction of $\delta_{\mathcal{A}}$ to tuples $(f, q_1, \ldots, q_r)$ such that $f \in H$. It is a subautomaton of $\mathcal{A}$. Note that some states of $\mathcal{A} \restriction H$ are not accessible, but we need not mind.

The *Weak Recognizability Theorem* ([CouEng], Chapters 5 and 6) states that, for every MS formula φ expressing a graph property, for every integer k, one can construct a finite automaton $\mathcal{A}_k$ over F_k that recognizes the set of terms $t \in T(F_k)$ such that $G(t) \models \varphi$. In [BCID] (Section 7.3.1) we prove a bit more: we construct a (single) fly-automaton $\mathcal{A}_\infty$ on F_∞ that recognizes the terms $t \in T(F_\infty)$ such that $G(t) \models \varphi$. The automata $\mathcal{A}_k$ are subautomata of $\mathcal{A}_\infty$.

(2) *Products of fly-automata.* Let $\mathcal{A}_1, ..., \mathcal{A}_k$ be FA over F and g be a computable mapping from $Q_{\mathcal{A}_1} \times ... \times Q_{\mathcal{A}_k}$ to $\mathbb{B}$ or to some effectively given domain. We define $\mathcal{A} := \mathcal{A}_1 \times_g ... \times_g \mathcal{A}_k$ as the FA with set of states $Q_{\mathcal{A}_1} \times ... \times Q_{\mathcal{A}_k}$, transitions defined by:

$$\delta_{\mathcal{A}}(f, \overline{q_1}, \ldots, \overline{q_{\rho(f)}}) := \{(p_1, \ldots, p_{\rho(f)}) \mid p_i \in \delta_{\mathcal{A}_i}(f, \overline{q_1}[i], \ldots, \overline{q_{\rho(f)}}[i]) \text{ for each } i\}$$

and output function defined by:

$$Out_{\mathcal{A}}((p_1, \ldots, p_k)) := g(p_1, \ldots, p_k).$$

Depending on g, $\mathcal{A}$ recognizes a language or defines a function. This construction will be used in different ways : see Propositions 10 and 15 and Theorem 17 below.

(3) *Output composition.* Let $\mathcal{A}$ define a mapping : $T(F) \to \mathcal{D}$ and g be computable: $\mathcal{D} \to \mathcal{D}'$ and such that $g(\bot_{\mathcal{D}}) = \bot_{\mathcal{D}'}$. We denote by $g \circ \mathcal{A}$ the automaton obtained from $\mathcal{A}$ by replacing $Out_{\mathcal{A}}$ by $g \circ Out_{\mathcal{A}}$. If $\mathcal{A}$ is deterministic, we have $Comp(g \circ \mathcal{A}) = g \circ Comp(\mathcal{A})$. Otherwise, we have $Comp(\det(g \circ \mathcal{A})) = Comp(\widehat{g} \circ \det(\mathcal{A})) = \widehat{g} \circ Comp(\det(\mathcal{A}))$ where $\widehat{g}(D) := \{g(d) \mid d \in D\}$. When $\mathcal{A}$ is not deterministic, we may want to compute $Comp(k \circ \det(\mathcal{A}))$ for mappings $k : \mathcal{P}_f(\mathcal{D}) \to \mathcal{D}'$. Some examples are

$k(D) := D - \{\bot_{\mathcal{D}}\}$ with $\mathcal{D}' = \mathcal{P}_f(\mathcal{D} - \{\bot_{\mathcal{D}}\})$,

$k(D) := \max(D)$ where $\mathcal{D}$ is linearly ordered, $\mathcal{D}' = \mathcal{D} \cup \{\bot\}$, $(\max(\emptyset) = \bot)$,

$k(D) := \Sigma(D)$ where $D \subseteq \mathbb{N}$, $\mathcal{D}' = \mathbb{N}$.

(4) *Image.* Let $h : T(H) \to T(F)$ be a relabelling having a *computable inverse*, that is, such that $h^{-1}(f) \subseteq H$ is finite and computable for each f. If $L \subseteq T(H)$, then $h(L) := \{h(t) \mid t \in L\}$. If $\mathcal{A}$ is a fly-automaton over H, we let $h(\mathcal{A})$ be the fly-automaton over F (see [BCID], Proposition 45) obtained from $\mathcal{A}$ by replacing each transition $f[q_1, \cdots, q_{\rho(f)}] \to_{\mathcal{A}} q$ by $h(f)[q_1, \cdots, q_{\rho(f)}] \to q$. We say that $h(\mathcal{A})$ is the *image of* $\mathcal{A}$ under h. The FA $h(\mathcal{A})$ is not deterministic in general, even if $\mathcal{A}$ is. We have $h(L(\mathcal{A}, q)) = L(h(\mathcal{A}), q)$ for every state q and, if $\mathcal{A}$ defines a language, $h(L(\mathcal{A})) = L(h(\mathcal{A}))$ (because $h(\mathcal{A})$ has the same accepting states as A). If $\mathcal{A}$ computes a function, then $Comp(\det(h(\mathcal{A})))(t)$ is equal to $\{Comp(\mathcal{A})(t') \mid h(t') = t\}$ if $\mathcal{A}$ is deterministic, and to $\bigcup\{Comp(\det(\mathcal{A}))(t') \mid h(t') = t\}$ otherwise.

(5) *Inverse image.* Let $h : T(H) \to T(F)$ be a computable relabelling. If $K \subseteq T(F)$, we define $h^{-1}(K)$ as $\{t \in T(H) \mid h(t) \in K\}$. If $\mathcal{A}$ is a FA over F, we let $h^{-1}(\mathcal{A})$ be the FA over H 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. If $\mathcal{A}$ defines a language, then $L(h^{-1}(\mathcal{A})) = h^{-1}(L(\mathcal{A}))$. We call $h^{-1}(\mathcal{A})$ the *inverse image* of $\mathcal{A}$ under h (cf. [BCID], Definition 17(h)). Note that $h^{-1}(\mathcal{A})$ is deterministic if $\mathcal{A}$ is so. If $\mathcal{A}$ defines a function : $T(F) \to \mathcal{D}$, then $Comp(h^{-1}(\mathcal{A})) = Comp(\mathcal{A}) \circ h$: $T(H) \to \mathcal{D}$.

Example 5 : *The number of runs of a nondeterministic FA.*

We now present a construction that we will generalize below (Example 23 and Definition 27). Let $\mathcal{A}$ be a nondeterministic FA over F without output. For each $t \in T(F)$, we define $\#_{AccRun}(t)$ as the number of accepting runs of $\mathcal{A}$ on t. We define a deterministic FA $\mathcal{B}$ that computes $\#_{AccRun}$. We define it from $\det(\mathcal{A})$ in such a way that, for each term t, if $q_{\det(\mathcal{A})}(t) = \{q_1, ..., q_p\}$, then $q_{\mathcal{B}}(t) = \{(q_1, m_1), ..., (q_p, m_p)\}$ where m_i is the number of runs of $\mathcal{A}$ that yield q_i at the root of t. We can also consider such a state as the finite mapping μ:$Q_{\mathcal{A}} \to \mathbb{N}$ such that $\mu(q_i) = m_i$ and $\mu(q) = 0$ if $q \notin \{q_1, ..., q_p\}$ (cf. Section 2.1). As output function, we take $Out_{\mathcal{B}}(\mu) := \Sigma\{\mu(q) \mid Acc_{\mathcal{A}}(q) = True\}$. Some typical transitions are as follows, with states handled as finite mappings:

$$
\begin{aligned}
&a \to \mu \text{ such that, for each } q \in Q_{\mathcal{A}},\\
&\qquad \mu(q) = \texttt{if } a \to_{\mathcal{A}} q \texttt{ then } 1 \texttt{ else } 0,\\
&f[\mu_1, \mu_2] \to \mu \text{ such that, for each } q \in Q_{\mathcal{A}},\\
&\qquad \mu(q) = \Sigma\{\mu_1(q_1).\mu_2(q_2) \mid f[q_1, q_2] \to_{\mathcal{A}} q\}.
\end{aligned}
$$

If $\mathcal{A}$ has nondeterminism degree d on a term t, then it has at most $|t|^d$ runs. The size of a state of $\mathcal{B}$ on this term is thus $O(d^2.\log(|t|))$ (numbers of runs are written in binary).

In this example, one can consider that a state q of $\mathcal{A}$ at a position u is enriched with an *attribute*, say $a(q, u)$, that records information about all the runs of $\mathcal{A}$ on the subterm issued from u that yield state q at u. This attribute is here the number of such runs. We get a nondeterministic FA $\mathcal{A}'$ whose states are pairs (q, m) in $Q_{\mathcal{A}} \times \mathbb{N}_+$. (Its nondeterminism is limited: for each q and at each position u, the automaton $\mathcal{A}'$ reaches a unique state of the form (q, m).) The FA $\mathcal{B}$ is then obtained from $\det(\mathcal{A}')$.

3.2 Polynomial-time fly-automata

We now classify *deterministic* FA according to their computation times. The restriction to deterministic FA is not a real one because we will always consider determinized runs of FA, hence implicitly only deterministic FA.

Definition 6 : *Polynomial-time fly-automata and related notions*

A deterministic fly-automaton over a signature F, possibly with output, is a *polynomial-time fly-automaton* (a *P-FA*) if its computation time on any term $t \in T(F)$ is P-bounded (cf. Section 2.5). It is an *FPT-fly-automaton* (an *FPT-FA* in short) or an *XP-fly-automaton* (an *XP-FA* in short) if its computation time is, respectively, FPT-bounded or XP-bounded. It is a *linear FPT-FA* if the computation time is bounded by $f(Sig(t)).\|t\|$ (equivalently by $f'(Sig(t)).|t|$) for some fixed function f (or f'). These three notions coincide if F is finite. A deterministic FA $\mathcal{A}$ over F is an XP-FA if and only if $\mathcal{A} \restriction F'$ is a P-FA for each finite subsignature F' of F.

Lemma 7 : Let $\mathcal{A}$ be a FA over a signature F.

(1) Let $\mathcal{A}$ be deterministic. It is a P-FA, resp. an FPT-FA, resp. an XP-FA if and only if there are functions p_1, p_2, p_3 such that, in the run of $\mathcal{A}$ on any term $t \in T(F)$:

$p_1(\|t\|)$ bounds the time for firing a transition,

$p_2(\|t\|)$ bounds the size of a state,

$p_3(\|t\|)$ bounds the time for checking if a state is accepting or for computing the output[8],

and these functions are polynomials, resp. are FPT-bounded, resp. are XP-bounded.

(2) If $\mathcal{A}$ is not deterministic, the fly-automaton $\det(\mathcal{A})$ is a P-FA, resp. an FPT-FA, resp. an XP-FA if and only if there are functions $p_1, ..., p_4$ such that, in the determinized run of $\mathcal{A}$ on any term $t \in T(F)$:

$p_1(\|t\|)$ bounds the time for firing the next transition[9],

$p_2(\|t\|)$ and $p_3(\|t\|)$ are as in (1),

$p_4(\|t\|)$ bounds the nondeterminism degree of $\mathcal{A}$ on t,

and these functions are polynomials, resp. are FPT-bounded, resp. are XP-bounded.

Proof : We prove (2) that yields easily (1).

"Only if". If $\det(\mathcal{A})$ is a P-FA with bounding polynomial p, then, one can take $p_i = p$ for $i = 1, ..., 4$.

"If". Let us conversely assume that $\mathcal{A}$ has bounding polynomials $p_1, ..., p_4$. The states of $\det(\mathcal{A})$ on a term t of size n are sets of at most $p_4(n)$ sequences of length at most $p_2(n)$, that we organize as trees with at most $p_4(n)$ branches of length at most $p_2(n)$. Firing a transition of $\det(\mathcal{A})$ at an occurrence u in t of a binary symbol f with sons u_1 and u_2 uses the following operations:

[8] by using $Out_{\mathcal{A}}$; it bounds also the size of the output

[9] We recall from Definition 1 that the sets $\delta_{\mathcal{A}}(f, q_1, \ldots, q_m)$ are linearly ordered; firing the next transition includes recognizing that there is no next transition.

for all states q_1 at u_1 and q_2 at u_2, we compute in time bounded by $p_4(n)^2.p_1(n)$ the states of $\delta_{\mathcal{A}}(f, q_1, q_2)$ and we insert them in the already constructed tree intended to encode the state of $\det(\mathcal{A})$ at u. (In this way we eliminate duplicates). Each insertion takes time at most $p_2(n)$, hence the total time is bounded by $p_4(n)^2.(p_1(n)+p_2(n))$.

In time bounded by $p_3(n).p_4(n)$ we can check if the state at the root is accepting or compute the output. The case of symbols of different arity is similar. Hence, if r is the maximal arity of a symbol in F, we can take the polynomial

$$p(n) = n.(p_1(n) + p_2(n)).p_4(n)^r + p_3(n).p_4(n)$$

to bound the global computation time.

The proof is similar for the two other types of bound. □

Remarks 8 : (1) For every MS formula $\varphi(X_1, ..., X_s)$ expressing a graph property, one can construct an FPT-FA $\mathcal{A}_\infty$ over $F_\infty^{(s)}$ that recognizes the set of terms $t*(X_1, ..., X_s)$ such that $G(t) \models \varphi(X_1, ..., X_s)$. The functions p_1, p_2, p_3 of Lemma 7(1) depend only on the minimum k such that $t \in T(F_k^{(s)})$. The recognition time is thus $f(k).|t|$ and even $f'(k).\left|V_{G(t)}\right|$ if t is a good term (cf. the end of Section 2.2). The function $f(k)$ may be a polynomial or hyper-exponential function. (Concrete cases are discussed in [BCID], see Table 20.) For each k, $\mathcal{A}_\infty$ has a finite subautomaton $\mathcal{A}_k$ over $F_k^{(s)}$ that recognizes the set $\{t*(X_1, ..., X_s) \in T(F_k^{(s)}) \mid G(t) \models \varphi(X_1, ..., X_s)\}$. We have $\mathcal{A}_k \subseteq \mathcal{A}_{k'}$ if $k < k'$ ([BCID], Section 7.3.1).

(2) For finding if a deterministic FA is a P-FA, or an FPT-FA or an XP-FA, the main value to examine is the maximal size of a state bounded by p_2, because in most cases, computing the output or the state of a transition is doable in polynomial time (with a small constant exponent) in the size of the considered states. For a nondeterministic FA, we must also examine the degree of nondeterminism bounded by p_4.

(3) It is clear that the height of a term and the number of its positions are P-FA computable (see Examples 3). So are, for $t \in T(F_\infty)$, the sets of port labels $\pi(t)$ and $\mu(t)$ (by FAs whose states on t are finite subsets of $\mu(t)$) and the number of vertices of $G(t)$ (it equals the number of occurrences of nullary symbols **a**). The set of good terms is thus P-FA recognizable.

(4) The mapping $\mathrm{Sat}X.P(X)$ (defined in the Introduction) is not P-FA computable, and even not XP-FA computable in general for the obvious reason that its output is not always of polynomial size (take $P(X)$ always true)[10].

Proposition 9 : Let F be a signature. Every **P**-computable (resp. **FPT**-computable or **XP**-computable) function α on $T(F)$ is computable by a P-FA (resp. by an FPT-FA or an XP-FA).

[10] Unless $\mathrm{Sat}X.P(X)$ is encoded in a particular compact way; here we take it is a straight list of sets.

Proof: Consider the deterministic FA $\mathcal{A}$ over F with set of states $T(F)$ that associates with each position u of the input term t the state t/u, (i.e., the subterm of t issued from u). The state at the root is t itself, and is obtained in linear time. We take α as output function. Then $\mathcal{A}$ is a P-FA (resp. an FPT-FA or an XP-FA). $\square$

Hence, our three notions of FA may look uninteresting. Actually, we will be interested by giving effective constructions of P-FA, FPT-FA and XP-FA from logical expressions of functions and properties (possibly *not MS expressible* but are decidable in polynomial time on graphs of bounded tree-width or clique-width). Our interest is in the uniformity and flexibility of constructions.

When we say that a function is P-FA computable, we mean of course that it is computable by a P-FA but, also that we have constructed this automaton or that we know how to construct it by an algorithm. The same remark applies for FPT-FA and XP-FA computability. All our existence proofs are effective.

3.3 Transformations and compositions of automata

In view of building algorithms by combining previously constructed automata, we define and analyze several operations on automata.

Proposition 10 : Let $\mathcal{A}_1, ..., \mathcal{A}_r$ be P-FA that compute functions $\alpha_1, ..., \alpha_r : T(F) \to \mathcal{D}$. There exists a P-FA $\mathcal{A}$ that computes the function $\alpha : T(F) \to \mathcal{D}^r$ such that $\alpha(t) = (\alpha_1(t), ..., \alpha_r(t))$. If $\mathcal{A}_1, ..., \mathcal{A}_r$ are FPT-FA or XP-FA, then $\mathcal{A}$ is of same type.

Proof : The product automaton $\mathcal{A} = \mathcal{A}_1 \times_g ... \times_g \mathcal{A}_r$ where $g(q_1, ..., q_r) := (Out_{\mathcal{A}_1}(q_1), ..., Out_{\mathcal{A}_r}(q_r))$ is a FA (cf. Definition 4(2)) that computes α. The computation time of $\mathcal{A}$ on a term is the sum of the computation times of $\mathcal{A}_1, ..., \mathcal{A}_r$ on this term. The claimed results follow.$\square$

Next we consider operations defined in Definition 4 that transform single automata.

Proposition 11 : Let $\mathcal{A}$ be a P-FA that computes $\alpha : T(F) \to \mathcal{D}$.
(1) If g is a **P**-computable function $\mathcal{D} \to \mathcal{D}$, then, there is a P-FA over F that computes $g \circ \alpha$.
(2) Let $h : F' \to F$ be a relabeling. There exists a P-FA over F' that computes the mapping $\alpha \circ h : T(F') \to \mathcal{D}$.
The same implications hold for FPT-FA and XP-FA.

Proof : (1) The FA $g \circ \mathcal{A}$ defined from $\mathcal{A}$ (output composition) by replacing $Out_{\mathcal{A}}$ by $g \circ Out_{\mathcal{A}}$ computes $g \circ \alpha$. Recall that the size of an output is polynomially bounded.

(2) Immediate by the inverse image construction. Recall that h is computable in linear time (cf. Section 2.4).

Each class P-FA, FPT-FA and XP-FA is preserved in both cases. □

Proposition 12 : Let $h : F \to F'$ be a relabeling with a computable inverse. Let $\mathcal{A}$ be a P-FA (resp. an FPT-FA, resp. an XP-FA) that computes $\alpha : T(F) \to \mathcal{D}$. The fly-automaton $\det(h(\mathcal{A}))$ over F' is a P-FA (resp. an FPT-FA, resp. an XP-FA) if and only if the nondeterminism degree of $h(\mathcal{A})$ is P-bounded (resp. FPT-bounded, resp. XP-bounded) in the size of terms over F'.

Proof : Let $\mathcal{A}$ be a P-FA. That $h(\mathcal{A})$ is a FA is proved in Proposition 45 of [BCID]. Then $\det(h(\mathcal{A}))$ is a P-FA by Lemma 7(2) and the definitions. The same lemma yields the cases of FPT-FA and XP-FA. □

In the sufficient conditions, the bounds on $ndeg_{h(\mathcal{A})}(t)$ can be replaced by bounds on $\left|Q_A \restriction h^{-1}(t)\right|$ (the number of states of $\mathcal{A}$ used on an input term t' such that $h(t') = t$, see Definition 1(b)) that are frequently easier to check.

Proposition 9 and the following one show that **P≠NP** implies that there is no alternative image construction for FA that preserves the polynomial-time property. (A related counter-example not depending on the hypothesis **P≠NP** will be given below, see Counter-example 18).

Proposition 13 : Assume that **P≠NP.** There exists finite signatures F, F', a relabeling $h : F' \to F$ and a set $L \subseteq T(F')$ that is decidable in polynomial time whereas $h(L)$ is not. There is a P-FA property $P(X)$ on terms in $T(F)$ such that $\exists X.P(X)$ is not P-FA decidable.

Proof: In order to use the NP-complete satisfiability problem (SAT), we let X be the infinite set of propositional variables $x_0, ..., x_n, ..$ and $T = T(\{\vee, \wedge, \neg\} \cup X)$. We introduce a unary function symbol f and a nullary one a, in order to encode x_i by $f^i(a)$. A term t in T is thus encoded by a term $\widetilde{t}$ in $T(F)$ where $F = \{\vee, \wedge, \neg, f, a\}$. The set $\widetilde{T}$ of terms $\widetilde{t}$ is a regular subset of $T(F)$.

To each occurrence u of a variable x_i in $t \in T$ corresponds an occurrence $\widetilde{u}$ of a in $\widetilde{t}$. Consider a term t in T given with an assignment $\nu : X \to \{True, False\}$. We denote by $\nu(t)$ the corresponding Boolean value of t. We let b be a new nullary symbol, $F' = \{\vee, \wedge, \neg, f, a, b\}$ and $h : F' \to F$ that maps b to a and is the identity everywhere else.

We let $\widetilde{t_\nu} \in T(F')$ be obtained from $\widetilde{t}$ by replacing a by b at each occurrence $\widetilde{u}$ such that u is an occurrence of a variable x_i whose value under ν is $False$. Hence, $\widetilde{t_\nu}$ encodes t and ν. We let M be the set of terms of the form $\widetilde{t_\nu}$. This set is decidable in polynomial time, but it is not regular. Finally, we let L be the set of terms $\widetilde{t_\nu}$ such that $\nu(t) = True$. This set is also **P**-decidable. Then, a satisfiability problem given by a term t has a solution if and only if $\widetilde{t} \in h(L)$. Hence, unless **P=NP**, the set $h(L)$ is not **P**-decidable.

For proving the second assertion, we let $P(X)$ be the property of a term s over F and a set $X \subseteq Pos(s)$ that :

$s = \widetilde{t}$ for some t in T,

$X \subseteq Pos_a(s)$ ($Pos_a(s)$ is the set of occurrences of a),

if s' is obtained from s by replacing by a by b at each of its occurrences in X, then $s' \in L$.

Then, $P(X)$ is **P**-decidable (for given set X) hence, by Proposition 9, P-FA decidable. Since $\exists X.P(X)$ is not **P**-decidable, it is not P-FA decidable, again by Proposition 9. $\square$

Two basic P-FA

Examples 14 : *Cardinality and identity.*
(a) We consider the function $Card$ that associates with a set X of positions of a term in some set $T(F)$ its cardinality $|X|$. Hence, $\overline{Card} : T(F^{(1)}) \to \mathbb{N}$. It is computed by a FA whose states are the natural numbers. On a term with n positions, the maximum value of a state is n, hence has size $\log(n)$ (numbers are written in binary notation). The output function is the identity. The transitions are obvious to describe. The resulting automaton is a P-FA, denoted by $\mathcal{A}_{Card(X)}$.

The computation time is $O(n.\log(n))$. (It is $O(n)$ if we admit that the addition of two numbers at most n can be done in constant time). (We will use it in Sections 5.2.4 and 5.2.6).

From $\mathcal{A}_{Card(X)}$ one can construct, for each integer p, a P-FA $\mathcal{A}_{Card(X)\leq p}$ to check that X has at most p elements. However, the automata $\mathcal{A}_{Card(X)\leq p}$ can be handled as instanciations of a unique P-FA that takes as input a term t, a set of positions X of this term and, as auxiliary input, an integer p. This idea is used in a natural way in the system AUTOGRAPH (see Section 6) but we do not modify the formal definitions so as to allows FA to depend on auxiliary parameters like integers.

(b) Next, we consider the function Id that associates with a set X of positions of a term $t \in T(F)$ this set itself. Positions in terms are defined as Dewey words (cf. Section 2.1). The construction of a FA denoted by $\mathcal{A}_{Id(X)}$ for the function Id is straightforward (cf. Example 3(b)). Its states are sets of positions of the input term, hence have size $O(\| t \|^2)$ (cf. Section 2.1). The automaton $\mathcal{A}_{Id(X)}$ is a P-FA. It may look trivial, but it will useful for the proof of Corollary 19 or when combined with others, by mean of Proposition 15 (see Section 5.1.1 for an example).

4 Fly-automata for logically defined functions and properties

We now examine if and when the transformations of automata representing certain logical constructions preserve the classes P-FA, FPT-FA and XP-FA. From Proposition 13, we know that this is not the case for existential set quantifications. We also examine in the same perspective the logic based functions defined in the Introduction.

Two functions (or properties) α and β, possibly with set arguments, are of *same type* if the domains of $\overline{\alpha}$ and $\overline{\beta}$ are the same, that is, are both $T(F)$ or both $T(F^{(s)})$ for some F and s.

Proposition 15 : (1) If $\alpha_1, ..., \alpha_r$ are P-FA computable functions of same type and g is a **P**-computable function (or relation) of appropriate type, the function (or the property) $g \circ (\alpha_1, ..., \alpha_r)$ is P-FA computable (or decidable).

(2) If α_1, α_2 and P are P-FA computable functions of same type and P is Boolean-valued, then the function $\texttt{if } P \texttt{ then } \alpha_1 \texttt{ else } \alpha_2$ is P-FA computable.

(3) If P and Q are P-FA decidable properties of same type, then, so are $\neg P$, $P \vee Q$ and $P \wedge Q$.

(4) The same three properties hold with FPT-FA and XP-FA.

Proof : Straightforward consequences of Propositions 10 and 11(1). For $P \vee Q$ and $P \wedge Q$ in Case (3), we use the product $\mathcal{A}_P \times_g \mathcal{A}_Q$ with two different functions g. For $P_1 \wedge ... \wedge P_r$, we denote the corresponding automaton by $\mathcal{A}_P \times_\wedge ... \times_\wedge \mathcal{A}_{P_r}$ and similarly for $P_1 \vee .. \vee P_r$. □

We denote by $\alpha \restriction P \wedge ... \wedge Q$ the function $\texttt{if } P \wedge ... \wedge Q \texttt{ then } \alpha \texttt{ else } \bot$: it is the restriction of α to its arguments that satisfy $P \wedge ... \wedge Q$ (and could be written $(...(\alpha \restriction P) \restriction ...) \restriction Q$).

We now consider substitutions of set terms and variables (cf. Section 2.3).

Proposition 16 : Let $\alpha(Y_1, ..., Y_m)$ denote a P-FA function on terms in $T(F)$ with set arguments $Y_1, ..., Y_m$. Let $S_1, ..., S_m$ be set terms over $\{X_1, ..., X_n\}$. The function $\beta(X_1, ..., X_n) := \alpha(S_1, ..., S_m)$ is P-FA computable. The same holds with FPT-FA and XP-FA.

Proof: We recall from Section 2.3 that $\overline{\beta} = \overline{\alpha} \circ h$ where h is a relabelling : $T(F^{(n)}) \to T(F^{(m)})$ that modifies only the Boolean part of each symbol. If $\mathcal{A}$ is a P-FA that computes $\overline{\alpha}$, then $\mathcal{B} := h^{-1}(\mathcal{A})$ is a P-FA by Proposition 11(2) that computes $\overline{\beta}$. The same proof works for FPT-FA and XP-FA. □

In Proposition 15, we combine functions and properties of same type. With the previous proposition we can extend this proposition to properties and functions that are not of same type. For example if we need $P(X_1, X_2) \wedge Q(X_1, X_2, X_3)$,

we redefine $P(X_1, X_2)$ into $P'(X_1, X_2, X_3)$ that is true if and only if $P(X_1, X_2)$ is, independently of X_3. Proposition 16 shows how to transform an automaton for $P(X_1, X_2)$ into one for $P'(X_1, X_2, X_3)$. Then $P(X_1, X_2) \wedge Q(X_1, X_2, X_3)$ is equivalent to $P'(X_1, X_2, X_3) \wedge Q(X_1, X_2, X_3)$ and we can apply Proposition 15.

4.1 First-order constructions

We now consider the more delicate case of existential quantifications. We define one more construction[11] : if $\alpha(\overline{X})$ is a function (relative to a term t), we define $\mathrm{SetVal}\overline{X}.\alpha(\overline{X})$ as the set of values $\alpha(\overline{X}) \neq \perp$ for all relevant tuples $\overline{X}$. We recall that $\exists x_1, ..., x_s.P(x_1, ..., x_s)$ (also written $\exists \overline{x}.P(\overline{x})$) abbreviates

$$\exists X_1, ..., X_s.(P(X_1, ..., X_s) \wedge Sgl(X_1) \wedge ... \wedge Sgl(X_s))$$

and similarly, that $\mathrm{SetVal}(x_1, ..., x_s).\alpha(x_1, ..., x_s)$ (also written $\mathrm{SetVal}\overline{x}.\alpha(\overline{x})$) stands for

$$\mathrm{SetVal}(X_1, ..., X_s).(\alpha(X_1, ..., X_s) \restriction Sgl(X_1) \wedge ... \wedge Sgl(X_s)).$$

Theorem 17 : (1) If $P(\overline{X})$ is a P-FA decidable property, then the properties $\exists \overline{x}.P(\overline{x})$ and $\forall \overline{x}.P(\overline{x})$ are P-FA decidable.

(2) If $\alpha(\overline{X})$ is a P-FA computable function, then the function $\mathrm{SetVal}\overline{x}.\alpha(\overline{x})$ is P-FA computable.

(3) The same implications hold for the classes FPT-FA and XP-FA.

Proof : (1) and (3). We let $\mathcal{A}$ be a deterministic FA over $F^{(s)}$ that decides $P(\overline{X})$.We let $\mathcal{B}_i$ be the deterministic FA over $F^{(s)}$ for $Sgl(X_i)$ with states 0,1 and $Error_{\mathcal{B}_i}$ such that :

$$run_{\mathcal{B}_i, t*\overline{X}}(u) = 0 \text{ if } X_i/u = \emptyset,$$
$$run_{\mathcal{B}_i, t*\overline{X}}(u) = 1 \text{ if } |X_i/u| = 1 \text{ and}$$
$$run_{\mathcal{B}_i, t*\overline{X}}(u) = Error_{\mathcal{B}_i} \text{ if } |X_i/u| \geq 2.$$

The deterministic FA $\mathcal{A} \times_g \mathcal{B}_1 \times_g ... \times_g \mathcal{B}_s$ decides property $P(X_1, ..., X_s) \wedge Sgl(X_1) \wedge ... \wedge Sgl(X_s)$: its set of states is $Q_{\mathcal{A}} \times Q_{\mathcal{B}_1} \times ... \times Q_{\mathcal{B}_s}$ and g is defined so that the set of accepting states is $Acc_{\mathcal{A}}^{-1}(True) \times \{1\} \times ... \times \{1\}$. We build a smaller deterministic FA $\mathcal{C}$ with set of states $\{Error_{\mathcal{C}}\} \cup ((Q_{\mathcal{A}} - \{Error_{\mathcal{A}}\}) \times \{0, 1\}^s)$ by merging into a unique error state $Error_{\mathcal{C}}$ all tuples of $Q_{\mathcal{A}} \times Q_{\mathcal{B}_1} \times ... \times Q_{\mathcal{B}_s}$, one component of which is an error state and same set of accepting states.

The nondeterministic automaton $pr_s(\mathcal{C})$ decides the property $\exists \overline{x}.P(\overline{x})$. Its states at a position u in a term $t \in T(F)$ are $Error_{\mathcal{C}}$ and the tuples of the form

[11] The notation $\exists X_1, ..., X_s.P(X_1, ..., X_s)$ abbreviates $\exists X_1 \exists X_2 ... \exists X_s.P(X_1, ..., X_s)$ but $\mathrm{SetVal}(X_1, ..., X_s).\alpha(X_1, ..., X_s)$ is not expressible as an iterated application of the construction $\mathrm{SetVal}X_i.\beta$.

$(run_{\mathcal{A},t*(X_1,...,X_s)}(u), |X_1/u|, ..., |X_s/u|)$
such that $|X_1/u|, ..., |X_s/u| \leq 1$.

Since $\mathcal{A}$ is deterministic, there are at most $1 + (|t| + 1)^s$ different such states and the nondeterminism degree of $pr_s(\mathcal{C})$ is bounded by the polynomial $p(n) = 1 + (n + 1)^s$ that does not depend on $Sig(t)$. Hence $\det(pr_s(\mathcal{C}))$ is a P-FA, an FPT-FA or an XP-FA by Lemma 7 if $\mathcal{A}$ is so.

Property $\forall \overline{x}.P(\overline{x})$ can be written $\neg\exists\overline{x}.\neg P(\overline{x})$. The results follow since, by by Proposition 15(3,4) the classes of P-FA, FPT-FA and XP-FA that check properties are closed under the transformation for negation.

(2) and (3). We now apply the same construction to a FA $\mathcal{A}$ over $F^{(s)}$ that computes $\alpha(\overline{X})$. As output function for $\mathcal{C}$, we take:

$Out_{\mathcal{C}}((q, 1, ..., 1)) = Out_{\mathcal{A}}(q)$, for $q \in Q_{\mathcal{A}}$,
$Out_{\mathcal{C}}(p) = \bot$, for all other states p of $\mathcal{C}$.

By the definitions, $Comp(\det(pr_s(\mathcal{C})))$ is equal to

$$\mathrm{SetVal}(X_1, ..., X_s).(\alpha(X_1, ..., X_s) \restriction Sgl(X_1) \wedge ... \wedge Sgl(X_s)) = \mathrm{SetVal}\overline{x}.\alpha(\overline{x})$$

hence, is P-FA, or FPT-FA or XP-FA computable by Lemma 7, depending on $\mathcal{A}$ as above. □

The construction of this proof is *generic* in that it applies to *any* deterministic FA $\mathcal{A}$ over $F^{(s)}$, even that is not of type XP. The hypotheses on the type, P, FPT or XP, of $\mathcal{A}$ are only used to determine the type of the resulting automaton.

In Theorem 17, we only handle first-order quantifications. Proposition 13 has shown that we cannot replace them by arbitrary set quantifications. We now give a counter-example that does not use any complexity hypothesis.

Counter-example 18 : We consider terms over $F = \{f, g, a\}$ where f is binary, g is unary and a is nullary. For every position u of $t \in T(F)$, we let $s(u) = |Pos(t)/u|$. For a set X of positions of t, we define $m(X)$ as $\{\!\{s(u) \mid u \in X\}\!\}$, the multiset of numbers $s(u)$ for $u \in X$.

We let $P(X)$ mean that X is not empty, its elements are first sons of occurrences of f, and the multiset $m(X)$ contains exactly two copies of each of its elements.

There is a deterministic FA $\mathcal{A}$ over $F^{(1)}$ that decides $P(X)$. The state $run_{\mathcal{A},t*X}(u)$ is $Error$ if X/u contains a position different from u that is not the first son of an occurrence of f or if $m(X/u)$ contains at least 3 copies of some integer. Otherwise, $run_{\mathcal{A},t*X}(u) = (\varepsilon, m(X/u))$ with $\varepsilon = 0$ if $u \notin X$ and $\varepsilon = 1$ if $u \in X$. States can thus be encoded by words of length $O(|t|.\log(|t|))$ and $\mathcal{A}$

is a P-FA. The accepting states are $(0, m)$ where m is not empty and contains exactly two copies of each of its elements[12].

Consider now the nondeterministic FA $pr_1(\mathcal{A})$ where pr_1: $F^{(1)} \to F$. It decides the property $\exists X.P(X)$. The second components of any state belonging to $run^*_{pr(\mathcal{A}),t}(u)$ are the multisets $m(X/u)$ that do not contain 3 copies of a same integer and that are associated with a set X containing only first sons of occurrences of f and . The maximum cardinality of the set $run^*_{pr(\mathcal{A}),t}(u)$ is the nondeterminism degree of $pr(\mathcal{A})$ on t. We now prove that it is not polynomially bounded in $|t|$, hence in the size of t.

For each p, we let $t = f(a, f(ga, f(gga, f(ggga, ..., f(g^p a, a))...)))$ with $p+1$ occurrences of f. Hence, $|t| = 3 + 2p + p(p+1)/2$. The set $run^*_{pr(\mathcal{A}),t}(root_t)$ contains the pairs $(0, m)$ for all subsets m of $[p+1]$. Hence, the nondeterminism degree of $pr(\mathcal{A})$ is not polynomially bounded in the size of the input term, and $pr(\mathcal{A})$ is not a P-FA.

For a comparison with Proposition 13, note that we can easily replace $\mathcal{A}$ by a P-FA that decides $\exists X.P(X)$ without using $pr(\mathcal{A})$ as an intermediate step. This counter-example only proves that the image construction for FA that corresponds to existential set quantifications does not preserve the polynomial-time property. □

We define $\mathrm{Sat}(x_1, ..., x_s).P(x_1, ..., x_s)$ as

$$\{(u_1, ..., u_s) \mid P(\{u_1\}, ..., \{u_s\}) = True\}.$$

This set is in bijection with

$$\mathrm{Sat}(X_1, ..., X_s).(P(X_1, ..., X_s) \wedge Sgl(X_1) \wedge ... \wedge Sgl(X_s))$$

that is equal to

$$\{(\{u_1\}, ..., \{u_s\}) \mid (u_1, ..., u_s) \in \mathrm{Sat}(x_1, ..., x_s).P(x_1, ..., x_s)\}.$$

We recall from the introduction that $\mathrm{Sat}\overline{x}.P(\overline{x})(t)$ is the set of assignments $\overline{x}$ such that $t \models P(\overline{x})$ and that $\#\overline{x}.P(\overline{x})(t)$ is the number of such assignments. Furthermore, $\mathrm{SetVal}\overline{x}.\alpha(\overline{x})(t)$ is the set of values of a function α for all tuples $\overline{x}$ in t such that this value is not $\perp$ (standing, typically, for an undefined value).

[12] It is easy to see that this automaton is actually minimal, in the sense that $run_{\mathcal{A},t*X}(u) = run_{\mathcal{A},t*X}(u')$ if and only if, for every context c in $Ctxt(F^{(1)})$, $c[t/u]$ is accepted if and only if $c[t/u']$ is accepted (see [BCID] for contexts). Intuitively, the information conveyed by $\mathcal{A}$ is not redundant. It is exactly what is needed.

Corollary 19 : If $P(\overline{X})$ is a P-FA decidable property, then the functions Sat$\overline{x}.P(\overline{x})$ and $\#\overline{x}.P(\overline{x})$ are P-FA computable. The same implication holds with FPT-FA and XP-FA.

Proof : We observe that Sat$\overline{x}.P(\overline{x})$ = SetVal$\overline{x}.\alpha(\overline{x})$ where:

$\alpha(\overline{x}) :=$ `if` $P(\overline{x})$ `then` $\overline{x}$ `else` $\bot$.

The result follows then from Propositions 15(2), Theorem 17 and a variant of $\mathcal{A}_{Id(X)}$ of Example 14(b). However, we can give a direct construction. We modify as follows the one done for proving Theorem 17. We replace each $\mathcal{B}_i$ by $\mathcal{B}'_i$ such that :

$run_{\mathcal{B}'_i,t*\overline{X}}(u) = \emptyset$ if $X_i/u = \emptyset$,

$run_{\mathcal{B}'_i,t*\overline{X}}(u) = \{w\}$ if $X_i/u = \{u.w\}$ (recall that positions are Dewey words) and

$run_{\mathcal{B}'_i,t*\overline{X}}(u) = Error_{\mathcal{B}'_i}$ if $|X_i/u| \geq 2$.

Then, we make the product $\mathcal{A} \times \mathcal{B}'_1 \times ... \times \mathcal{B}'_s$ into a deterministic automaton $\mathcal{C}'$ with set of states $\{Error_{\mathcal{C}'}\} \cup ((Q_{\mathcal{A}} - \{Error_{\mathcal{A}}\}) \times \mathcal{P}_{\leq 1}([\rho(F)]^*)^s)$ similarly as in the proof of Theorem 17. The deterministic automaton $\mathcal{C}$", defined as $\det(pr(\mathcal{C}'))$ equipped with the output function such that for $Z \subseteq Q_{\mathcal{C}'} = Q_{pr(\mathcal{C}')}$:

$$Out_{\mathcal{C}"}(Z) = \{(x_1, ..., x_s) \mid (q, \{x_1\}, ..., \{x_s\}) \in Z, Acc_{\mathcal{A}}(q) = True\}, (1)$$

defines Sat$\overline{x}.P(\overline{x})$. The states of $pr(\mathcal{C}')$ at a position u of t are $Error_{\mathcal{C}'}$ and tuples $(run_{\mathcal{A},t*(X_1,...,X_s)}(u), X_1, ..., X_s)$ such that $|X_1|, ..., |X_s| \leq 1$ and $X_1 \cup ... \cup X_s \subseteq [r]^*$. Since $\mathcal{A}$ is deterministic, there are at most $1+(|t|+1)^s$ different such states at each position u. The nondeterminism degree of $\det(pr(\mathcal{C}'))$ is bounded as in the proof of Theorem 17. The conclusions follow from Lemma 7.

Since the value $\#\overline{x}.P(\overline{x})$ on a term t is computable in linear time from that of Sat$\overline{x}.P(\overline{x})$, we get the corresponding assertions (by using Proposition 11(1)). However, there is a more direct construction, that does not use Sat$\overline{x}.P(\overline{x})$ as an intermediate step. It consists in counting the number of accepting runs of $pr(\mathcal{C}')$, which we did in Example 5. □

Remarks : (1) From SetVal$\overline{x}.\alpha(\overline{x})$, we can obtain in polynomial time the maximum or the minimum value of $\alpha(\{x_1\}, ..., \{x_s\})$ if the range of α is linearly ordered and two values can be compared in polynomial time. The corresponding functions are thus P-FA (or FPT-FA or XP-FA) computable.

(2) The results of Theorem 17 and Corollary 19 remain valid if each condition $Sgl(X_i)$ is replaced by $Card(X_i) = c_i$ or $Card(X_i) \leq c_i$ for fixed integers c_i (see Section 5.2.3 for an example). In particular, we can compute (letting $\overline{X} = (X_1, ..., X_s)$) :

$$\#\overline{X}.(P(\overline{X}) \wedge Card(X_1) \leq c_1 \wedge ... \wedge Card(X_s) \leq c_s).$$

The exponents in the bounding polynomial become larger, but they still depend only on the numbers $c_1, ..., c_s$. (The polynomial $p(n) = 1 + (n+1)^s$ in the proof of Theorem 17(1) is replaced by $1 + (n+1)^{c_1+...+c_s}$. By Counter-example 18, this fact does not hold with $Card(X_i) \geq c_i$: just take $c_i = 0$.

Furthermore, by Example 14(a), there is a generic construction that takes as input an arbitrary FA over $F^{(s)}$ (without output), that is $\mathcal{A}_{P(\overline{X})}$ for some property $P(\overline{X})$ and produces a FA that checks

$$P(\overline{X}) \wedge Card(X_1) \leq c_1 \wedge ... \wedge Card(X_s) \leq c_s$$

by taking as input a term t, a tuple $X_1, ..., X_s$ of positions of t and a sequence of numbers $c_1, ..., c_s$. It yields a construction that takes as input $\mathcal{A}$ as above and a sequence $c_1, ..., c_s$, and produces a FA that computes

$$\#\overline{X}.(P(\overline{X}) \wedge Card(X_1) \leq c_1 \wedge ... \wedge Card(X_s) \leq c_s)(t).$$

4.2 Monadic Second-order constructions

Although Theorem 17 does not extend to arbitrary existential set quantifications, we can get some results for them and more generally, for the computation of multispectra and all derived functions, e.g., $\#\overline{X}.P(\overline{X})$, $\mathrm{Sp}\overline{X}.P(\overline{X})$, $\mathrm{MinCard}X.P(X)$ (defined in the introduction) and yet others.

4.2.1 Inductive computation of sets of satisfying tuples

Let F be an effectively given signature, $\overline{X} = (X_1, ..., X_s)$ and $P(\overline{X})$ be a property of terms in $T(F)$ with s set arguments. Let $\mathcal{A}$ be a deterministic FA over $F^{(s)}$ that recognizes $T_{P(\overline{X})}$ (cf. Section 2.3). Our aim is to check (efficiently) the property $\exists\overline{X}.P(\overline{X})$ and to compute the functions $\#\overline{X}.P(\overline{X})$, $\mathrm{Sp}\overline{X}.P(\overline{X})$, $\mathrm{MSp}\overline{X}.P(\overline{X})$, $\mathrm{MinCard}X.P(X)$, etc. by FA constructed from $\mathcal{A}$ in uniform ways. All these properties and functions can be computed from $\mathrm{Sat}\overline{X}.P(\overline{X})$. Although this set cannot in general be computed efficiently, we first consider its computation by a FA. From this automaton, we will derive FA for the other functions of interest. Every FA over $F^{(s)}$ without output decides a property $P(\overline{X})$ of terms in $T(F)$ with s set arguments. Our constructions apply thus to "arbitrary" FA, but they are more understandable if we think of the corresponding property of terms.

Definitions 20 : We let $F, s, P(\overline{X}), \mathcal{A}$ be as above and $r = \rho(F)$. We insist that $\mathcal{A}$ is deterministic.

(a) Let $t \in T(F^{(s)})$. For each state q in $Q_{\mathcal{A}}$ and position u of t, we let $Z(q,u)$ be the set of s-tuples $\overline{X} \in \mathcal{P}(Pos(t/u))^s$ such that $q_{\mathcal{A}}((t/u) * \overline{X}) = q$. This set is finite. Since $\mathcal{A}$ is deterministic, we have:

$$Z(q,u) \cap Z(q',u) = \emptyset \text{ if } q \neq q' \tag{2}$$

and clearly,

$$\mathrm{Sat}\overline{X}.P(\overline{X})(t) = \uplus\{Z(q, root_t) \mid q \text{ is accepting}\} \tag{3}$$

(we recall that $\uplus$ denotes the union of pairwise disjoint sets).

(b) If E and E' are disjoint sets, $Z \subseteq \mathcal{P}(E)^s$ and $Z' \subseteq \mathcal{P}(E')^s$ we define:

$$Z \circledast Z' := \{(X_1 \cup Y_1, \ldots, X_s \cup Y_s) \mid (X_1, \ldots, X_s) \in Z, \\ (Y_1, \ldots, Y_s) \in Z'\}.$$

This operation is associative in the following sense: if E, E' and E" are pairwise disjoint, $Z \subseteq \mathcal{P}(E)^s$, $Z' \subseteq \mathcal{P}(E')^s$ and Z" $\subseteq \mathcal{P}(E")^s$ then $Z \circledast (Z' \circledast Z") = (Z \circledast Z') \circledast Z"$.

(c) If $u \in E$ and $w \in \{0,1\}^s$, we define :

$$u^w := (U_1, ..., U_s) \text{ where } U_i := \{u\} \text{ if } w[i] = 1 \text{ and } U_i := \emptyset \\ \text{if } w[i] = 0.$$

(d) If $Z \in \mathcal{P}([r]^*)^s$ and $i \in [r]$, we let $i.Z$ be obtained by replacing in Z each word $u \in [r]^*$ by $i.u$. This notion will be used for tuples of sets of Dewey words. Let $t = f(t_1, t_2)$ and Z be an s-tuple of sets of positions of t_1, formally defined as an s-tuple of sets of Dewey words. Considered as an s-tuple of sets of positions of t, it is then formally defined as $1.Z$.

Definition 21: *A deterministic FA over F that computes* $\mathrm{Sat}\overline{X}.P(\overline{X})$.
We let $\mathcal{A} := \mathcal{A}_{P(\overline{X})}$ be as in Definitions 20. The sets $Z(q,u)$ can be computed bottom-up as follows.
If u is an occurrence of a nullary symbol a, then:

$$Z(q,u) = \{\varepsilon^w \mid w \in \{0,1\}^s, (a,w) \rightarrow_{\mathcal{A}} q\}. \tag{4}$$

If u is an occurrence of a binary symbol f with sons u_1 and u_2 :

$$Z(q,u) = \uplus\{\{\varepsilon^w\} \circledast 1.Z(q_1,u_1) \circledast 2.Z(q_2,u_2) \mid w \in \{0,1\}^s, \\ \text{and } (f,w)[q_1,q_2] \rightarrow_{\mathcal{A}} q\}. \tag{5}$$

The inclusion $\supseteq$ is clear. We prove the other one and the disjointness of union: consider a tuple in $Z(q,u)$; it can be expressed in a unique way as $\{\varepsilon^w\} \circledast 1.\{(X_1, \ldots, X_s)\} \circledast 2.\{(Y_1, \ldots, Y_s)\}$ for some $w \in \{0,1\}^s$ and sets $X_1, \ldots, Y_s$ such that $X_1 \cup \ldots \cup X_s \subseteq Pos(t/u_1)$ and $Y_1 \cup \ldots \cup Y_s \subseteq Pos(t/u_2)$. Since $\mathcal{A}$ is deterministic, $(X_1, \ldots, X_s) \in Z(q_1,u_1)$ and $(Y_1, \ldots, Y_s) \in Z(q_2,u_2)$ for

unique states q_1, q_2 and we have $(f,w)[q_1,q_2] \rightarrow_{\mathcal{A}} q$. Hence, the considered tuple belongs to a unique component in the right handside of (5).

There are similar equalities for the operations of other arities.

For terms defining graphs, the Boolean sequences w are only attached to the nullary symbols **a** that denote vertices. Hence, in (5) the term $\{\varepsilon^w\}$ is absent and we have $f[q_1,q_2] \rightarrow_{\mathcal{A}} q$ instead of $(f,w)[q_1,q_2] \rightarrow_{\mathcal{A}} q$. In the special case of (4) where u is an occurrence of the nullary symbol $\varnothing$ that denotes the empty graph we have :

$$Z(q,u) = \texttt{if } \varnothing \rightarrow_{\mathcal{A}} q \texttt{ then } \{(\emptyset,...,\emptyset)\} \texttt{ else } \emptyset. \qquad (4')$$

We obtain a deterministic FA $\mathcal{A}^{\mathrm{Sat}}$ over F that computes $\mathrm{Sat}\overline{X}.P(\overline{X})$. Its state at position u in a term $t \in T(F)$ is the function σ with domain $Q_{\mathcal{A}}$ that maps each state q to $Z(q,u)$. Each set $Z(q,u)$ is finite and can be encoded by a word over a fixed finite alphabet. Each such function σ can be seen and, more precisely implemented, as the set $\{(q,\sigma(q)) \mid \sigma(q) \neq \emptyset\}$ that is finite by (2). Hence, the set of states of $\mathcal{A}^{\mathrm{Sat}}$ is effectively given. To be precise, we define $Q_{\mathcal{A}^{\mathrm{Sat}}}$ as the set of mappings $\sigma : Q_{\mathcal{A}} \rightarrow \mathcal{P}_f(\mathcal{P}_f([r]^*)^s)$ such that $\sigma(q) \cap \sigma(q') = \emptyset$ if $q \neq q'$ and $\sigma(q) \neq \emptyset$ for finitely many states q only. (Not all these states are accessible.) Equalities (4) and (5) define its transitions. By equality (3), the output function can be taken as :

$$Out_{\mathcal{A}^{\mathrm{Sat}}}(\sigma) := \uplus\{\sigma(q) \mid q \text{ is accepting}\}. \qquad (6)$$

The notation $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ for $\mathcal{A}^{\mathrm{Sat}}$ will be used to recall the considered property $P(\overline{X})$. The transformation of $\mathcal{A}$ into $\mathcal{A}^{\mathrm{Sat}}$ is the same for all deterministic FA over $F^{(s)}$. To summarize, we have:

Proposition 22 : $Comp(\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}) = \mathrm{Sat}\overline{X}.P(\overline{X})$.

The construction of $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ is similar to that of Example 5. Given a deterministic FA $\mathcal{A}$ over $F^{(s)}$ that checks $P(\overline{X})$, we consider the nondeterministic FA $pr_s(\mathcal{A})$ over F, and we let $\mathcal{B} := \det(pr_s(\mathcal{A}))$. We equip each q_i in a state $\{q_1,...,q_p\}$ of $\mathcal{B}$ at position u with an *attribute*. Here the attribute is the set $Z(q_i,u)$ that records the different ways q_i can be reached at u in a run of $pr_s(\mathcal{A})$. The states of $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ are sets of the form $\{(q,Z(q,u)) \mid Z(q,u) \neq \emptyset, q \in Q_{\mathcal{A}}\}$. (We have $Z(q,u) = \emptyset$ if q is not reachable at u for any choice of $\overline{X}$).

The automaton $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ is of little practical use because its states are in general "too large" (but this was not the case in the proof of Corollary 19). However, if we wish to compute some value $\gamma(\mathrm{Sat}\overline{X}.P(\overline{X})(t))$ for $t \in T(F)$ rather than $\mathrm{Sat}\overline{X}.P(\overline{X})(t)$ itself, and if the mapping γ has some good "homomorphic" properties, then we may be able to replace $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ by a "smaller"

automaton that computes more efficiently $\gamma(\mathrm{Sat}\overline{X}.P(\overline{X})(t))$. To introduce the general construction, we show a concrete example.

Example 23: *The number of satisfying assignments.*

We define an automaton for computing $\#\overline{X}.P(\overline{X})$, the cardinality of $\mathrm{Sat}\overline{X}.P(\overline{X})$. We replace $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ by a simpler automaton $\mathcal{B}$ by using the following observations. For every term $t \in T(F)$ we have, using (3):

$$\#\overline{X}.P(\overline{X})(t) = |\mathrm{Sat}\overline{X}.P(\overline{X})(t)| = \Sigma\{|Z(q, root_t)| \mid q \text{ is accepting}\}, \quad (7)$$

($\Sigma\{... \mid ...\}$ denotes a summation over a multiset) because (clearly):

$$|Z \uplus Z'| = |Z| + |Z'|. \quad (8)$$

If u is an occurrence of a nullary symbol a, we have:

$$|Z(q,u)| = |\{w \mid w \in \{0,1\}^s, (a,w) \rightarrow_{\mathcal{A}} q\}|. \quad (9)$$

If u is an occurrence of a binary symbol f with sons u_1 and u_2, we have:

$$|Z(q,u)| = \Sigma\{|Z(q_1,u_1)|.|Z(q_2,u_2)| \mid w \in \{0,1\}^s, \text{ and } (f,w)[q_1,q_2] \rightarrow_{\mathcal{A}} q\}, \quad (10)$$

because (clearly) :

$$|Z \circledast Z'| = |Z|.|Z'|. \quad (11)$$

Hence we transform $\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ into $\mathcal{B}$ by replacing each component of a state of the form $Z(q,u)$ by its cardinality. We obtain a deterministic automaton whose states are finite mappings ν: $Q_{\mathcal{A}} \rightarrow \mathbb{N}$ equivalently, the finite sets $\{(q, \nu(q)) \mid \nu(q) \neq 0\}$. Its transitions on symbols of positive arity are defined by Equality (10) because of the "homomorphic properties" (8) and (11). Its output function is defined by:

$$Out_{\mathcal{B}}(\nu) := \Sigma\{\nu(q) \mid q \text{ is accepting}\}. \quad (12)$$

Clearly, $Comp(\mathcal{B}) = \#\overline{X}.P(\overline{X})$. $\square$

This construction is that of Example 5 applied to the nondeterministic automaton $pr(\mathcal{A}_{P(\overline{X})})$. For generalizing it, we introduce homomorphisms of automata.

Definitions 24: *Homomorphisms of deterministic fly-automata.*

Let $\mathcal{A}$ and $\mathcal{B}$ be two deterministic FA over F with outputs taking values in sets $\mathcal{D}$ and $\mathcal{E}$. A homomorphism[13]: $\mathcal{A} \rightarrow \mathcal{B}$ is a pair of computable mappings (h, h') such that $h : Q_{\mathcal{A}} \rightarrow Q_{\mathcal{B}}$, $h' : \mathcal{D} \rightarrow \mathcal{E}$ and the following hold:

[13] A deterministic FA over F is an F-algebra (see Section 3.4 of [CouEng]) equipped with an output function. The first component of a homomorphism of automata is a homomorphism of the corresponding algebras.

(i) for every $f \in F$ and $q_1, ..., q_{\rho(f)}$, $q \in Q_{\mathcal{A}}$,
if $f[q_1, ..., q_{\rho(f)}] \rightarrow_{\mathcal{A}} q$, then $f[h(q_1), ..., h(q_{\rho(f)})] \rightarrow_{\mathcal{B}} h(q)$,
(ii) $Out_{\mathcal{B}} \circ h = h' \circ Out_{\mathcal{A}}$.

These conditions imply that, for every $t \in T(F)$, every position u of t,

(iii) $run_{\mathcal{B},t}(u) = h(run_{\mathcal{A},t}(u))$ and
(iv) $Comp(\mathcal{B}) = h' \circ Comp(\mathcal{A})$. $\square$

In practice, given $\mathcal{A}$ and h' such that $h' \circ Comp(\mathcal{A})$ is to be computed, the effort consists in finding h and $\mathcal{B}$ satisfying the above conditions and such that $\mathcal{B}$ uses, on each term, fewer and smaller states than $\mathcal{A}$ so as to be more efficient. We now generalize the construction of Example 23 to mappings γ, called *aggregate functions* in the context of databases ([AVH]).

Definitions 25: *Aggregate homomorphisms.*
Let F be an effectively given signature with symbols of maximum arity $r = \rho(F)$ and s be a positive integer. An *aggregate function* is a computable mapping $\gamma : \mathcal{P}_f(\mathcal{P}_f([r]^*)^s) \rightarrow \mathcal{E}$ (hence, $\mathcal{E}$ is effectively given; the set $\mathcal{P}_f(\mathcal{P}_f([r]^*)^s)$ is effectively given, cf. Section 2.4). We let $\perp_{\mathcal{E}} := \gamma(\emptyset)$. It is an *aggregate homomorphism* if there exist computable binary operations $\uplus^\gamma$ and $\circledast^\gamma$ on $\mathcal{E}$ such that $\uplus^\gamma$ is associative and commutative and, for every Z and Z' in $\mathcal{P}_f(\mathcal{P}_f([r]^*)^s)$, we have:

(a.1) $\gamma(Z \uplus Z') = \gamma(Z) \uplus^\gamma \gamma(Z')$ if Z and Z' are disjoint,
(a.2) $\gamma(Z \circledast Z') = \gamma(Z) \circledast^\gamma \gamma(Z')$ if $Z \subseteq \mathcal{P}_f(E)^s, Z' \subseteq \mathcal{P}_f(E')^s$ and $E \cap E' = \emptyset$,
(a.3) $\gamma(i.Z) = \gamma(Z)$ if $Z \subseteq \mathcal{P}_f([r]^*)^s$ and $i \in [r]$, (cf. Definitions 19(d) for $i.Z$).

Condition (a.3) is met by cardinality functions, for example, $\gamma(Z) := |Z|$ or $\gamma(Z)$ defined as the maximal cardinality of $|X_1| + ... + |X_s|$ for $(X_1, ..., X_s) \in Z$.With these definitions and notation we have:

Proposition 26 : If γ is an aggregate homomorphism and $\mathcal{A}_{P(\overline{X})}$ is a deterministic FA, there exists a deterministic FA $\mathcal{A}^\gamma_{P(\overline{X})}$ such that $Comp(\mathcal{A}^\gamma_{P(\overline{X})}) = \gamma \circ Comp(\mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}) = \gamma \circ \mathrm{Sat}\overline{X}.P(\overline{X})$.

Proof: We let $\mathcal{A} = \mathcal{A}_{P(\overline{X})}$, $\mathcal{A}^{\mathrm{Sat}} = \mathcal{A}^{\mathrm{Sat}}_{P(\overline{X})}$ and γ be an aggregate homomorphism : $\mathcal{P}_f(\mathcal{P}_f([r]^*)^s) \rightarrow \mathcal{E}$. We construct as follows a deterministic FA over F denoted by $\mathcal{A}^\gamma$:

(b.1) Its set of states is $[Q_{\mathcal{A}} \to \mathcal{E}]_f$, the set mappings $\sigma{:}Q_{\mathcal{A}} \to \mathcal{E}$, such that $\sigma(q) \neq \bot_{\mathcal{E}}$ for finitely many states q; each such mapping can be seen as (and implemented by) the finite set $\{(q, \sigma(q)) \mid \sigma(q) \neq \bot_{\mathcal{E}}\}$, hence the set of states is effectively given;

(b.2) Its transition for a nullary symbol a is $a \to \sigma$, with σ such that, for each state q,

$$\sigma(q) := \gamma(\{\varepsilon^w \mid w \in \{0,1\}^s, (a,w) \to_{\mathcal{A}} q\})$$
$$= \uplus^\gamma \{\!\{\gamma(\{\varepsilon^w\}) \mid w \in \{0,1\}^s, (a,w) \to_{\mathcal{A}} q\}\!\},$$

(b.3) Its transition for a binary symbol f is $f[\sigma_1, \sigma_2] \to \sigma$ such that for every state q,

$$\sigma(q) := \uplus^\gamma \{\!\{\gamma(\{\varepsilon^w\}) \circledast^\gamma \sigma_1(q_1) \circledast^\gamma \sigma_2(q_2) \mid w \in \{0,1\}^s, (f,w)[q_1,q_2] \to_{\mathcal{A}} q\}\!\},$$

(b.4) Its output function is :

$$Out^\gamma_{\mathcal{A}_{P(\overline{X})}}(\sigma) := \uplus^\gamma \{\!\{\sigma(q) \mid q \text{ is accepting}\}\!\}.$$

The transitions for symbols of other arities are defined similarly.

We claim that the pair (h, γ) such that $h(\sigma) = \gamma \circ \sigma$ is a homomorphism : $\mathcal{A}^{\mathrm{Sat}} \to \mathcal{A}^\gamma$. Here, the set $\mathcal{E}$ of Definition 24 is $\mathcal{P}_f(\mathcal{P}_f([r]^*)^s)$ and this fact will give the result by (iv). We now check conditions (i) and (ii) of this definition.

Condition (i). We first consider the case of $a \to_{\mathcal{A}^{\mathrm{Sat}}} \sigma$. Then, for each state q of $\mathcal{A}$ we have :

$$\sigma(q) = \{\varepsilon^w \mid w \in \{0,1\}^s, (a,w) \to_{\mathcal{A}} q\} \qquad \text{(cf. (4))},$$

and thus $a \to_{\mathcal{A}^\gamma} \gamma \circ \sigma$ by (b.2).

Let f be a binary symbol and $f[\sigma_1, \sigma_2] \to_{\mathcal{A}^{\mathrm{Sat}}} \sigma$. We must prove that

$$f[\gamma \circ \sigma_1, \gamma \circ \sigma_2] \to_{\mathcal{A}^\gamma} \gamma \circ \sigma.$$

Let q be a state of $\mathcal{A}$. We have by (5) :

$$\sigma(q) = \uplus\{\!\{\{\varepsilon^w\} \circledast 1.\sigma_1(q_1) \circledast 2.\sigma_1(q_2) \mid w \in \{0,1\}^s, \text{ and } (f,w)[q_1,q_2] \to_{\mathcal{A}} q\}\!\},$$

hence:

$$\gamma(\sigma(q)) = \uplus^\gamma \{\!\{\gamma(\{\varepsilon^w\}) \circledast^\gamma \gamma(1.\sigma_1(q_1)) \circledast^\gamma \gamma(2.\sigma_1(q_2)) \mid w \in \{0,1\}^s, \text{ and } (f,w)[q_1,q_2] \to_{\mathcal{A}} q\}\!\}.$$

Since $\gamma(1.\sigma_1(q_1)) = \gamma(\sigma_1(q_1))$ and $\gamma(2.\sigma_2(q_2)) = \gamma(\sigma_2(q_2))$ by (a.3), we get:

$$(\gamma \circ \sigma)(q) = \uplus^\gamma \{\!\{\gamma(\{\varepsilon^w\}) \circledast^\gamma (\gamma \circ \sigma_1)(q_1) \circledast^\gamma (\gamma \circ \sigma_2)(q_2) \mid w \in \{0,1\}^s, \text{ and } (f,w)[q_1,q_2] \to_{\mathcal{A}} q\}\!\}.$$

which, by (b.3), shows that $f[\gamma \circ \sigma_1, \gamma \circ \sigma_2] \rightarrow_{\mathcal{A}^\gamma} \gamma \circ \sigma$.

Condition (ii). We must prove that $Out_{\mathcal{A}^\gamma} \circ h = \gamma \circ Out_{\mathcal{A}^{\mathrm{Sat}}}$. Let σ be a state of $\mathcal{A}^{\mathrm{Sat}}$. By (6):

$Out_{\mathcal{A}^{\mathrm{Sat}}}(\sigma) = \uplus\{\sigma(q) \mid q \text{ is accepting}\}$, hence,
$\gamma(Out_{\mathcal{A}^{\mathrm{Sat}}}(\sigma)) = \quad \uplus^\gamma\{\gamma(\sigma(q)) \mid q \text{ is accepting}\} = Out_{\mathcal{A}^\gamma}(\gamma \circ \sigma)$,

which proves the desired fact. □

We denote $\mathcal{A}^\gamma$ by $\mathcal{A}^\gamma_{P(\overline{X})}$ for indicating the considered property although the transformations of $\mathcal{A}$ into $\mathcal{A}^{\mathrm{Sat}}$ and into $\mathcal{A}^\gamma$ apply in uniform ways to arbitrary FA over $F^{(s)}$.

4.2.2 Application to multispectra and other functions

Our aim is now to apply Proposition 26 to the computation of $\mathrm{MSp}\overline{X}.P(\overline{X})$, $\mathrm{Sp}\overline{X}.P(\overline{X})$, $\mathrm{MinCard}X.P(X)$ etc. In each case, we define an appropriate aggregate homomorphism.

Definitions 27 : *Some aggregate homomorphisms.*

We let $F, r, s, P(\overline{X}), \mathcal{A}$ be as in Definition 20. We define several aggregate homomorphisms γ: $\mathcal{P}_f(\mathcal{P}_f([r]^*)^s) \rightarrow \mathcal{E}$ for appropriate sets $\mathcal{E}$. We recall that $\perp = \gamma(\emptyset)$.

(a) For checking $t \models \exists\overline{X}.P(\overline{X})$. It is clear that $t \models \exists\overline{X}.P(\overline{X})$ if and only if $\mathrm{Sat}\overline{X}.P(\overline{X})(t)$ is not empty. We take $\mathcal{E} := \{True, False\}$ and γ such that $\gamma(Z) :=$ if $Z = \emptyset$ then $False$ else $True$. Hence $\perp_{\mathcal{E}} = False$. We take $\circledast^\gamma := \vee$ and $\uplus^\gamma := \vee$ and we have an aggregate homomorphism. The automaton $\mathcal{A}^\gamma$ is isomorphic to $\det(pr_s(\mathcal{A}))$. This case shows how the known construction fits in the present setting.

(b) For computing $\#\overline{X}.P(\overline{X})(t)$. We have discussed this case in Example 23. The general method applies with $\mathcal{E} := \mathbb{N}$, $\gamma(Z) := |Z|$, $\circledast^\gamma := \cdot$ and $\uplus^\gamma := +$. Here, $\perp_{\mathcal{E}} = 0$. We will denote by $\mathcal{A}^\#$ the automaton $\mathcal{A}^\gamma$.

(c) For computing $\mathrm{MSp}\overline{X}.P(\overline{X})(t)$. We let $\mathcal{E} := [\mathbb{N}^s \rightarrow \mathbb{N}]_f$ and $\gamma(Z)$ be such that $\gamma(Z)(\overline{n})$ (for $\overline{n} \in \mathbb{N}^s$) is the number of tuples $(X_1, ..., X_s)$ in Z such that $(|X_1|, ..., |X_s|) = \overline{n}$. Hence, $\mathrm{MSp}\overline{X}.P(\overline{X})(t) = \gamma(\mathrm{Sat}\overline{X}.P(\overline{X})(t))$.

If μ and μ' belong to $[\mathbb{N}^s \rightarrow \mathbb{N}]_f$, we define $\mu + \mu'$ and $\mu * \mu'$ in $[\mathbb{N}^s \rightarrow \mathbb{N}]_f$ by:

$(\mu + \mu')(n_1, ..., n_s) := \mu(n_1, ..., n_s) + \mu'(n_1, ..., n_s)$, and
$(\mu * \mu')(n_1, ..., n_s) :=$
$\qquad \Sigma\{\mu(p_1, ..., p_s).\mu'(n_1 - p_1, ..., n_s - p_s) \mid 0 \leq p_i \leq n_i\}$.

We take $\uplus^\gamma := +$ and $\circledast^\gamma := *$. Here, $\perp_{\mathcal{E}}$ is $\mathbf{0}$, the constant mapping with value 0. The verification that γ is homomorphic is routine. (In particular, if $Z \subseteq \mathcal{P}_f(E)^s$ and $Z' \subseteq \mathcal{P}_f(E')^s$ with $E \cap E' = \emptyset$, we have $\gamma(Z \circledast Z') = \gamma(Z) * \gamma(Z')$.) We will denote by $\mathcal{A}^{\mathrm{MSp}}$ the automaton $\mathcal{A}^\gamma$.

(d) For computing $\mathrm{Sp}\overline{X}.P(\overline{X})(t)$. We let $\mathcal{E} := [\mathbb{N}^s \to \{0,1\}]_f$ and $\gamma(Z)$ be such that $\gamma(Z)(\overline{n}) = 1$ if some tuple $(X_1, ..., X_s)$ in Z is such that $(|X_1|, ..., |X_s|) = \overline{n}$ and, otherwise, $\gamma(Z)(\overline{n}) = 0$. Hence, $\overline{n} \in \mathrm{Sp}\overline{X}.P(\overline{X})(t)$ if and only if $\gamma(\mathrm{Sat}\overline{X}.P(\overline{X})(t))(\overline{n}) = 1$. Then γ is homomorphic if we let the operations $\uplus^\gamma$ and $\circledast^\gamma$ as in the previous case but defined with min instead of $\cdot$ and max instead of $+$. We will denote by $\mathcal{A}^{\mathrm{Sp}}$ the automaton $\mathcal{A}^\gamma$.

(e) For computing $\mathrm{MinCard}X.P(X)(t)$ and $\mathrm{MaxCard}X.P(X)(t)$. Here $s = 1$. For computing $\mathrm{MinCard}X.P(X)(t)$, we let $\mathcal{E} := \mathbb{N} \cup \{\perp_{\mathcal{E}}\}$. We define $\gamma(Z) := \perp_{\mathcal{E}}$ if $Z = \emptyset$, and, otherwise, $\gamma(Z) := \min\{|X| \mid X \in Z\}$. Then γ is homomorphic if we let $\uplus^\gamma := \min$ (with $\min\{\perp_{\mathcal{E}}, n\} = n$) and $\circledast^\gamma := +$ (with $\perp_{\mathcal{E}} + n = \perp_{\mathcal{E}}$). The case of $\mathrm{MaxCard}X.P(X)$ is similar. We will denote by $\mathcal{A}^{\mathrm{MinCard}}$ and $\mathcal{A}^{\mathrm{MaxCard}}$ the obtained automata.

The reader will easily define an aggregate homomorphism for computing the number of sets X of minimal or maximal cardinality that satisfy $P(X)$. So the above list can be extended.□

Remark 28 : In each case (a)-(e) the structure $\langle \mathcal{E}, \uplus^\gamma, \circledast^\gamma, \gamma(\emptyset), \gamma((\emptyset, ..., \emptyset))\rangle$ is a semi-ring and some of these semi-rings are related by semi-ring homomorphisms; the corresponding automata $\mathcal{A}^\gamma$ are also related by homomorphisms, but we do not develop this formal aspect.

The system AUTOGRAPH implements separately the transformations of $\mathcal{A}$ into $\mathcal{A}^{\#}$, $\mathcal{A}^{\mathrm{MSp}}$, $\mathcal{A}^{\mathrm{Sp}}$, $\mathcal{A}^{\mathrm{MSp}}$, $\mathcal{A}^{\mathrm{MinCard}}$ and $\mathcal{A}^{\mathrm{MaxCard}}$. But the proof of Proposition 26 could be made into a "meta-transformation" of $\mathcal{A}$ into $\mathcal{A}^\gamma$ that would take a suitable representation of $\langle \mathcal{E}, \uplus^\gamma, \circledast^\gamma, \gamma(\emptyset), \gamma((\emptyset, ..., \emptyset))\rangle$ as input.□

The construction of Proposition 26 used for the different aggregate homomorphisms of Definitions 27 (or rather their associated operations $\circledast^\gamma$ and $\uplus^\gamma$) yields FA for computing $\#\overline{X}.P(\overline{X})(t)$, $\mathrm{MSp}\overline{X}.P(\overline{X})$, $\mathrm{Sp}\overline{X}.P(\overline{X})$, $\mathrm{MinCard}X.P(X)$ and $\mathrm{MaxCard}X.P(X)$. Next, we consider conditions ensuring that these automata are P-FA, FPT-FA or XP-FA. We recall that if the signature F is finite, the notions of P-FA, FPT-FA and XP-FA coincide. Our main application will be to the infinite signature F_∞ that generates all graphs.

We let $F, r, s, P(\overline{X})$ be as before, $pr = pr_s$ be the projection $F^{(s)} \to F$ that deletes Booleans (cf. Section 2.3) and $\alpha(\overline{X})$ be a function on terms that takes s set arguments. The constructions $\mathcal{A}^{\mathrm{Sat}}$ and $\mathcal{A}^\gamma$ of Definitions 21 and Proposition 26 are modifications of that of $\det(pr_s(\mathcal{A}))$. Lemma 7 shows the importance of the nondeterminism degree for analyzing the computation time of determinized automata.

Theorem 29 : (1) Let $P(\overline{X})$ be decided by a P-FA $\mathcal{A}$ over $F^{(s)}$ such that the mapping $ndeg_{pr(\mathcal{A})}$ is P-bounded. Then, the properties $\exists\overline{X}.P(\overline{X})$ and $\forall\overline{X}.P(\overline{X})$ are P-FA decidable and the functions MSp$\overline{X}.P(\overline{X})$, Sp$\overline{X}.P(\overline{X})$, $\#\overline{X}.P(\overline{X})$, MinCard$X.P(X)$ and MaxCard$X.P(X)$ are P-FA computable.

(2) If $\alpha(\overline{X})$ is computed by a P-FA such that $ndeg_{pr(\mathcal{A})}$ is P-bounded, then the function SetVal$\overline{X}.\alpha(\overline{X})$ is P-FA computable.

(3) These implications hold if we replace in these statements, the P- by FPT- or by XP-.

Since we have $ndeg_{pr(\mathcal{A})}(t) \leq |Q_{\mathcal{A}} \restriction pr^{-1}(t)|$ (cf. Proposition 12), we can replace in these statements, the P-, FPT- or XP-boundings of $ndeg_{pr(\mathcal{A})}$ by the corresponding ones for the mapping $t \longmapsto |Q_{\mathcal{A}} \restriction pr^{-1}(t)|$.

Proof : (1) The result follows from Lemma 7 for property $\exists\overline{X}.P(\overline{X})$ because this property can be checked by the determinized automaton $\det(pr(\mathcal{A}))$. The case of $\forall\overline{X}.P(\overline{X})$ is proved as in Theorem 17.

Next we consider the deterministic FA $\mathcal{A}^{\gamma}$ that computes MSp$\overline{X}.P(\overline{X})$, where γ is as in Definition 27(c). At position u in a term t, the state of $\mathcal{A}^{\gamma}$ is the set $\{(q, \gamma(Z(q,u))) \mid Z(q,u) \neq \emptyset\}$ of cardinality is bounded by $ndeg_{pr(\mathcal{A})}(t)$. Each component $\gamma(Z(q,u))$ is a function $\nu{:}\mathbb{N}^s \to \mathbb{N}$. Let $n = |t|$. This function is finite: it maps $[0,n]^s$ to $[0, 2^{s.n}]$ and has value 0 outside of $[0,n]^s$. It can be encoded by a word of length at most $(n+1)^s.\log(2^{s.n}) = O(n^{s+1})$. (The values of ν are written in binary). Hence the size of state is $O(ndeg_{pr(\mathcal{A})}(t).\|t\|^{s+1})$.

We must bound the time for computing the transitions and the output.

If $\mu, \mu'$are mappings : $[0,n]^s \to [0, 2^{s.n}]$ then, computing $\mu + \mu'$ takes time bounded by $(n+1)^s.\log(2^{s.n}) = O(n^{s+1})$ and, similarly, computing $\mu * \mu'$ takes time $O(n^{2s+1})$.

We let $p_1(\|t\|)$ bound the time for firing a transition of $\mathcal{A}$ (cf. Lemma 7). By (b.2) of Proposition 26, computing the transition of $\mathcal{A}^{\gamma}$ at a nullary symbol of t takes time $2^s.p_1(\|t\|)$.

By (b.3), computing the transition of $\mathcal{A}^{\gamma}$ at a symbol of t of arity r_1 needs at most $2^s.ndeg_{pr(\mathcal{A})}(t)^{r_1}$ operations $+$ and $*$ on mappings : $[0,n]^s \to [0, 2^{s.n}]$, hence a computation time bounded by $2^s.ndeg_{pr(\mathcal{A})}(t)^{r_2}.p_1(\|t\|).O(n^{2s+1})$ where r_2 is the maximal arity of a symbol in t. If $ndeg_{pr(\mathcal{A})}(t)$ is P-bounded, then the bound on the computation time of a transition is of same type.

Similarly, for computing the output (by (b.4) of Definition 25), we need at most $ndeg_{pr(\mathcal{A})}(t)$ checks that a state is accepting, with cost at most $p_3(\|t\|)$ for each (cf. Lemma 7) and the same number of additions. This gives a computation time bounded by $ndeg_{pr(\mathcal{A})}(t).(p_3(\|t\|) + O(n^{s+1}))$. Again, if $ndeg_{pr(\mathcal{A})}(t)$ is P-bounded, then the bound on the computation time of a transition is of same type.

We get the announced result for MSp$\overline{X}.P(\overline{X})$. For Sp$\overline{X}.P(\overline{X})$, $\#\overline{X}.P(\overline{X})$, MinCard$X.P(X)$ and MaxCard$X.P(X)$ we use the other aggregate functions of Definition 27. Each of them is a simplification of the one for MSp$\overline{X}.P(\overline{X})$. The size of $\gamma(Z(q,u))$ is smaller, and so are the computation times of the transitions and the output. Hence, the above argument applies as well.

(2) We now consider SetVal$\overline{X}.\alpha(\overline{X})$ where $\alpha(\overline{X})$ is computed by a deterministic FA $\mathcal{A}$. For each term t and assignment $\overline{X}$ of sets of positions of t, $\alpha(t, \overline{X}) = Out_{\mathcal{A}}(q_{\mathcal{A}}(t * \overline{X})))$. Hence, SetVal$\overline{X}.\alpha(\overline{X})(t)$ is the set of values $Out_{\mathcal{A}}(q)$ for $q \in q_{\det(pr(\mathcal{A}))}(t)$. (Note that SetVal$\overline{X}.\alpha(\overline{X})(t)$ is *not* computed from Sat$\overline{X}.[\alpha(\overline{X}) \neq \bot](t)$ that could be much too large.) Hence, the time taken to compute SetVal$\overline{X}.\alpha(\overline{X})(t)$ is that for computing the set $q_{\det(pr(\mathcal{A}))}(t)$ of cardinality at most $ndeg_{pr(\mathcal{A})}(t)$ plus that, bounded by $ndeg_{pr(\mathcal{A})}(t).p_3(\|t\|)$, for computing the final output. Hence, we conclude as in the cases considered in (1).

(3) The proofs are similar if p_1, p_2, p_3 and $ndeg_{pr(\mathcal{A})}$ are FPT- or XP-bounded. □

Remarks 30 : (a) Even if F is finite, we cannot omit in Theorem 29 the hypothesis that $pr(\mathcal{A})$ has a nondeterminism degree bounded in some way because the validity of $\exists\overline{X}.P(\overline{X})$ can be determined in polynomial time from either MSp$\overline{X}.P(\overline{X})$, Sp$\overline{X}.P(\overline{X})$, $\#\overline{X}.P(\overline{X})$, MinCard$X.P(X)$ or MaxCard$X.P(X)$. Otherwise, by the proof of Proposition 13, we would get **P=NP.**

(b) Let $F = F_{\infty}$. Every MS expressible graph property $P(\overline{X})$ is decided by a FA $\mathcal{A}$ over $F^{(s)}$ that has, for each k, a finite subautomaton over $F_k^{(s)}$ (cf. Remarks 8(1)). Hence, $\mathcal{A}$ is an FPT-FA, $ndeg_{pr(\mathcal{A})}$ is FPT-bounded and all functions of Theorem 29(1) are computable by FPT-FA. ■

Theorem 29 suggests the following questions.

Questions 31 : Can one "upgrade" the FA used to prove this theorem into FA of same type (P, FPT or XP) so as determine the following tuples:

(a) when we know that $t \models \exists\overline{X}.P(\overline{X})$, we want an s-tuple witnessing this fact (without constructing the full set Sat$\overline{X}.P(\overline{X})(t)$);

(b) we want an s-tuple $\overline{X}$ with given s-tuple of cardinalities belonging to Sp$\overline{X}.P(\overline{X})(t)$, that satisfies $P(\overline{X})$ in t;

(c) we want a set X of minimum or maximum cardinality satisfying $P(X)$ in the case where $t \models \exists X.P(X)$.

We answer these questions informally, that is, without developing more the formal framework. The transformation of $\mathcal{A}^{\mathrm{Sat}}$ into $\mathcal{A}^{\gamma}$ defined in Proposition 26 and used with the evaluations of Definition 27 consists in replacing in every state $\{(q, Z(q,u)) \mid Z(q,u) \neq \emptyset\}$ of $\mathcal{A}^{\mathrm{Sat}}$ each component $Z(q,u)$ by a simpler object $\gamma(Z(q,u))$. We answer Questions 31 by using the same idea, although the mappings γ will not be aggregate homomorphisms because condition (a.3) of Definition 25 will not hold. Here are the solutions.

(a) For selecting a tuple in Sat$\overline{X}.P(\overline{X})(t)$, we replace $Z(q,u)$ by one of its elements. That is, we let $\gamma(Z(q,u)) = \{\overline{X}\}$ for some $\overline{X} \in Z(q,u)$. For having a deterministic FA, such a mapping γ can choose the smallest tuple relative to

some lexicographic order whenever a choice must be made. For transitions on symbols of positive arity, we use $Z \circledast^{\gamma} Z'$ defined as $Z \circledast Z'$ (that contains at most one element if Z and Z' do so) and $Z \uplus^{\gamma} Z'$ that chooses (in a deterministic way) some element of $Z \cup Z'$. Each position in $t \in T(F)$ is a word over $[r]$ of length at most $ht(t)$. Hence, the size of a state is bounded by $ndeg_{pr(\mathcal{A})}(t).s.ht(t).\log(r)$, and computing the transitions and the output is polynomial in the size of states.

(b) For getting $\overline{X}$ having a given s-tuple of cardinalities (that necessarily must belong to $\mathrm{Sp}\overline{X}.P(\overline{X})(t)$), we let $\gamma(Z(q,u))$ be a subset $Z(q,u)$ such that:

> if $(X_1, ..., X_s) \in Z(q,u)$, there is in $\gamma(Z(q,u))$ a unique tuple $(Y_1, ..., Y_s)$ such that $(|Y_1|, ..., |Y_s|) = (|X_1|, ..., |X_s|)$.

The size of a state on a term with n positions is at most $ndeg_{pr(\mathcal{A})}(t).(n+1)^s.s.ht(t).\log(r)$. For computing transitions, we use $Z \circledast^{\gamma} Z' := Clean(Z \circledast Z')$ and $Z \uplus^{\gamma} Z' := Clean(Z \cup Z')$, where $Clean(Z)$ eliminates tuples from a set of tuples Z so as to meet the above condition (with Z in place of $Z(q,u)$). Computing the transitions and the output is polynomial in the size of states. Note that we obtain more than required: we get a "rich" version of $\mathrm{Sp}\overline{X}.P(\overline{X})(t)$ where each tuple of number is replaced by a corresponding tuple of sets of positions.

(c) Here we have $s = 1$ and we want a set X of minimum cardinality satisfying $P(X)$. Then $Z(q,u)$ is a set of sets and not a set of tuples of sets. We let $\gamma(Z(q,u))$ be some element of minimal cardinality in $\gamma(Z(q,u))$. Then $Z \uplus^{\gamma} Z'$ selects a set of minimal cardinality in $Z \cup Z'$, and $Z \circledast^{\gamma} Z'$ does the same in $Z \circledast Z'$. The latter is correct because $Min(Z \circledast Z') = Min(Z) \circledast Min(Z')$ where $Min(Z)$ is the set of sets in Z of minimal cardinality and Z and Z' are sets of subsets of disjoint sets.

In all cases, we obtain a FA of same type (P, FPT or XP) as the one $\mathcal{A}$ for $P(\overline{X})$, whose nondeterminism degree of $pr(\mathcal{A})$ is bounded as in Theorem 29, actually in terms of the number of states of $\mathcal{A}$ on a term.

Counter-example 32

We might hope that Theorem 29(1) and the corresponding statements for FPT-FA and XP-FA extends to formulas that mix existential and universal quantifiers over the variables in $\overline{X}$. This not the case. More precisely, the following assertion is *false*:

> (A) : Let F be a finite signature and $P(X_1, ..., X_s)$ be a property of terms over F decided by a P-FA $\mathcal{A}$ over $F^{(s)}$ such that $pr_s(\mathcal{A})$ has a polynomially bounded nondeterminism degree. Let $h : F^{(s)} \to F^{(s-1)}$ be the relabelling that deletes the last Boolean. The automaton $h(\mathcal{A} \times_{\wedge} \mathcal{A}_{Sgl(X_s)})$ decides $Q(X_1, ..., X_{s-1})$ defined as $\exists X_s.(P(X_1, ..., X_s) \wedge Sgl(X_s))$ and has the same property, that is,

$pr_{s-1}(\det(h(\mathcal{A}\times_{\wedge}\mathcal{A}_{Sgl(X_s)})))$ has a polynomially bounded nondeterminism degree.

Note that $\exists X_s$ in (A) is a first-order quantification. For disproving (A), we can modify slightly the construction of Proposition 13 and construct a property $P(X_1, ..., X_5)$ of terms over a finite signature F that is decided by a P-FA $\mathcal{A}$ over $F^{(5)}$, such that $pr_5(\mathcal{A})$ has a polynomially bounded nondeterminism degree but the property Q defined as :

$$\exists X_1 \forall x_2, x_3, x_4 \exists x_5 . P(X_1, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\})$$

is not **P**-decidable (unless **P** = **NP**). If (A) would be true, then the deterministic FA $\mathcal{B}$ over $F^{(1)}$ derived from $\mathcal{A}$ that decides, for each given X_1, the validity of

$$\forall x_2, x_3, x_4 \exists x_5 . P(X_1, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\})$$

would be such that $pr_1(\mathcal{B})$ has a polynomially bounded nondeterminism degree. Then Q would be **P**-decidable.

4.3 Summary of results

The following table summarizes the *preservation results* of this section : we mean by this that the classes of functions and properties that are P-FA, FPT-FA or XP-FA computable (or decidable) are preserved under constructions of three types: composition, FO constructions and MS constructions.

	Construction	Conditions and proofs
Composition	$g \circ (\alpha_1, ..., \alpha_r)$, if Pthen $\alpha_1$else α_2, $\neg P$, $P \vee Q$, $P \wedge Q$, $\alpha \restriction P$, $\alpha(S_1, ..., S_m)$, $P(S_1, ..., S_m)$.	g is **P**-computable, [Prop.15] [Proposition 15 and Thm. 17] $S_1, ..., S_m$: set terms, [Thm. 17]
FO const.	$\exists \overline{x}.P(\overline{x})$, $\forall \overline{x}.P(\overline{x})$, SetVal$\overline{x}.\alpha(\overline{x})$, Sat$\overline{x}.P(\overline{x})$, $\#\overline{x}.P(\overline{x})$.	[Theorem 17] [Corollary 19]
MS const.	$\exists \overline{X}.P(\overline{X})$, $\forall \overline{X}.P(\overline{X})$, SetVal$\overline{X}.\alpha(\overline{X})$, $\#\overline{X}.P(\overline{X})$,MSp$\overline{X}.P(\overline{X})$, Sp$\overline{X}.P(\overline{X})$, MinCard$X.P(X)$, MaxCard$X.P(X)$.	P or α is defined by P-FA $\mathcal{A}$ such that $ndeg_{pr(\mathcal{A})}$ is P-, FPT- or XP-bounded, [Theorem 29]

Table 1 : Preservation results from Section 4.

5 Application to terms and graphs of bounded clique-width

We now work out examples about terms and graphs. In particular, we specify FA for some basic functions on terms and graphs. The constructions will also use

the FA defined in Section 5 of [BCID] for several MS definable graph properties. Monadic second-order (MS) logic is reviewed in the appendix.

5.1 Properties of terms and functions on terms

In this section, we only consider finite signatures F, for which we recall that the notions of P-FA, FPT-FA and XP-FA are trivially identical.

Example 33 : *The prefix order.* For every term $t \in T(F)$, the prefix order on $Pos(t)$ is the unique strict linear order $<$ such that, for every $u \in Pos(t)$ with sequence of sons $u_1, ..., u_r$, $r > 0$, we have $\{u\} < Pos(t)/u_1 < ... < Pos(t)/u_r$. (For nonempty sets X, Y, $X < Y$ means that $x < y$ for every $x \in X$ and $y \in Y$.) The property $X_1 < X_2$ is MS expressible, hence, it is decided by a finite automaton. However, we can construct a deterministic automaton $\mathcal{A}$ with five states without using the general theorem and the logical expression of $X_1 < X_2$. The states are 0, 1, 2, 3 and *Error* (they do not depend on the signature F, and this construction actually works for infinite signatures, with the same states) and their meaning is defined by:

$0 : X_1 = X_2 = \emptyset$, (i.e., $q_{\mathcal{A}}(t * (X_1, X_2)) = 0$ if and only if $X_1 = X_2 = \emptyset$),

$1 : X_1 = \emptyset, X_2 \neq \emptyset$,

$2 : X_2 = \emptyset, X_1 \neq \emptyset$,

$3 : X_1 \neq \emptyset, X_2 \neq \emptyset, X_1 < X_2$; it is the unique accepting state,

Error : the other cases (in particular, $X_1 \cap X_2 \neq \emptyset$).

The transitions are easy to write and we get a finite automaton.

For a term t denoting a graph G, the restriction of this order to leaves is a linear order on the vertices of G. This order depends on t, and not only on G. It is useful in some cases as an auxiliary tool for expressing properties or functions of G that are *order-invariant*, *i.e.*, that are not affected by a change of order. See Section 5.2.1 for an example.

Properties and functions on subterms.

If X is a set of positions of a term t, we define $lca(X)$ as the least common ancestor of all positions in X, with $lca(\emptyset) = \perp$.

For a function $\alpha(X_1, \cdots, X_s)$, we define $\beta(X_1, \cdots, X_s, X_{s+1})$ as the function such that, in a term t:

$\beta(X_1, \cdots, X_s, X_{s+1})$ is $\alpha(X_1/lca(X_{s+1}), \cdots, X_s/lca(X_{s+1}))$ computed in the subterm $t/lca(X_{s+1})$ if $X_{s+1} \neq \emptyset$,

$\beta(X_1, \cdots, X_s, X_{s+1}) = \perp$ if $X_{s+1} = \emptyset$.

We denote this function β by $\alpha^{\downarrow}$. (This definition is applicable to the case where α is a property.) The transformation of α into $\alpha^{\downarrow}$ is a kind of relativization. Another one will be considered in Proposition 39.

Proposition 34 : From a P-FA $\mathcal{A}$ computing a function $\overline{\alpha}$:$T(F^{(s)}) \to \mathcal{D}$, we can construct a P-FA $\mathcal{B}$ that computes the function $\overline{\alpha^{\downarrow}}$:$T(F^{(s+1)}) \to \mathcal{D}$.

Proof : We give the construction for $s = 0$. For the general case, it suffices to replace F by $F^{(s)}$. We define $\mathcal{B}$ with set of states $Q_{\mathcal{A}} \cup (Q_{\mathcal{A}} \times Q_{\mathcal{A}})$ and transitions such that, for every term $t \in T(F)$ and $X \subseteq Pos(t)$, we have :

$q_{\mathcal{B}}(t * X) = q_{\mathcal{A}}(t) \in Q_{\mathcal{A}}$ if $X = \emptyset$,
$q_{\mathcal{B}}(t * X) = (q_{\mathcal{A}}(t), q_{\mathcal{A}}(t/lca(X))) \in Q_{\mathcal{A}} \times Q_{\mathcal{A}}$ otherwise;
the output function is defined by $Out_{\mathcal{B}}((q,p)) := Out_{\mathcal{A}}(p)$
and $Out_{\mathcal{B}}(q) := \perp$ for all $p, q \in Q_{\mathcal{A}}$.

The following table only shows the transitions for a nullary symbol a and a binary one f. The similar transitions for all other symbols are easy to write. (Since $\alpha^{\downarrow}$ takes as argument a set of positions, all symbols are equipped with Booleans.) The FA $\mathcal{B}$ is also a P-FA. $\square$

Transition of $\mathcal{B}$	Condition
$[\mathbf{a}, 0] \to q$ $[\mathbf{a}, 1] \to (q, q)$	$\mathbf{a} \to_{\mathcal{A}} q$
$[f, 0](q_1, q_2) \to q$ $[f, 0]((q_1, p), q_2) \to (q, p)$ $[f, 0](q_1, (q_2, p)) \to (q, p)$ $[f, 0]((q_1, p_1), (q_2, p_2)) \to (q, q)$	$f(q_1, q_2) \to_{\mathcal{A}} q$
$[f, 1](q_1, q_2) \to (q, q)$ $[f, 1]((q_1, p), q_2) \to (q, q)$ $[f, 1](q_1, (q_2, p)) \to (q, q)$ $[f, 1]((q_1, p_1), (q_2, p_2)) \to (q, q)$	$f(q_1, q_2) \to_{\mathcal{A}} q$

Table 2: Automaton $\mathcal{B}$ for $\alpha^{\downarrow}$.

Remark: The reader will easily modify this automaton into a smaller one for the function $\alpha^{\downarrow} := \alpha^{\downarrow} \restriction Sgl(X_{s+1})$, i.e., such that :

$\alpha^{\downarrow}(t, X_1, \cdots, X_s, X_{s+1}) = \alpha(t/u, X_1/u, \cdots, X_s/u)$ if $X_{s+1} = \{u\}$ for some u,
$\alpha^{\downarrow}(t, X_1, \cdots, X_s, X_{s+1}) = \perp$ otherwise.

5.1.1 Uniform terms

First, we observe that the height of a term t can be computed by an obvious P-FA $\mathcal{A}_{ht}$ whose states are positive integers (the state at u is $ht(t/u)$ where $ht(a) = 1$ for a nullary symbol a). A term t is *uniform* (this is denoted by $Unif(t)$) if and only if any two leaves of its syntactic tree are at the same distance to the root. This is equivalent to the condition that, for every position u with sons u' and u", the subterms t/u' and t/u" have same height. The automaton $\mathcal{A}_{ht}$ can thus be modified into a P-FA $\mathcal{A}_{Unif}$ that decides uniformity, with set of states is $\mathbb{N}_+ \cup \{Error\}$, such that:

$q_{\mathcal{A}_{Unif}}(t) = ht(t)$ if t is uniform
$q_{\mathcal{A}_{Unif}}(t) = Error$ if t is not uniform.

This automaton can be combined with others to yield more complicated functions. Let us for example consider the set $W(t)$ of triples $(u, |t/u|, ht(t/u))$ such that t/u is a maximal uniform subterm (that is, u is closest to the root such that t/u is uniform).

To detail this example, we define for a term t :

$W := \mathrm{SetVal}\, u.\alpha(u)$ where
$\alpha(u) := (\alpha_1(u), \alpha_2(u), \alpha_3(u)) \restriction MaxUnif(u)$,
$MaxUnif(u)$ expresses that t/u is a maximal uniform subterm of t,
$\alpha_1(u) := u$,
$\alpha_2(u) := |t/u|$,
$\alpha_3(u) := ht(t/u)$.

Our objective is to apply Proposition 15 and Theorem 17(2). The three functions $\alpha_1, \alpha_2, \alpha_3$ are P-FA computable and $MaxUnif(u)$ is $MaxUnif(\{u\})$ where $MaxUnif(U)$ is defined by:

$$Unif^{\downarrow}(U) \wedge Sgl(U) \wedge \neg\exists W(Sgl(W) \wedge son(W, U) \wedge Unif^{\downarrow}(W)).$$

Properties $Unif^{\downarrow}(U)$ and $Unif^{\downarrow}(W)$ are P-FA decidable by Proposition 34 applied to $\mathcal{A}_{Unif}$. So are $Sgl(W) \wedge son(W, U)$ (see the appendix for *son* and MS logic on terms), $\neg\exists W(Sgl(W) \wedge son(W, U) \wedge Unif^{\downarrow}(W))$ by Propositions 15,16 and Theorem 17, and finally $MaxUnif(U)$, again by Proposition 15. Hence, α is P-FA computable by Proposition 15, and W is also by Theorem 17(2).

5.1.2 An extension of monadic second-order logic

We generalize this example by defining an extension of MS logic where we can express P-FA functions and properties that are not MS expressible.

Definition 35 : *An extension of MS logic on terms.*

Given a finite signature F, we consider properties and functions constructed in the following way:

(a) We use free set variables X_1, ..., X_s (that will not be quantified), first-order variables, $y_1, ..., y_m$ and set terms over $X_1, ..., X_s, Y_1, ..., Y_m$ (the variables $Y_1, ..., Y_m$ will always be restricted by conditions $Sgl(Y_1), ..., Sgl(Y_m)$ and will correspond to $y_1, ..., y_m$, cf. the end of Section 2.3).

(b) As basic properties, we use $Unif$ and all properties P expressible by MS formulas (that can use other bound variables than X_1, ..., X_s, Y_1, ... Y_m). These properties depend on F. As basic functions, we use ht, $Card$ and Id.

(c) We construct properties from already constructed properties P, Q, ... and from functions $\alpha, \beta, \alpha_1, ..., \alpha_r, ...$ by the following compositions:

$P \wedge Q, P \vee Q, \neg P$,

$R \circ (\alpha_1, ..., \alpha_r)$ where R is an r-ary **P**-decidable relation on $\mathcal{D}$,

$P^{\downarrow}$ and $P(S_1, ..., S_p)$ where $S_1, ..., S_p$ are set terms over $X_1, ..., X_s, Y_1, ..., Y_m$,

$\exists \overline{y}.P(\overline{y})$ where $\overline{y}$ is a tuple of variables among $y_1, ..., y_m$.

(d) Similarly, we construct functions in the following ways:

$g \circ (\alpha_1, ..., \alpha_r)$ where g is **P**-computable : $\mathcal{D} \to \mathcal{D}^r$,

$\alpha^{\downarrow}$ and $\alpha(S_1, ..., S_p)$ where $S_1, ..., S_p$ are set terms over $X_1, ..., X_s, Y_1, ..., Y_m$,

$\mathrm{SetVal}\overline{y}.\alpha(\overline{y}), \#\overline{y}.P(\overline{y})$ and $\mathrm{Sat}\overline{y}.P(\overline{y})$ where $\overline{y}$ is a tuple of variables among $y_1, ..., y_m$.

We assume that we have for R in (c) and g in (d) a polynomial-time algorithm. We denote by $\mathcal{PF}(F)$ the set of all these formulas.

Remarks : (1) Our description has some redundancy: for example, the truth value of $\exists \overline{y}.P(\overline{y})$ can be expressed as $g \circ \#\overline{y}.P(\overline{y})$ where $g(0) := False$ and $g(n) := True$ for $n > 0$, so that g is **P**-computable.

(2) A generalized quantification of the form "there exist at least p elements y satisfying $P(y)$" can be expressed by $\#\overline{y}.P(\overline{y}) \geq p$, hence, is also of the form $g \circ \#\overline{y}.P(\overline{y})$ for a **P**-computable function g.

(3) The use of set terms permits to use $P(X \cup \{y\})$ where y is bound. For example, we can write:

$$Unif^{\downarrow}(X) \wedge \forall Y.(Sgl(Y) \wedge X \cap Y = \emptyset \Longrightarrow \neg Unif^{\downarrow}(X \cup Y)).$$

Theorem 36 : Let F be a finite signature. Every property (or function) defined by a formula of $\mathcal{PF}(F)$ is decidable (or computable) by a P-FA over

F or $F^{(s)}$. Such an automaton can be constructed from automata for the basic properties and functions.

Proof : Straightforward by induction on the structure of a formula describing the considered property or function by the results of Section 4.1, that are based on uniform construction schemes.□

The language $\mathcal{PF}(F)$ does not exhaust the possibilities of extension of MS logic that yield P-FA computable properties and functions. We can for example introduce a *relativized height* $ht(t, X)$ for $t \in T(F)$ and $X \subseteq Pos(t)$, defined as the maximal number of elements of X on a branch of the syntactic tree of t. A P-FA is straightforward to construct for this function, that does not seem to be expressible by a formula of $\mathcal{PF}(F)$. However, there are strong limitations to such extensions (see Section 5.3).

5.2 Properties of graphs and functions on graphs

Graphs are always given by terms over F_∞ or F_∞^{u}, and moreover, by good terms for having nicer time bounds (cf. Section 2.2 and the appendix for the corresponding preprocessing). Table 3 reviews constructions of P-FA from [BCID]. In this table, $Partition(X_1, ..., X_s)$ means that $(X_1, ..., X_s)$ is a partition of the vertex set, St that the considered graph has no edge (i.e., is *stable*), $Link(X_1, X_2)$ that it has edges from X_1 to X_2, $Path(X_1, X_2)$ that X_1 consists of two vertices linked by a path with vertices in X_2, $Clique$ that the graph is a clique, $Conn$ that it is connected, $Cycle$ that it has an undirected cycle and $DirCycle$ a directed cycle. Finally, $edg(X_1, X_2)$ is equivalent to $Link(X_1, X_2) \wedge Sgl(X_1) \wedge Sgl(X_2)$.

Property	Size of a state
Sgl, $Partition$, $X_1 \subseteq X_2$, $X_1 = \emptyset$	independent of k
$edg(X_1, X_2)$	$O(\log(k))$
St, $Link(X_1, X_2)$	$O(k)$
$Path(X_1, X_2)$, $DirCycle$, $Clique$	$O(k^2)$
$Conn, Cycle$	$O(\log(k) \cdot \min\{n, k.2^{O(k)}\})$

Table 3: Sizes of states for some basic properties.

From the definitions of the automata for $Conn$ and $Cycle$ ([BCID], Sections 6 and 5.2.7), we get the immediate upper bound $O(\log(k).k.2^{O(k)})$ for the size of a state. However, a careful inspection also yields the upper bound $O(\log(k).n)$. Hence, contrary to the first impression, the automata for connectedness and existence of undirected cycles are actually P-FA. All others are clearly P-FA (computing the transitions involves no complicated calculations).

We now review how Theorem 29 applies to a graph property $P(\overline{X})$ that is MS expressible. (If $P(\overline{X})$ is an MS expressible property of graphs, it can be translated into an MS expressible property on terms that define graphs, hence,

Theorem 29 is applicable). There exists in this case an FPT-FA $\mathcal{A}$ over $F_\infty^{(s)}$ that decides $P(\overline{X})$, and the nondeterminism degree of $pr(\mathcal{A})$ is FPT-bounded, with a bound that depends only on the smallest integer k such that the considered term is in $T(F_k)$. It follows from Theorem 29(1) that the properties $\exists\overline{X}.P(\overline{X})$ and $\forall\overline{X}.P(\overline{X})$ are decidable by FPT-FA (this fact is actually well-known) and that the functions $\mathrm{MSp}\overline{X}.P(\overline{X})$, $\mathrm{Sp}\overline{X}.P(\overline{X})$, $\#\overline{X}.P(\overline{X})$, $\mathrm{MinCard}X.P(X)$ and $\mathrm{MaxCard}X.P(X)$ are computable by FPT-FA. We will consider properties and functions that are not MS expressible (cf. the introduction for the notion of an MS expressible function), but to which we can apply the logic based constructions of Section 4.

It is not hard to build a P-FA $\mathcal{G}$ over F_∞, that checks if a term is good. It follows that if we have constructed a FA $\mathcal{A}$ that works with a time bound of type P, FPT or XP on good terms, then the FA $\mathcal{A} \times_\wedge \mathcal{G}$ rejects the terms that are not good and works as $\mathcal{A}$ on the good ones. Its type of time bound on all terms is of same type as that of $\mathcal{A}$. A polynomial-time transformation of a term into an equivalent good one is described in the appendix (Proposition 44). We now recall from [BCID] another condition on terms that is checkable by a P-FA and can be assumed thanks to a polynomial time preprocessing.

Definition 37 : *Irredundant term.*
A term $t \in T(F_\infty)$ is *irredundant* if, for each of its subterms of the form $\overrightarrow{add}_{a,b}(t')$ (or $add_{a,b}(t')$), there is in $G(t')$ no edge from an a-port to a b-port (or between an a-port and a b-port). This means that each such operation $\overrightarrow{add}_{a,b}$ or $add_{a,b}$ creates exactly $m.m'$ edges where m is the number of a-ports of $G(t')$ and m' is that of b-ports. We may have no a-port or no b-port in $G(t')$: the operation $\overrightarrow{add}_{a,b}$ (or $add_{a,b}$) is then useless, but the considered term t may still be irredundant according to the definition. It follows in particular that irredundancy is preserved by the replacement of subterms by the nullary symbol $\varnothing$ that denotes the empty graph. Many automata will be constructed below so as to run on irredundant terms.

We will prove in Proposition 44 (in the appendix) that every term t in $T(F_\infty)$ can be transformed in polynomial time into an equivalent good and irredundant term. Hence, we can build automata $\mathcal{A}$ intended to work correctly on good irredundant terms and and that may give possibly erroneous outputs on the others. Furthermore, we can build a P-FA $\mathcal{GI}$ that checks if the input term is good and irredundant (see the appendix). So the FA $\mathcal{A} \times_\wedge \mathcal{GI}$ gives the correct outputs on good irredundant terms and reject the others. This automaton has the same type (P, FPT or XP) as $\mathcal{A}$.

We now extend to functions Lemma 15 of [BCID] that concerns only the relativization of properties.

Definition 39 : *Relativization.* For a function $\alpha(X_1, \cdots, X_s)$ with (vertex) set arguments in a graph G, we define $\beta(X_1, \cdots, X_s, Y)$ as $\alpha(X_1 \cap Y, \cdots, X_s \cap Y)$ computed in the induced subgraph $G[Y]$. Hence, it is defined if and only if

$\alpha(X_1 \cap Y, \cdots, X_s \cap Y)$ is defined in $G[Y]$. We have in particular (with the notation of [BCID]) $Def(\beta)(X_1, \cdots, X_s, Y) = Def(\alpha)(X_1, \cdots, X_s)[Y]$ (in G).

We denote this function β by $\alpha^{[Y]}$. If α is a property P and $s = 0$, we use the standard notation $P[Y]$ instead of $P^{[Y]}$.

It is now convenient to take $Y = X_{s+1}$. We define h as the mapping: $F_\infty^{(s+1)} \to F_\infty^{(s)}$ such that, for every $\mathbf{c} \in \mathbf{C}$ and $w \in \{0,1\}^s$, we have $h((\mathbf{c}, w0)) := \varnothing$ and $h((\mathbf{c}, w1)) := (\mathbf{c}, w)$. With these hypotheses and notation, we have $\overline{\beta} = \overline{\alpha} \circ h$.

Proposition 39 : If $\alpha(X_1, \cdots, X_s)$ is a function (or a property) that can be computed (or decided) by a P-FA or an FPT-FA or an XP-FA, then the same holds for $\alpha^{[X_{s+1}]}(X_1, \cdots, X_s)$.

Proof : Let $\alpha(X_1, \cdots, X_s)$ be computed (or decided) by a FA $\mathcal{A}$. Then $\alpha^{[X_{s+1}]}(X_1, \cdots, X_s)$ is computed (or decided) by $h^{-1}(\mathcal{A})$ where h is defined in Definition 39. The result follows from Proposition 11(2).□

Remark: This relativization does not apply to terms because if S is the logical structure representing a term and X is a set of positions of t, then $S[X]$ does not always represent a term. Proposition 34 deals with a related notion for terms.

5.2.1 Counting subgraphs

Let H be a connected undirected graph. An induced subgraph of an undirected graph G is H-*induced* if it is isomorphic to H. A *triangle* is thus a K_3-induced subgraph. Our objectives are to use FA to count and to enumerate the H-induced subgraphs of a given graph. The property of a set $X \subseteq V_G$ that $G[X] \simeq H$ is MS expressible. Hence, automata that compute the functions $\#X.$"$G[X] \simeq H$" and Sat$X.$"$G[X] \simeq H$" will give us the desired algorithms. The property $G[X] \simeq H$ implies that X has fixed cardinality $|V_H|$. Hence, we can apply Corollary 19 (with the remark at the end of Section 4.1). However, a direct construction yields smaller automata. We do it on the example where H is $House$, i.e., the graph K_5 with vertices $1, 2, 3, 4, 5$ minus the four edges $1-4, 1-5, 3-4$ and $2-5$.

A FA automaton over F_k^{u} with $O(k^2)$ states for $edg(X, Y)$ can be constructed (see [BCID], Section 5.1.2). We let $P(\overline{X})$, with $\overline{X} = (X_1, X_2, X_3, X_4, X_5)$, stand for:

$edg(X_1, X_2) \wedge edg(X_1, X_3) \wedge edg(X_2, X_3) \wedge edg(X_2, X_4) \wedge edg(X_4, X_5) \wedge$
$edg(X_3, X_5) \wedge \neg edg(X_1, X_4) \wedge \neg edg(X_1, X_5) \wedge \neg edg(X_3, X_4) \wedge \neg edg(X_2, X_5)$.

From Propositions 15 and 16, we get for $P(\overline{X})$ an automaton that uses $O(k^{20})$ states on terms in $T(F_k^{\mathrm{u}(5)})$, but a direct construction yields a P-FA with $2^7 \cdot (k+1)^5 + 1$ states on these terms ($2^m \cdot (k+1)^n + 1$ states for a graph H with n vertices and m edges). By Corollary 19, we get from it automata that

compute $\#\overline{X}.P(\overline{X})$ and $\mathrm{Sat}\overline{X}.P(\overline{X})$. However, the number of $House$-induced subgraphs of $G(t)$ is only the half of $\#\overline{X}.P(\overline{X})$ because the graph $House$ has two automorphisms (including the identity). This means that the automaton that computes $\#\overline{X}.P(\overline{X})$ does some useless computations. We can avoid this drawback by replacing $P(\overline{X})$ by $P(\overline{X}) \wedge X_2 < X_3$. See Section 5.1 for a very simple FA that computes $<$. (The construction of Example 33 can easily be adapted to the present case where X_2 and X_3 are singletons containing only leaves.) The role of this condition is to select a single 5-tuple for each $House$-induced graph.

Note that the linear order $<$ we use depend on the term t, and not only on the graph $G(t)$. However, the value $\#\overline{X}.P(\overline{X})$ is the same for all terms t. It is defined by means of an order but this definition is *order-invariant*. (See [Cou96] on this notion and [EKS] for its applications to model-checking.)

The same improvement applies for the enumeration problem in order to avoid duplications in the enumeration of $House$-induced subgraphs.

Even without using the linear order, Theorem 17 and Corollary 19 yield P-FA that compute the functions $\#X."G[X] \simeq H"$ and $\mathrm{Sat}X."G[X] \simeq H"$ for each fixed graph H.

Remark: We might also wish to determine the maximal number of pairwise vertex-disjoint H-induced subgraphs. We can do that with *edge set quantifications*. Let $E_H(X)$ mean that X is the set of edges of the union of a set of vertex-disjoint H-induced subgraphs. This number is thus $\mathrm{MaxCard}X.E_H(X)$. But this expression uses edge set quantification, which does not yield appropriate automata over F_∞ and FPT algorithms for graphs of bounded clique-width. It works for graphs of bounded tree-width. The reader will find in [CouEng] Section 6.3.5 and in [Cou12] methods for adapting the present constructions to edge set quantifications and graphs of bounded tree-width.

By adapting a construction of [GHO], we can build an XP-FA that checks if a directed graph is Hamiltonian, although this property is not MS-expressible. However, this property is MS_2-expressible, *i.e.*, expressible in the extension of MS logic allowing edge set quantifications (see the appendix). It is perhaps possible to perform a similar construction for computing $\mathrm{MaxCard}X.E_H(X)$ for graphs of bounded clique-width.

5.2.2 Edge counting and degree

For a p-graph G and $X \subseteq V_G$, we denote by λ_X the mapping that gives, for each label a the number of a-ports in X. If $X = V_G$, we denote it by λ_G.

We denote by $\Lambda[C, n]$ the set of mappings $\lambda : C \to [0, n]$ such that $\Sigma_{i \in C}\lambda(i) \leq n$. If C has k elements, this set has cardinality $\binom{n+k}{k}$ (by an easy bijective proof), hence its cardinality is between $((n+k)/k)^k$ and $(n+1)^k$, and so is $\Theta(n^k)$ for fixed k. We will bound it by $(n+1)^k$.

Counting the edges of induced subgraphs

Let X be a set of vertices of a directed graph G. (All graphs are loop-free). We let $e(X)$ be the number of edges of $G[X]$. For computing this number, we define a deterministic FA $\mathcal{A}_k$ over $F_k^{(1)}$, intended to run on irredundant terms written with labels in $C := [k]$. Its set of states is $\mathbb{N} \times [C \to \mathbb{N}]$ and we want that :

$$q_{\mathcal{A}_k}(t * X) = (e(X), \lambda_X).$$

The transitions are easy to write. We only give the example of:

$$\overrightarrow{add}_{a,b}[(m, \lambda)] \to (m + \lambda(a).\lambda(b), \lambda).$$

This transition is correct with respect to the above requirement because t is assumed irredundant. The result $e(X)$ is the first component of the state reached at the root of $t * X$.[14].

On a term that denotes a graph with n vertices, each state belongs to the set $[0, n(n-1)] \times \Lambda(C, n)$ of cardinality less than $(n+1)^{k+2}$, hence, has size $O(k.\log(n))$ (the integers m and the values of λ are written in binary notation). Transitions and outputs can be computed in time $O(k.\log(n))$. Hence, $\mathcal{A}_k$ is a P-FA. As explained in Section 2.1, it is convenient to represent a function $\lambda : C \to \mathbb{N}$ by the set $\{(a, \lambda(a)) \mid \lambda(a) \neq 0\}$. This formalization implies that $\mathcal{A}_k$ is a subautomaton of $\mathcal{A}_{k'}$ if $k < k'$. Hence, the union of the automata $\mathcal{A}_k$ is a FA $\mathcal{A}_\infty$ over $F_\infty^{(1)}$. This FA is a P-FA on good irredundant terms t because $k < \|t\|$ for them. By observations made in Definition 37, we get a P-FA that rejects the terms that are not good and irredundant[15].

Note that the value $e(X)$ is not the cardinality of some set Y satisfying an MS formula $\varphi(X, Y)$ because we do not allow edge set quantifications, and this description needs that Y denotes a set of edges. A similar observation holds for $e(X_1, X_2)$ considered next.

Counting the edges between disjoint sets of vertices

We first consider directed graphs. We generalize the notion of outdegree of a vertex by defining $e(X_1, X_2)$ as the number of edges from X_1 to X_2 if X_1 and X_2 are disjoint sets of vertices and as $\perp$ otherwise. Hence $e(\{x\}, V_G - \{x\})$ is the outdegree of x in G. We define a deterministic FA $\mathcal{B}_k$ over $F_k^{(2)}$, intended to run on irredundant terms written with labels in $C := [k]$. Its set of states is $(\mathbb{N} \times [C \to \mathbb{N}] \times [C \to \mathbb{N}]) \cup \{Error\}$ and we want that :

$q_{\mathcal{B}_k}(t * (X_1, X_2)) = Error$ if $X_1 \cap X_2 \neq \emptyset$, and
$q_{\mathcal{B}_k}(t * (X_1, X_2)) = (e(X_1, X_2), \lambda_{X_1}, \lambda_{X_2})$ otherwise.

[14] We can alternatively construct an automaton over F_k that computes the number of edges of $G(t)$ and derive $\mathcal{A}_k$ from it by using Proposition 42(1) (with $s = 0$).

[15] It is not hard to see that a term t in $T(F^{(s)})$ is good (resp. irredundant) if and only if $pr_s(t)$ is.

The transitions are easy to write. We only give the example of:

$$\overrightarrow{add}_{a,b}[(m, \lambda_1, \lambda_2)] \rightarrow (m + \lambda_1(a).\lambda_2(b), \lambda_1, \lambda_2),$$

which is correct with respect to the above requirement because t is assumed irredundant.

On a term that denotes a graph with n vertices, each state belongs to the set $([0, (n-1)^2] \times \Lambda(C, n) \times \Lambda(C, n)) \cup \{Error\}$ of cardinality less than $(n+1)^{2k+2}$ hence, has size $O(k.\log(n))$ (the integers m and the values of λ_1 and λ_2 are written in binary notation). Transitions and outputs can be computed in time $O(k.\log(n))$. Hence, $\mathcal{B}_k$ is a P-FA. As for $\mathcal{A}_k$ above, the union of the automata $\mathcal{B}_k$ is a P-FA $\mathcal{B}_\infty$ over $F^{(2)}_\infty$, constructed for good and irredundant terms.

The function $e(X, X^c) \restriction Sgl(X)$ gives the outdegree of a single vertex (the unique element of X). A P-FA with smaller states (but still of size $O(k.\log(n))$) can be built for it. For an undirected graph, we define $e(X_1, X_2)$ as the number of edges between X_1 and X_2 if X_1 and X_2 are disjoint and $\perp$ otherwise. The construction is similar with:

$$add_{a,b}[(m, \lambda_1, \lambda_2)] \rightarrow (m + \lambda_1(a).\lambda_2(b) + \lambda_1(b).\lambda_2(a), \lambda_1, \lambda_2),$$

again for irredundant terms.

Maximum directed cut

For a directed graph G, we want to compute the maximal number of edges from a subset X of V_G to its complement, hence the maximal value of $e(X, X^c)$. This problem is considered in [LKM] and [GHO]. The P-FA for $e(X, X^c)$ uses less than $(n+1)^{2k+2}$ states on a term in $T(F_k)$ denoting a graph G with n vertices. By Lemma 7 and Theorem 29, we get an algorithm that computes the set $\{e(X, X^c) \mid X \subseteq V_G\}$ in time $O(n^{4k+a})$ for some constant a. The article [GHO] gives an algorithm taking time $O(n^{4.2^{r(G)}+b})$ where $r(G)$ is the *bi-rankwidth* of the considered graph G. We recall that $r(G)/2 \leq cwd(G) \leq 2.2^{r(G)}$ ([KanRao]). Hence, our method gives an algorithm of comparable time complexity.

Maximum degree

As we are discussing degrees, we observe that the F_k-automata of [BCID], Section 5.2.6 that check if an undirected graph G has maximal degree $MaxDeg(G)$ at most some fixed number d can be made into a P-FA $\mathcal{A}_{MaxDeg}$ that computes the function $MaxDeg$. Its states are pairs $(\alpha, \lambda) \in [C \rightarrow \mathbb{N}] \times [C \rightarrow \mathbb{N}]$ such that $q_{\mathcal{A}_{MaxDeg}}(t) = (\alpha_{G(t)}, \lambda_{G(t)})$ where, for $a \in C$, $\alpha_{G(t)}(a)$ is the maximal degree of an a-port. The size of a state on a term $t \in T(F_k)$ is bounded by $2k \cdot \log(|V_{G(t)}|)$.

5.2.3 Regularity of a graph

The regularity of an undirected graph is not MS expressible: to prove that, we observe that the complete bipartite graph $K_{n,m}$ is regular if and only if $n = m$ and we apply the arguments of Proposition 5.13 of [CouEng].

That a graph is not regular can be expressed by a FA constructed from the formula $\exists X, Y.(P(X,Y) \wedge Sgl(X) \wedge Sgl(Y))$ where $P(X,Y)$ is the property $e(X, X^c) \neq e(Y, Y^c)$. By the previous construction, Propositions 15, 16 and Corollary 34, this property is P-FA decidable, and we can apply Proposition 11(1) to get a P-FA for checking that a graph is not regular, hence a P-FA that checks regularity. However, we can construct directly a simpler P-FA without using an intermediate nondeterministic automaton.

Let G be defined by an irredundant term t. Here is the key fact: if in $G(t/u)$ two a-ports x and y have degrees d and d' these vertices have in G degrees $d+e$ and $d'+e$ for some e, because every edge added between x and a b-port z by operations outside of t/u (i.e., in the context of u in t) is also added between y and z. Since t is irredundant, such edges do not exist already in $G(t/u)$ and so, the degrees of x and y are increased by the same value e by these operations. If the degrees are different in $G(t/u)$, so are they in G. We recall that $\pi(G)$ is the set of port labels of the vertices of G and $\lambda_G(a)$ is the number of its a-ports.

The notation is as in Section 5.2.2. The set of states of $\mathcal{A}_{Reg,k}$ is defined as $([C \to (\mathbb{N} \cup \{\perp\})] \times [C \to \mathbb{N}]) \cup \{Error\}$ and we want that, for every term $t \in T(F_k)$:

$q_{\mathcal{A}_{Reg,k}}(t) = Error$ if two a-ports of $G(t)$ is have different degrees,

$q_{\mathcal{A}_{Reg,k}}(t) = (\deg_{G(t)}, \lambda_{G(t)})$ where, for every a in $\pi(G(t))$, $\deg_{G(t)}(a)$ is the common degree of all a-ports of $G(t)$ (and is $\perp$ if there is no a-port).

In the run on a term t such that $G(t)$ has n vertices, less than $(n+1)^{2k}$ states occur and these states have size $O(k.\log(n))$. In the transition table (Table 4), we let (∂, λ) denote a state that is not $Error$. Hence, if (∂, λ) is accessible, then we have $\partial(a) = \perp$ if and only if $\lambda(a) = 0$. We denote respectively by $\mathbf{0}$ and $\perp$ the constant mappings with values 0 and $\perp$. We take $\max\{\perp, n\} = n$ (for the transitions on $\oplus$). It is clear that the transitions can be computed in time $O(k.\log(n))$. Hence, we have a P-FA $\mathcal{A}_{Reg,k}$, as in Section 5.2.2.

Transitions	Conditions
$\varnothing \to (\bot, \emptyset)$	
$\mathbf{a} \to (\partial, \lambda)$	$\partial(x) =$ if $x = a$ then 0 else $\bot$, $\lambda(x) =$ if $x = a$ then 1 else 0.
$add_{a,b}[(\partial, \lambda)] \to (\partial', \lambda)$	If $\lambda(a) = 0$ or $\lambda(b) = 0$ then $\partial' = \partial$ else $\partial'(a) = \partial(a) + \lambda(b)$, $\partial'(b) = \partial(b) + \lambda(a)$ and $\partial'(x) = \partial(x)$ for $x \notin \{a, b\}$.
$relab_{a\to b}[(\partial, \lambda)] \to Error$	$\partial(a) \neq \partial(b), \partial(a) \neq \bot$ and $\partial(b) \neq \bot$.
$relab_{a\to b}[(\partial, \lambda)] \to (\partial, \lambda')$	The previous case does not apply, $\lambda'(a) = 0, \lambda'(b) = \lambda(b) + \lambda(a)$ and $\lambda'(x) = \lambda(x)$ for $x \notin \{a, b\}$.
$\oplus[(\partial_1, \lambda_1), (\partial_2, \lambda_2)] \to Error$	$\partial_1(a) \neq \partial_2(a)$ for some a such that $\partial_1(a) \neq \bot, \partial_2(a) \neq \bot$.
$\oplus[(\partial_1, \lambda_1), (\partial_2, \lambda_2)] \to (\partial, \pi)$	The previous case does not apply, $\partial(x) = \max\{\partial_1(a), \partial_2(a)\}$ and $\lambda(x) = \lambda_1(x) + \lambda_2(x)$ for all x.

Table 4 Transitions of $\mathcal{A}_{Reg}$

By taking the union of the automata $\mathcal{A}_{Reg,k}$, we get a P-FA $\mathcal{A}_{Reg}$. From it, we get by Proposition 39 a P-FA $\mathcal{A}_{Reg[X]}$. The nondeterminism degree of $pr(\mathcal{A}_{Reg[X]})$ is bounded by $O(n^{2k})$ where the exponent depends on the bound k on clique-width.

The property $\exists X.(Card_{\leq p}(X) \wedge Reg[X^c])$ expressing that the considered graph becomes regular if we remove at most p vertices is P-FA decidable by Corollary 34 (and the remark at the end of Section 4.1).

The function MaxCard$X.Reg[X]$ that defines the maximal cardinality of a regular induced subgraph of the considered graph is thus XP-FA computable (by Theorem 29). So is the property that the graph can be partioned into two regular subgraphs, expressed by $\exists X.(Reg[X] \wedge Reg[X^c])$ (the proof uses the same propositions). We get a time complexity $O(n^{8k+a})$ for some constant a, similar to the case of maximum directed cut.

5.2.4 Graph partition problems

Many partition problems consist in finding an s-tuple $(X_1, ..., X_s)$ satisfying :

$$Partition(X_1, ..., X_s) \wedge P_1(X_1) \wedge ... \wedge P_s(X_s),$$

where, $P_1, ..., P_s$ are properties of sets of vertices that can be MS expressible or defined by a FA. We may also wish to count the number of such partitions, or to find one that minimizes or maximizes the cardinality of X_1 or the number $Int(X_1, ..., X_s) := e(X_1) + ... + e(X_s)$ of *internal edges of* $(X_1, ..., X_s)$, *i.e.*, the

number of edges in some of the induced subgraphs $G[X_1], ..., G[X_s]$. We have discussed above the partitioning of a graph into two regular induced subgraphs. Vertex coloring problems are of this type with $P_i(X_i)$ being stability for each i, (*i.e.*, the induced subgraphs have no edge) and a fixed number s of allowed colors.

By Theorem 29, if the properties $P_i(X_i)$ are MS expressible, then the corresponding partition problem expressed by

$$\exists X_1, ..., X_s.Partition(X_1, ..., X_s) \wedge P_1(X_1) \wedge ... \wedge P_s(X_s),$$

is decidable by an FPT-FA. If they are decidable by an FPT-FA or an XP-FA, then it is decidable by an FPT-FA or an XP-FA, provided the conditions of Theorem 29 on degree of nondeterminism are satisfied. Proposition 13 shows that these conditions cannot be avoided. A particular case is the partition into planar subgraphs, i.e., each $P_i(X_i)$ is the MS expressible property that the induced subgraph $G[X_i]$ is planar. However, the implementation of the corresponding automaton seems rather hard. Let us go back to coloring problems.

Variations on vertex coloring

We let $Col(X_1, ..., X_s)$ abbreviate the MS property $Partition(X_1, ..., X_s) \wedge P_1(X_1) \wedge ... \wedge P_s(X_s)$ where each $P_i(X_i)$ expresses the stability of $G[X_i]$ (i.e., $G[X_i]$ has no edge). The function $\#(X_1, ..., X_s).Col(X_1, ..., X_s)$ counts the number of s-colorings of the given graph G. (This number is $\chi_G(s)$ where χ_G is the chromatic polynomial of G. See Section 6 for a use of this fact.) It is thus FPT-FA computable. Another number of possible interest is, if G is s-colorable:

$$\mathrm{MinCard}X.(\exists X_1, ..., X_{s-1}.Col(X, X_1, ..., X_{s-1}))$$

which is 0 if G is $(s-1)$-colorable. Otherwise, it indicates how close G is to be $(s-1)$-colorable. By Theorem 29, this number is computable by an FPT-FA.

There are other definitions of approximate s-colorings. One of them is the notion of (s, d)-*defective coloring*, expressed by the MS sentence:

$$\exists X_1, ..., X_s.(Partition(X_1, ..., X_s) \wedge Deg_{\leq d}[X_1] \wedge \ ... \ \wedge Deg_{\leq d}[X_s]).$$

For fixed s, we consider the problem of determining the smallest d for which this property holds. This number is at most $\lceil n/s \rceil$ for a graph with n vertices.

We recall from [BCID] that the property $Deg_{\leq d}[X]$ meaning that each vertex of X has degree at most d in $G[X]$ is decided by an FPT-FA whose number of states on a term in $T(F_k^{\mathrm{u}(1)})$ is $O(d^{2k})$. It follows that the existence of an (s, d)-defective coloring can be checked, for a graph with n vertices, in time $O(n \cdot d^{4s.k+a})$ for some constant a. By checking the existence of an (s, d)-defective coloring for successive values of d starting from 1, one can find the minimal value of d in time $O(n^{4s.k+a+1})$ hence $O(n^{8s.2^{rwd(G)}+a+1})$ which is similar to the time bound $O(n^{4s.2^{rwd(G)}+b})$ given in [GHO] (because for every undirected graph G, we have $cwd(G) \leq 2^{rwd(G)+1} - 1$).

Another possibility is to define

$$MD(X_1, ..., X_s) := (MaxDeg[X_1], ..., \ MaxDeg[X_s])$$

and to compute the set:

$$\mathrm{SetVal}(X_1, ..., X_s).MD(X_1, ..., X_s) \restriction Partition(X_1, ..., X_s),$$

from which the existence of an (s, d)-defective coloring can easily be determined. Since the automaton for $MaxDeg$ uses $O(n^{2k})$ states for a graph with n vertices defined by a term in $T(F_k)$, we get for $MD(X_1, ..., X_s)$ the bound $O(n^{2k.s})$ and, by Lemma 7 and Theorem 29, the bound $O(n^{4s.k+c})$ for some constant c, which is the same as above.

Graph partition problems with numerical constraints

Some partition problems consist in finding an s-tuple $(X_1, ..., X_s)$ satisfying:

$$Partition(X_1, ..., X_s) \wedge P_1(X_1) \wedge \ ... \ \wedge P_s(X_s) \wedge R(|X_1|, ..., |X_s|),$$

where, $P_1, ..., P_s$ are properties of sets and R is a **P**-computable arithmetic condition. An example is the notion of *equitable s-coloring* : $P_i(X_i)$ is stability for each i and condition $R(|X_1|, ..., |X_s|)$ expresses that any two numbers $|X_i|$ and $|X_j|$ differ by at most 1. The existence of an equitable 3-coloring is not trivial : it holds for the cycles but not for the graphs $K_{n,n}$ for large n. The existence of an equitable s-coloring is W[1]-hard for the parameter defined as s plus the tree-width ([Fell]), hence presumably not FPT for this parameter. Our constructions yield, for each integer s, an FPT-FA for checking the existence of an equitable s-coloring for clique-width as parameter. We obtain the answer from :

$$\mathrm{Sp}(X_1, ..., X_s).(Partition(X_1, ..., X_s) \wedge P_1(X_1) \wedge \ ... \ \wedge P_s(X_s))$$

that is computable by an FPT-FA.

Optimization problems.

We may want to minimize the number $Int(X_1, ..., X_s)$ over certain partitions $(X_1, ..., X_s)$ of the vertex set. This number is 0 for some partition if and only if the considered graph is s-colorable.

Let us examine the minimization over the partitions that satisfy $P_1(X_1) \wedge ... \wedge P_s(X_s) \wedge R(|X_1|, ..., |X_s|)$ where R is a **P**-computable arithmetic condition. We compute the set $S(P_1, ..., P_s)$ of $(s+1)$-tuples $(|X_1|, ..., |X_s|, Int(X_1, ..., X_s))$ such that $X_1, ..., X_s$ satisfy $Partition(X_1, ..., X_s) \wedge P_1(X_1) \wedge \ ... \ \wedge P_s(X_s)$, and from it, to select the minimal value of $Int(X_1, ..., X_s)$ such that $R(|X_1|, ..., |X_s|)$ holds. For a graph with n vertices, the number of such tuples is bounded by $(n+1)^s(n(n-1)/2+1)$, hence it is not hopeless to compute it by an FPT or XP algorithm. We sketch the construction of a FA doing that.

By an easy modification of the P-FA that computes $e(X)$ (cf. Section 5.2.2), we can construct one for $Int(X_1, ..., X_s)$ that uses less than $(n+1)^{s.k+2}$ states

(for a graph with n vertices defined by a term in $T(F_k)$). The P-FA computing $(|X_1|, ..., |X_s|)$ uses at most $(n+1)^s$ states. Let $\mathcal{A}_1, ..., \mathcal{A}_s$ be FA that compute respectively $P_1(X_1), ..., P_s(X_s)$. Assume that the nondeterminism degrees of $pr_s(\mathcal{A}_1)$, ..., $pr_s(\mathcal{A}_s)$ are respectively bounded by $b_1(n,k), ..., b_s(n,k)$. The nondeterminism degree of $pr_s(\mathcal{A})$ such that $\mathcal{A}$ is constructed from $\mathcal{A}_{Int}, \mathcal{A}_1, ..., \mathcal{A}_s$ by the methods of Section 4 is thus bounded by $n^{O(k.s)}.b_1(n,k) \cdot ... \cdot b_s(n,k)$. Hence, $S(P_1, ..., P_s)$ is XP-FA computable if $b_1(n,k), ..., b_s(n,k)$ are XP-bounded, and we get nothing better if $b_1(n,k), ..., b_s(n,k)$ are FPT- or P-bounded, because of the bound by $n^{O(k.s)}$ for $pr_s(\mathcal{A}_{Int})$.

Remarks : In general, the minimum value of a *weight* $w(X_1, ..., X_s)$ over all tuples $(X_1, ..., X_s)$ satisfying some condition necessitates to compute the set of all possible values $w(X_1, ..., X_s)$ and to select the minimal one. In some cases, this minimal value can be computed inductively. This is the case, to take an easy example, if $w(X_1, ..., X_s)$ is $1 \cdot |X_1| + 2 \cdot |X_2| + ... + s \cdot |X_s|$ because then, if $\overline{X} = (X_1, ..., X_s)$ and $\overline{Y} = (Y_1, ..., Y_s)$ are tuples of subsets of two disjoint sets, we have $w(X_1 \cup Y_1, ..., X_s \cup Y_s) = w(\overline{X}) + w(\overline{Y})$. It follows that the minimum value of $w(X_1, ..., X_s)$ can be computed as MinCard$X.P(X)$, by an easy modification of Definition 27(e).

5.2.5 Connected components

The empty graph is defined as connected and a connected component as nonempty. We have discussed in detail connectedness, denoted by *Conn*, in [BCID], and we come back to this important graph property. We show that the general constructions (cf. Theorem 29) can be improved in some cases. We consider undirected graphs.

Number and sizes of connected components.

We denote by $\kappa(G)$ the number of connected components of a graph G, by $\kappa(G,p)$ the number of those with p vertices, by $MinComp(G)$ (resp. $MaxComp(G)$) the minimum (resp. maximum) number of vertices of a connected component of G. We will compute these values by FA.

The MS formula $CC(X)$ defined as $Conn[X] \wedge X \neq \emptyset \wedge \neg Link(X, X^c)$ expresses that X is the vertex set of a connected component. Hence, $\kappa(G) = \#X.CC(X)(G)$, $\kappa(G,p) =$ MSp$X.CC(X)(G)(p)$, $MinComp(G)=$

MinCard$X.CC(X)(G)$ and $MaxComp(G)$=MaxCard$X.CC(X)$. These values can be computed by FPT-FA constructed by using Propositions 15, 16 and Theorem 29 in the following way: we let $\mathcal{A}$ decide $Link(X, X^c)$; the nondeterminism degree of $pr_1(\mathcal{A})$ on a term $t \in T(F_k^{\mathrm{u}})$ is bounded by 2^{2k}. The corresponding bound for the automaton that decides $Conn[X]$ is 2^{2^k} (cf. [BCID], Section 6). Then, we can use the above mentioned results. However, we can construct a more efficient (smaller) FA by a direct construction that modifies the FA $\mathcal{A}_{Conn}$ of [BCID] for connectedness.

First we consider the computation of $\kappa(G) = \#X.CC(X)(G)$ for G not empty. The formula $\neg Link(X, X^c)$ expresses that X is a (possibly empty) the vertex set of a union of connected components. Hence, $\#X.\neg Link(X, X^c)(G) = 2^{\kappa(G)}$. The construction of the FA computing $\#X.\neg Link(X, X^c)(G)$ is clearly easier than that for $\#X.CC(X)(G)$. This FA allows even to check if G is connected (this property is equivalent to $\#X.\neg Link(X, X^c)(G) = 2$). However, it is an FPT-FA, whereas we noted above that the automaton $\mathcal{A}_{Conn}$ of [BCID] that checks connectedness is a P-FA.

We can alternatively construct "directly" a *deterministic* FA $\mathcal{A}_\kappa$ to compute $\kappa(G)$. Its states are sets of pairs (L, m) such that $L \subseteq C := [k]$ and m is an integer. For every term t in $T(F_k^{\mathrm{u}})$ we want that:

$q_{\mathcal{A}_\kappa}(t) = \{(L, m) \mid L \subseteq C$, m is the number of connected components of $G(t)$ type $L\}$.

The transitions are easy to write; the output function is then defined by:

$Out_{\mathcal{A}_\kappa}(q) := \Sigma\{|L| \cdot m \mid (L, m) \in q\}$.

If $G(t)$ has n vertices, the size of a state on t is $O(n \cdot \log(k))$ and so, $\mathcal{A}_\kappa$ is a P-FA.

We now explain why this automaton is better than the one constructed by using Theorem 29. We recall that the states of $\mathcal{A}_{Conn}$ are such that :

$q_{\mathcal{A}_{Conn}}(t) = (L, L)$ with $L \subseteq C$, if $G(t)$ is not connected and all its connected components have type L,

otherwise,

$q_{\mathcal{A}_{Conn}}(t)$ is the set of types (nonempty subsets of C) of the connected components of $G(t)$.

The graph $G(t)$ is connected if and only if the state at the root is the empty set or $\{L\}$ for some nonempty set L. It is clear that $\mathcal{A}_{Conn}$ is a homomorphic image of $\mathcal{A}_\kappa$. Note that $\mathcal{A}_{Conn}$ yields more information than just the connectedness of $G(t)$: it computes also the set of types of the connected components. By Propositions 15 and 16, we get for property $CC(X)$ an automaton $\mathcal{A}_{CC(X)}$ such that, for every t and X :

$q_{\mathcal{A}_{CC(X)}}(t * X)$ is *Error* if there is an edge between X and its complement;

otherwise, X is a union of connected components of $G(t)$, and

$q_{\mathcal{A}_{CC(X)}}(t * X)$ records the set of types, let us call it $\sigma(X)$, of these connected components.

We simplify for clarity: the state $q_{\mathcal{A}_{CC(X)}}(t * X)$ contains more than the set $\sigma(X)$. This is why we write that "it records ..." and not "it is $\sigma(X)$". Then, let $\mathcal{A}'_\kappa$ be constructed from $\mathcal{A}_{CC(X)}$ by Theorem 29 (or Example 5) so as to compute $\kappa(G(t))$. For each term t, the state $q_{\mathcal{A}'_\kappa}(t)$ records, for each set σ of sets of labels, the number of sets X such that $\sigma(X) = \sigma$. This is more than needed: the state $q_{\mathcal{A}_\kappa}(t)$ records only information about the connected components of $G(t)$, not about all unions of connected components. If for example $G(t)$ is the graph:

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

then $q_{\mathcal{A}_\kappa}(t) = \{(ab, 2), (bc, 1), (cd, 1)\}$ whereas $q_{\mathcal{A}'_\kappa}(t)$ records $\{(ab, 3), (bc, 1), (abc, 3), (cd, 1), (bcd, 1), (abcd, 6)\}$.

Some tests

We have tested these automata on a connected graph $G = add_{a,b}(H)$ of clique-width 3 with 17 vertices such that H has 8 connected components, each with 2 or 3 vertices. The quickest automaton on a term defining G is $\mathcal{A}_\kappa$ (taking 0.0012 s), followed by $\mathcal{A}_{Conn}$ (0.0014 s) and $\mathcal{A}_{\#X.\neg Link(X,X^c)}$ (0.33 s) whereas $\mathcal{A}_{\#X.CC(X)}$ takes 39 s. It is interesting to note that the use of unbounded integers in $\mathcal{A}_\kappa$ makes the computation quicker than by $\mathcal{A}_{Conn}$ although $\mathcal{A}_{Conn}$ is finite on terms in $T(F_3)$.

Counting components by their size.

We now consider the computation of $\mathrm{MSp}X.CC(X)(G)$. First we observe that for each integer p, $\mathrm{MSp}X.CC(X)(G)(p)$ is computable from the values $\mathrm{MSp}X.\neg Link(X, X^c)(G)(p')$ for $p' \leq p$. (We made above a similar observation for the computation of $\kappa(G)$ from $\#X.\neg Link(X, X^c)(G)$). However, as in this previous case, we can construct a P-FA $\mathcal{B}$ derived from $\mathcal{A}_{Conn}$ (and generalizing the previous $\mathcal{A}_\kappa$) such that, for every term $t \in T(F^{\mathrm{u}}_\infty)$:

> its state $q_\mathcal{B}(t)$ is the set of triples (L, p, m) such that L is a nonempty set of port labels, $m, p \in \mathbb{N}_+$ and m is the number of connected component of $G(t)$ of type L having p vertices.

If $t \in T(F^{\mathrm{u}}_k)$ and $G(t)$ has n vertices, then $n = \Sigma_{(L,p,m)\in q_\mathcal{B}(t)} m.p$. Hence, $q_\mathcal{B}(t)$ can be described by a word of length $O(n. \log(k))$ (even if numbers are written in unary; the factor $\log(k)$ corresponds to the coding of labels). Here are the transitions :

$\varnothing \to \emptyset$,

$\mathbf{a} \to \{(\{a\}, 1, 1)\}$,

$\oplus[q, q'] \to q$" where q" is the set obtained by replacing iteratively in the multiset $q \sqcup q'$ (defined as the multiset union of two sets) any pair $\{(L, p, m), (L, p, m')\}$ by the unique triple $(L, p, m + m')$,

$relab_h[q] \to q'$: For each set L, we let $h(L)$ be the set obtained from L by replacing a by $h(a)$; then q' is the set of triples (L', p, m') such that :

$L' := h(L)$ for some $(L, p, m) \in q$,

$m' := \Sigma\{m \mid L' = h(L) \text{ and } (L, p, m) \in q\}$.

Finally, we describe the transitions $add_{a,b}[q] \rightarrow q'$. There are two cases.

Case 1 : a or b is not present in q or they are both present in q but in a unique triple of the form $(L, p, 1)$ (with $a, b \in L$). Then $q' := q$.

Case 2 : Case 1 does not apply. We let:

$q"$ be the set of triples in q that contain neither a nor b,

$L' := \cup\{L \mid (L, p, m) \in q - q"\}$ and

$p' := \Sigma\{p \cdot m \mid (L, p, m) \in q - q"\}$,

and finally $q' := q" \cup \{(L', p', 1)\}$.

We illustrate this case with an example:

$q = \{(\{a\}, 2, 1), (\{a\}, 1, 4), (\{a, b, c\}, 4, 1), (\{b, d\}, 3, 2), (\{c, d\}, 3, 4)\}$,

$q' = \{(\{a, b, c, d\}, 16, 1), (\{c, d\}, 3, 4)\}$,

where 16 is obtained as $2 \cdot 1 \ + 1 \cdot 4 + 4 \cdot 1 \ + 3 \cdot 2$ because the connected components of types $\{a\}$, $\{a, b, c\}$ and $\{b, d\}$ get fused into a unique one (of type $\{a, b, c, d\}$).

For computing MSp$X.CC(X)$ we take the output function:

$Out_{\mathcal{B}}(q) := \mu$ such that, $\mu(p) := \Sigma\{m \mid (L, p, m) \in q\}$ for $p \in \mathbb{N}_+$.

It is clear that the transitions and the output function can be computed in time poly($\| t \|$). Hence, $\mathcal{B}$ is a P-FA. From MSp$X.CC(X)(G)$ we get $\kappa(G, p)$ for each p.

Tools for separation problems.

For dealing with separation problems, it is useful to compare the cardinality of a set of vertices X to the number of connected components of $G - X$ $(:= G[V_G - X])$ and to the maximal cardinality of a connected component of $G - X$. For this purpose, we define for a graph G:

$\alpha(G) = \{(|X|, \kappa(G[X^c])) \mid X \subseteq V_G\}$,

$\beta(G) = \{(|X|, MaxCardCC(G[X^c])) \mid X \subseteq V_G\}$,

where $MaxCardCC(G[X^c])$ is the maximal cardinality of a connected component of $G[X^c]$. From $\alpha(G)$, one can determine, for given integers p and q, if there exists a set X of cardinality at most p whose deletion splits the graph in at least q connected components. Similarly, from $\beta(G)$ one can determine if

there is such a set X whose deletion splits the graph in connected components of size at most q.

Let $P(X,U)$ mean that U has one and only one vertex in each connected component of $G[X^c]$ and $Q(X,Y)$ mean that Y is the vertex set of a connected component of $G[X^c]$. These properties are MS expressible. Then $\alpha(G)$ is nothing but $\mathrm{Sp}(X,U).P(X,U)$ computed in G, and $\beta(G)$ can be obtained from $\mathrm{Sp}(X,Y).Q(X,Y)$. Hence, by Example 14(a), Propositions 15, 16 and Theorem 29, these two values are computable by FPT-FA.

5.3 Undecidability and intractability facts.

Let $\mathcal{R}$ be a family of **P**-computable numerical predicates (integers are given in binary notation) and let $\mathrm{MS}+\mathcal{R}$ denote the extension of monadic second-order logic with the additional atomic formulas $R(|X_1|,...,|X_s|)$ for R in $\mathcal{R}$. (We write $\mathrm{MS}+R$ for $\mathrm{MS}+\{R\}$). We have seen such formulas in Section 5.2.4. We wish to examine when the model-checking problem for $\mathrm{MS}+\mathcal{R}$ is FPT or XP. Actually we will mainly consider the case of words over finite alphabets, so the question reduces to whether it is polynomial-time solvable.

We first discuss undecidability results. There is no implication between (un)decidability results on the one hand and complexity results on the other, but decidability and FPT results for terms and graphs of bounded clique-width are proved with the same tools. Undecidability results are actually easier to prove and they help to foresee the difficulties regarding complexity.

We let $Eq(n,m)$ mean $n=m$; this binary relation defines a semi-linear set of pairs of integers.

Proposition 40 : The satisfiability problem for $\mathrm{MS}+Eq$ and $\mathrm{MS}+R$ where R is a unary predicate that is not ultimately periodic is not decidable over finite words.

That is, one cannot decide if a given sentence of $\mathrm{MS}+Eq$ or $\mathrm{MS}+R$ is satisfied by some word over a fixed finite alphabet.

Proof: The case of $\mathrm{MS}+Eq$ is proved in Proposition 7.60 of [CouEng] and the other one in [Bes].□

We now consider the model-checking problem.

Definition 41 : *Separating sets of integers.*

Let $R\subseteq\mathbb{N}$, $p,n\in\mathbb{N}$ such that $n>p$. We say that R *separates on* $[0,n]$ *the integers in* $[0,p]$ if, for every $x,y\in[0,p]$:

$x \neq y$ if and only if there exists $z \in \mathbb{N}$ such that $x + y + z \in [0, n]$ and,

either $x + z \in R$ and $y + z \notin R$ or $y + z \in R$ and $x + z \notin R$.

We say that an infinite set $R \subseteq \mathbb{N}$ is *separating* if there exists n_0 such that, for every $n > n_0$, R separates on $[0, n]$ the integers in $[0, \lfloor \log(n) \rfloor]$. The sets $\{n! \mid n \in \mathbb{N}\}$, $\{2^n \mid n \in \mathbb{N}\}$ and that of prime numbers are separating. An ultimately periodic set of integers is not separating. The set $D := \{a_n \mid n \in \mathbb{N}\}$ such that $a_0 = 1$, $a_{n+1} = 2^{a_n+3}$ is not ultimately periodic and not separating either. (To see this, observe that D does not separate $a_n + 1$ and $a_n + 2$ on $[0, a_{n+1} - 1]$.)

The notion of separation will be used as follows: if R separates on $[0, n]$ the integers in $[0, p]$, then for any two disjoint subsets X and Y of $[n]$, if $|X|, |Y| \leq p$, then:

$|X| = |Y|$ if and only if:
$[n] \models \forall Z.[Z \cap (X \cup Y) = \emptyset \Longrightarrow (R(|X \cup Z|) \Longleftrightarrow R(|Y \cup Z|))]$.

This means that the equipotence of small sets can be expressed in MS + R.

Proposition 42: Let R be a unary predicate that defines a separating subset of $\mathbb{N}$. If $\mathbf{P} \neq \mathbf{NP}$, the model-checking problem for MS + Eq and MS + R are not $\mathbf{P}$-solvable.

Proof: We first consider the case of MS + Eq. We use a method similar to that of Proposition 13. We denote by P any satisfiability problem expressed in conjunctive normal form with variables $x_1, ..., x_n$, all occurring in P. We let $w(P)$ be the word representing P with x_i written as x followed by the binary writing of i (with no leading 0). For example, if P is $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_4 \vee \neg x_5)$ then $w(P)$ is the word $(x1 \vee x10 \vee \neg x11) \wedge (x11 \vee \neg x100 \vee \neg x101)$ over the alphabet $A := \{(,), \vee, \wedge, \neg, x, 0, 1\}$.The factors of this word belonging to $\{0, 1\}^*$ have length at most $1 + \lfloor \log(n) \rfloor$. The word $w(P)$ is represented by the logical structure
$S(P) := \langle [|w(P)|], \leq, (lab_a)_{a \in A} \rangle$ such that $lab_a(i)$ holds if and only if i is an occurrence of a in $w(P)$.

There exists a formula $\varphi(U)$ of MS + Eq, written with $\leq$ and the unary relations lab_a for $a \in A$, such that an arbitrary satisfiability problem P as above has a solution if and only if $S(P) \models \exists U.\varphi(U)$. The set U defines a set of occurrences of x in $w(P)$ whose corresponding variable x_i has value $True$, and we require that, either all occurrences of any variable x_i or none of them has value $True$. This condition is expressed by a formula $\varphi_1(U)$ of MS + Eq where $Eq(|X|, |Y|)$ is only used for sets X and Y of consecutive occurrences of 0 and 1's. The formula $\varphi(U)$ is then taken of the form $\varphi_1(U) \wedge \varphi_2(U)$ where

$\varphi_2(U)$ is a first-order formula expressing that the truth values defined by a set U satisfying $\varphi_1(U)$ form a solution of P.

If the sentence $\exists U.\varphi(U)$ could be checked in words $w \in A^*$ in time poly($|w|$), then every satisfiability problem P could be checked in time poly($|w(P)|$) and we would have $\mathbf{P} = \mathbf{NP}$.

We now translate $\varphi_1(U)$ into a sentence $\varphi_3(U)$ of MS+R such that $\exists U.(\varphi_3(U) \wedge \varphi_2(U))$ is equivalent to $\exists U.\varphi(U)$ in every structure $S(P)$. It is clear that $2n < |w(P)|$ as all variables $x_1, ..., x_n$ occurring in P. Hence, any sequence of 0 and 1's in $w(P)$ has length bounded by $1 + \lfloor \log(n) \rfloor = \lfloor \log(2n) \rfloor \leq \lfloor \log(|w(P)|) \rfloor$. The equality tests $Eq(|X|, |Y|)$ of $\varphi_1(U)$ can be expressed in terms of R (that we assume separating) and except perhaps for finitely many problems P. Hence, the satisfiability of P is expressed in $S(P)$ by a sentence in MS + R, and so, the model-checking problem for MS + R is not **P**-solvable in polynomial time either. □

Questions 43 : (1) Can one replace in the previous proposition "R is separating" by "R is not ultimately periodic"? It might happen that the model-checking problem for MS + R where R is very sparse (like the above set D) is **P**-decidable on words.

(2) Let R be **P**-decidable. Is the model-checking problem for MS + R is **NP**-decidable on words? The same question can be raised for MS + Eq, or more generally, for MS + R if R is a semi-linear subset of $\mathbb{N}^k, k \geq 2$.

6 Implementation

AUTOGRAPH

The system AUTOGRAPH, written in LISP (and presented in the conference paper [BCID13a]) is intended for verifications of graph properties and computations of functions on graphs. Its main parts are as follows.

(1) A library of basic fly-automata over F_∞ :

for properties of sets : $X \subseteq Y$, $X = \emptyset$, $Sgl(X)$, $Card_{\leq p}(X)$, $Card_{p,q}(X)$, $Partition(X_1, ..., X_s)$ and the function $Card(X)$,

for the atomic formulas of MS logic over p-graphs : $edg(X, Y)$ and $lab_a(X)$,

for some important MS graph properties (cf. [BCID]) : stability, being a clique, $Link(X, Y)$, $Path(X, Y)$, connectedness, existence of directed or undirected cycles, degree at most d,

for some graph properties that are not MS expressible : regularity,

for functions on graphs : number of edges between two sets, maximum degree.

(2) A library of procedures that transform or compose fly-automata: these functions implement Propositions 10, 15, 16 and 39, and Theorems 17 and 29.

There is no parser for the formulas expressing properties and functions. The translation of these formulas into LISP programs that call the basic FA and the composition procedures is easily done by hand.

Some automata (in particular for cycles, regularity and other degree computations) are defined so as to work correctly on irredundant terms. A preprocessing can verify whether a term is irredundant, and transform it into an equivalent irredundant one if it is not. Whether input terms are good or not may affect the computation time, but not the correctness of the outputs.

Experiments

In [BCID], we have reported some experimental results concerning Petersen's and McGee's (classic) graphs. Using FA that count the number of satisfying assignments, we obtained that Petersen's graph has 12960 4-colorings. (This value can be verified as it is also given by the chromatic polynomial). We found also that McGee's graph has 57024 acyclic 3-colorings in less than 6 hours. (See [BCID] for definitions and details.)

Enumeration

AUTOGRAPH includes a method for enumerating (i.e., listing, not just counting) the sets $\mathrm{Sat}\overline{X}.P(\overline{X})$, by using an existing FA $\mathcal{A}$ for $P(\overline{X})$. A specific enumeration program is generated for each term (see [Dur]). Running it is also interesting for accelerating in some cases the verification that $\exists\overline{X}.P(\overline{X})$ is true, because the computation can stop as soon as the existence of some satisfying tuple $\overline{X}$ is confirmed. More precisely, the nondeterministic automaton $pr(\mathcal{A})$ is not run deterministically (cf. Definition 1(c)), but its potentially accepting runs are constructed by enumeration. This technique works for $\exists\overline{X}.P(\overline{X})$ but not for $\forall\overline{X}.P(\overline{X})$, $\#\overline{X}.P(\overline{X})$, $\mathrm{MSp}\overline{X}.P(\overline{X})$ etc... because these properties and functions are somehow based on a complete knowledge of $\mathrm{Sat}\overline{X}.P(\overline{X})$.

Using terms with shared subterms

Equal subterms of a "large" term t can be fused and t can be replaced by a *directed acyclic graph* (a *dag*). The construction from t of a dag where any two equal subterms are shared can be done in linear time by using the minimization algorithm of deterministic acyclic finite automata presented in [Rev]. Deterministic FA can run on such dags in a straightforward manner. We have tested that on 4-colorable graphs defined recursively by $G_{n+1} = t(G_n, G_n)$ where $t \in T(F_7, \{x, y\})$ (a term with two variables x and y denoting p-graphs; G_n has $10.2^n - 6$ vertices and $27.2^n - 23$ edges). We checked that these graphs are 4-colorable by using the term in $T(F_7)$ and the dag resulting from the recursive definition. The computation times are as follows:

n	term	dag
6	11 mn	1 mn, 6 s
9	88 mn	1 mn, 32 s
20		4 mn
28		40 mn
30		2 h, 26 mn

Do we need optimal terms?

Graphs are given by terms in $T(F_\infty)$ and no *a priori* bound on the clique-width must be given since all FA are over F_∞. As an input graph is given by a term t over F_k with $k \geq cwd(G)$, one may ask how important it is that k is close to $cwd(G)$. Every graph with n vertices is denoted by a term in $T(F_n)$ where each vertex has a distinct label and no relabelling is made. Such a term, if it is irredundant, has size $O(n^2.\log(n))$. Hence, as input to a P-FA, it yields a polynomial time computation. This not the case with an FPT- or XP-FA.

7 Conclusion

We have given logic based methods for constructing FPT and XP graph algorithms based on automata. Our constructions allow several types of optimizations:

- different logical expressions of a property can lead to different automata having different observed computation times,

- direct constructions of FA are sometimes better than the general ones resulting from Theorem 29.

We have noticed these latter facts in Sections 5.2.3 and 5.2.5. Can one identify general criteria for the possibility of such optimizations? In the same direction, we have cases where FPT-FA are easier to implement and practically more efficient than certain equivalent P-FA and similarly for XP-FA and FPT-FA. When does this hold?

About dags. Here is a question related to the possibility of using dags :

How can one transform a term in $T(F_k)$ into an equivalent one in $T(F_k)$, or in $T(F_{k'})$ for some k' not much larger than k, whose associated minimal dag (the one with a maximal sharing of subterms) has as few nodes as possible?

Edge quantifications. The logical representation of graphs used in this article does not allow edge set quantifications in MS formulas. MS formulas written with edge set quantifications are more expressive, and more functions based on them, such as $\#\overline{X}.\varphi(\overline{X})$, can be defined. An easy way to allow edge set quantifications is to replace a graph G by its *incidence graph* $I(G)$ (consisting of vertices, edges and the associated incidence relation). The tree-width of $I(G)$ is, up to 1, that of G, hence the clique-width of $I(G)$ is bounded in terms of the tree-width of G. MS formulas over $I(G)$ allow thus quantifications over sets of edges of G (details are in [CouEng]). So the constructions of FA presented in this article work for edge set quantifications (in the expression of properties and functions) and tree-width (but not clique-width) as parameter. However, more directed constructions based on terms representing tree-decompositions would be useful, due to the importance of tree-width. This issue is discussed [Cou12].

Acknowledgements : We thank C. Paul for useful comments.

8 References

[AHV] S. Abiteboul, R. Hull and V. Vianu, *Foundations of databases.* Addison-Wesley, 1995.

[ALS] S. Arnborg, J. Lagergren and D. Seese, Easy problems for tree-decomposable graphs. *J. Algorithms* **12** (1991) 308-340.

[Bes] A. Bès, Expansions of MSO by cardinality relations, *Preprint*, April 2013, submitted for publication.

[CouMos] B. Courcelle and M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.* **109** (993) 49-82.

[Cou96] B. Courcelle, The monadic second-order logic of graphs X: Linear orders, *Theoretical Computer Science*, 160 (1996) 87-143.

[Cou12] B. Courcelle, On the model-checking of monadic second-order formulas with edge set quantifications. *Discrete Applied Mathematics* **160** (2012) 866-887.

[BCID] B. Courcelle and I. Durand, Automata for the verification of monadic second-order graph properties, *J. Applied Logic*, **10** (2012) 368-409.

[BCID13] B. Courcelle and I. Durand, Model-checking by infinite fly-automata, in *Proceedings of 5-th Conference on Algebraic Informatics* (CAI), *Lec. Notes Comput. Sci.*, 2013, to appear.

[BCID13a] B. Courcelle and I. Durand, Infinite transducers on terms denoting graphs, in
Proceedings of the 6th European Lisp Symposium, Madrid, June 2013.

[CouEng] B. Courcelle and J. Engelfriet, *Graph structure and monadic second-order logic, a language theoretic approach*, Volume **138** of *Encyclopedia of mathematics and its application*, Cambridge University Press, June 2012.

[CMR] B. Courcelle, J. Makowsky and U. Rotics: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, **33** (2000) 125-150.

[Dur] I. Durand, Object enumeration, in *Proc. of 5th Europeal LISP Conference*, Zadar, Croatia, May 2012, pp. 43-57.

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

[EKS] V. Engelmann, S. Kreutzer and S. Siebertz, First-order and monadic second-order model-checking on ordered structures, in *Proc. of the 27th Symposium on Logic in Computer Science*, Dubrovnik, Croatia, 2012, pp. 275-284

[Fell] M. Fellows *et al.*, On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.* **209** (2011) 143-153.

[FG] J. Flum and M. Grohe, *Parameterized complexity theory*, Springer, 2006.

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

[GHO] R. Ganian, P. Hlinený and J. Obdrzálek, A unified approach to polynomial algorithms on graphs of bounded (bi-)rank-width. *Eur. J. Comb.* **34** (2013) 680-701.

[KanRao] M. Kanté and M. Rao, The Rank-Width of Edge-Coloured Graphs, *Theory of Computing Systems* (2012), in press.

[LKM] M. Lampis, G. Kaouri and V. Mitsou, On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization* **8** (2011) 129-138.

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

[Rei] K. Reinhardt, The Complexity of translating logic to finite automata, in *Automata, Logics, and Infinite Games: A Guide to Current Research*, E. Graedel et al. eds., *Lecture Notes in Computer Science* **2500** (2002) 231-238.

[Rev] D. Revuz, Minimisation of acyclic deterministic automata, *Theoret. Comput. Sci.* **92** (1992) 181-189.

[VeaBjo] M. Veanes, N. Bjorner, Symbolic automata: the toolkit, in *Proceedings of TACAS 2012, Lec. Notes Comput. Sci.* **7214** (2012) 472-477.

9 Appendix

9.1 Monadic second-order logic

Representing terms by logical structures.

We let F be a signature that can be countably infinite, with symbols of maximal arity $r \in \mathbb{N}$ (cf. Section 2.4)). If $t \in T(F)$, we define $S(t)$ as the relational structure $\langle Pos(t), son_t, (br_{i\,t})_{i\in[r]}, (lab_{f\,t})_{f\in F}\rangle$ where $son_t(x,y)$ holds if and only if y is a son of x, $br_{i\,t}(x)$ holds if and only if x is the i-th son of its father, and $lab_{f\,t}(x)$ holds if and only if x is an occurrence of f. This structure has an infinite number of components if F is infinite, but only finitely many of them are nonempty sets or relations. (Only $Pos(t), son_t, br_{i\,t}$ for $i \in [\rho(Sig(t))]$ and $lab_{f\,t}$ for $f \in \rho(Sig(t))$ are not empty). Hence, $S(t)$ can be encoded by a finite word over a fixed finite alphabet and one can decide if $S(t) \models \varphi(\overline{X})$ for given $\varphi(\overline{X})$ and $t * \overline{X} \in T(F^{(s)})$.

Consider now a property $P(\overline{X})$ expressed by a monadic second-order formula $\varphi(\overline{X})$ written with the (finitely many) relation symbols son, br_i, and lab_f for $f \in Sig(\varphi)$ where $Sig(\varphi)$ is the set of symbols of F that occur in φ (in relational symbols lab_f). Let L be the set of terms $t * \overline{X} \in T(F^{(s)})$ such that $S(t) \models \varphi(\overline{X})$. This set is recursive and a fly-automaton recognizing it can be constructed by induction on the structure of φ, see [BCID], Section 7.3. Since for every finite $H \subseteq F$, the number of states occurring in its runs on a term $t \in T(H)$ is finite, it is a linear FPT-FA (cf. Definition 6(b)).

Let H be any subset of F. Two terms $t, t' \in T(F^{(s)})$ are H-equivalent, written $t \sim_H t'$, if $Pos(t) = Pos(t')$ and every u in $Pos(t)$, is either an occurrence in t of $(f,w) \in H \times \{0,1\}^s$ and an occurrence in t' of the same symbol, or is an occurrence in t of $(f,w) \in (F-H) \times \{0,1\}^s$ and an occurrence in t' of (f',w) such that f' is not in H and has the same arity as f. If $t, t' \in T(F)$ and $t \sim_{Sig(\varphi)} t'$, then, for every s-tuple $\overline{X}$ of subsets of $Pos(t)(= Pos(t'))$:

$$S(t) \models \varphi(\overline{X}) \text{ if and only if } S(t') \models \varphi(\overline{X}).$$

This is clear because lab_f has no occurrence in φ if f is not in H, hence, φ cannot make any difference between $f \notin H$ and any symbol not in H of same arity. (The same holds if $\varphi(\overline{X})$ is a second-order formula). So, the set of terms $t * \overline{X} \in T(F^{(s)})$ such that $S(t) \models \varphi(\overline{X})$ is saturated for the equivalence $\simeq_{Sig(\varphi)}$. It is the closure under $\sim_{Sig(\varphi)}$ of a regular subset of $T(F^{(s)})$.

Representing graphs by logical structures.

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 $\mathbb{N}$ is identified with the structure $\langle V_G, edg_G, (lab_{a\,G})_{a \in \mathbb{N}} \rangle$ where $lab_{a\,G}$ is the set of a-ports of G. Since only finitely many sets $lab_{a\,G}$ are not empty, this structure can be encoded by a finite word over a fixed finite alphabet, as $S(t)$ discussed above.

We will mainly consider properties of (and functions on) graphs, and not of (and on) p-graphs (but the formal setting would allow that). However, as we have seen in [BCID] and in Section 5.2, port labels are important for the construction of FA that check properties of (and compute functions on) graphs. For checking a property of $G(t)$, we use an automaton whose states encode information about the p-graphs $G(t)/u$ for all positions u of t. This information depends on $\pi(G(t)/u)$, and a major concern is its size, because a large size implies that a long time is necessary for computing transitions.

Monadic second-order formulas

The basic syntax of monadic second-order formulas (MS formulas in short) uses set variables $X_1, ..., X_n, ...$ but no first-order variables. Formulas are written without universal quantifications and they can use set terms (cf. Section 2.3). These constraints yield no loss of generality (see, e.g., Chapter 5 of [CouEng]). To express properties of terms, we use the atomic formulas

$son(X_i, X_j)$ meaning that X_i and X_j denote singleton sets $\{x\}$ and $\{y\}$ such that y is a son of x,

$br_i(X_j)$ meaning that X_j denotes $\{x\}$ such that x is the i-th son of its father,

$lab_f(X_j)$ meaning that X_j denotes $\{x\}$ such that x is an occurrence of f.

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 $q \geq 2$).[16]

[16] 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 [CouEng] for situations where the distinction matters.

In order to express properties of a p-graph G, we use the atomic formulas $X_i \subseteq X_j$, $X_i = \emptyset$, $Sgl(X_i)$, $Card_{p,q}(X_i)$ as for terms together with :

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

$lab_a(X_i)$ meaning that X_j denotes $\{x\}$ such that x is an a-port.

Furthermore, it is convenient to require that the free variables of every formula of the form $\exists X_n.\varphi$ are among $X_1, ..., X_{n-1}$. (Every subformula of a well-written formula must satisfy this condition). This syntactic constraint yields no loss of generality (see Chapter 6 of [CouEng] for details) but it makes easier the construction of automata. In examples, we use set variables X, Y, universal quantifications, and other obvious notation to make formulas readable. A *first-order existential quantification* is a construction of the form $\exists X_n.(Sgl(X_n) \wedge \varphi(X_1, ..., X_n))$, also written $\exists x_n.\varphi(X_1, ..., X_{n-1}, \{x_n\})$ for readability. All quantifications of a first-order formula have this form. First-order order formulas may have free set variables and may be built with set terms. So, $\exists x_2.\varphi(X_1, X_1^c - \{x_2\})$ is a first-order formula if φ contains only first-order quantifications.

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

$\langle V_G, edg_G, (lab_{a\,G})_{a\in\mathbb{N}}\rangle \models \varphi(X_1, ..., X_n)$
if and only if $P(X_1, ..., X_n)$ is true in G.

9.2 Good and irredundant terms

We prove a technical result about terms in $T(F_\infty)$. We recall that F_∞ is the signature of clique-width operations (cf. Section 2.2 for detailed definitions).

Proposition 44 : (1) The set of good and irredundant terms in $T(F_\infty)$ is P-FA recognizable.
(2) There exists a polynomial-time algorithm that transforms every term in $T(F_\infty)$ into an equivalent term that is good and irredundant.

We recall that for two terms t and t', $t \approx t'$ means that they define isomorphic p-graphs. For t in $T(F_\infty)$, $\pi(t)$ is the set of port labels of $G(t)$, $\max \pi(t)$ is the maximal label in $\pi(t)$, $\mu(t)$ is the set of port labels that occur in t and $\max \mu(t)$ is the maximal one in $\mu(t)$ (we recall that port labels are positive integers).

Proof : (1) We have already observed after Definition 7 that the set of good terms is P-FA recognizable. By Proposition 8(2) of [BCID] the set of terms that

are not irredundant can be recognized by a nondeterministic FA whose states on a term t are pairs of port labels in $\mu(t)$ and nondeterminism degree is at most $|\mu(t)|^2$, hence poly($\|t\|$). By determinizing it and taking the complement, we get a P-FA $\mathcal{A}$ that recognizes the set of irredundant terms. By taking the product of $\mathcal{A}$ with the one recognizing good terms, we get a P-FA (by Proposition 15) that recognizes the good and irredundant terms.

(2) Proposition 8 of [BCID] gives, for each integer k, a linear-time algorithm that transforms a term t in $T(F_k)$ into an equivalent irredundant one t' such that $|t'| = |t|$ and $\|t'\| \leq \|t\|$. This algorithm attaches to each position of t a set of pairs of port labels from $\mu(t)$. These sets can be encoded in size $|\mu(t)|^2.\log(k) \leq$ poly($\|t\|$) and we obtain a unique polynomial-time algorithm taking as input a term in $T(F_\infty)$.

We can assume that the input term t is irredundant and we transform it into an equivalent one that is good and still irredundant. By induction on the structure of $t \in T(F_k)$,we will define:

a good term $\widehat{t} \in T(F_{k'})$ for some $k' \leq k$ such that $\pi(\widehat{t}) = [\max \pi(\widehat{t})]$
and a bijection $h_t : \pi(\widehat{t}) \to \pi(t)$ such that $t \approx relab_{h_t}(\widehat{t})$.

The inductive definition is shown in Table 5. Condition (1) states that ℓ is a bijection : $[|\pi(t)|] \to \pi(t)$ such that $\ell(i) = h_{t_1}(i)$ for $i \in [\max \pi(t_1)]$; (clearly, $|\pi(t)| \geq \max \pi(t_1)$).

t	$\widehat{t}$	h_t	Conditions
t	$\varnothing$	Id	$\pi(t) = \emptyset$ (i.e., $G(t) = \varnothing$)
$\mathbf{a}$	$\mathbf{1}$	$1 \to a$	
$t_1 \oplus t_2$	$\widehat{t_2}$	h_{t_2}	$\pi(t_1) = \emptyset$
$t_1 \oplus t_2$	$\widehat{t_1}$	h_{t_1}	$\pi(t_2) = \emptyset$
$t_1 \oplus t_2$	$\widehat{t_1} \oplus relab_{\ell^{-1} \circ h_{t_2}}(\widehat{t_2})$	ℓ	$\pi(t_1) \neq \emptyset, \pi(t_2) \neq \emptyset$ and (1)
$\overrightarrow{add}_{a,b}(t_1)$	$\widehat{t_1}$	h_{t_1}	$\{a,b\} \not\subseteq \pi(t_1)$
$\overrightarrow{add}_{a,b}(t_1)$	$\overrightarrow{add}_{h_{t_1}^{-1}(a), h_{t_1}^{-1}(b)}(\widehat{t_1})$	h_{t_1}	$\{a,b\} \subseteq \pi(t_1)$
$relab_h(t_1)$	$\widehat{t_1}$	$h \circ h_{t_1}$	

Table 5 : Inductive construction of $\widehat{t} \in T(F_{k'})$ and h_t.

It is clear that $\widehat{t}$ and h_t can be computed in polynomial time from t.

Claim 1 : $\widehat{t} \in T(F_{k'})$ for some $k' \leq k$, $\pi(\widehat{t}) = [\max \pi(\widehat{t})]$, h_t is a bijection $: \pi(\widehat{t}) \to \pi(t)$ and $t \approx relab_{h_t}(\widehat{t})$.

Proof : These facts are clear from the inductive construction and we have $k' = \max \mu(\widehat{t})$. $\square$

Claim 2 : $\widehat{t}$ is irredundant.

Proof : Because t is assumed irredundant. $\square$

Claim 3 : $\widehat{t}$ is good.

Proof : Let n be the number vertices of $G(t)$, assumed to have at least one edge. (The case of graphs without edges is easily treated separately). The inductive construction shows that, for each subterm t' of $\widehat{t}$, each label $\pi(t')$ labels some vertex of $G(t')$, hence $\max \mu(\widehat{t})$ is at most the number of vertices of $G(\widehat{t})$, equal to n.

Again by induction, we can see that $\oplus$ has $n-1$ occurrences in $\widehat{t}$ (because $\widehat{t}$ has no occurrence of $\varnothing$ and $G(t) \simeq G(t')$), and that the symbols $relab_h$ have at most $2n-1$ occurrences (one can of course delete those of the form $relab_{Id}$).

The number of operations $\overrightarrow{add_{a,b}}$ is at most $(n-1) \cdot (k'^2 - k')$ because $\widehat{t}$ is irredundant by Claim 2. It follows that $|\widehat{t}| \leq n+n-1+2n-1+(n-1)\cdot(k'^2-k') \leq (k'+1)^2.n+1$ as one checks easily (noting that $k' \geq 2$ because $G(\widehat{t})$ has edges). Hence $\widehat{t}$ is good. $\square$