Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Discrete Applied Mathematics 131 (2003) 129–150
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Query efficient implementation of graphs of bounded clique-width☆

B. Courcelle*, R. Vanicat

Laboratoire d'Informatique (LaBRI), CNRS, UMR 5800, Université Bordeaux I, 351 Cours de la Libération, 33405 Talence, France

Received 5 June 2000; received in revised form 22 February 2001; accepted 5 May 2002

Abstract

If $P(x_1,\ldots,x_k)$ is a graph property expressible in *monadic second-order logic*, where $x_1,\ldots,x_k$ denote vertices, if G is a graph with n vertices and of clique-width at most p where p is fixed, then we can associate with each vertex u of G a piece of information $I(u)$ of size $O(\log(n))$ such that, for all vertices $x_1,\ldots,x_k$ of G, one can decide whether $P(x_1,\ldots,x_k)$ holds in time $O(\log(n))$ by using only $I(x_1),\ldots,I(x_k)$. The preprocessing can be done in time $O(n\log(n))$.

One can do the same for any fixed *monadic second-order optimization function* (like distance) by using information of size $O(\log^2(n))$ for each vertex and computation time $O(\log^2(n))$. In this case preprocessing time is $O(-\log^2(n))$.

Clique-width is a complexity measure on graphs similar to tree-width, but more powerful since every set of graphs of bounded tree-width has bounded clique-width, but not conversely.

Similar results apply to graphs of tree-width at most w and to properties and functions expressed in the version of monadic second-order logic allowing quantifications on sets of edges.

1. Introduction

The starting point of this work is the notion of implicit representation of a graph, as considered by Spinrad [15]. The idea is to associate with each vertex of a

☆ This work has been supported by the European Training and Mobility in Research Network Get Grats.
* Corresponding author.
E-mail address: courcell@labri.fr (B. Courcelle).

0166-218X/03/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
PII: S0166-218X(02)00421-3

graph of size n (number of vertices) a bit sequence of length $O(\log(n))$ making it possible to determine whether two vertices are adjacent just by processing (by some fixed algorithm) the sequences attached to the two given vertices.

One may also wish to determine similarly the distance between two vertices, from (hopefully short) bit sequences attached to them. See Gavoille et al. [13].

In this paper we consider similarly, and more generally, properties of k-tuples of vertices (generalizing adjacency) formalized in monadic second-order (MS in short) logic, and optimization functions (generalizing distance) on graphs also formalized in MS logic.

Our proof technique is as follows. We first prove the results for a-balanced binary trees, i.e. trees with height at most $a\log(n)$ where n is the number of leaves and a is a constant. Our proof uses the basic result according to which MS logic is equivalent to finite-state automata on finite trees (more precisely on trees representing terms written with finitely many function and constant symbols).

For optimization functions, we use the algebraic methods of [8,14].

Certain graphs can be defined from trees by mappings from structures to structures formalized by MS logical formulas (they are called "MS transductions" in [5]). The MS properties of the graphs (or the MS optimization functions on them) can thus be formalized as MS properties of (or MS optimization functions on) the trees defining them. In this way, the results for trees can be transferred to graphs. This technique applies to graphs of tree-width at most k (for any fixed k), because the mapping from tree-decompositions (of width at most k) to the corresponding graphs is an MS-transduction. (See the survey paper in [7] in this journal about MS logic and algorithms.) It also applies to graphs of clique-width at most k, by the same technique.

Clique-width is a complexity measure on graphs somewhat similar to tree-width, but more powerful since every set of graphs of bounded tree-width has bounded clique-width but not conversely (cliques have clique-width 2 but unbounded tree-width). It is studied in Courcelle Olariu [9] and originates from [11].

In both case we need *a-balanced* tree decompositions (or *a-balanced* algebraic expressions in the case of clique-width). But every tree has a 3-balanced tree-decomposition of width at most 2 (this tree-decomposition is not optimal since trees have tree-width one). Our initial result about balanced trees extends first to arbitrary binary trees, and then to all graphs of clique-width bounded by a fixed value.

The paper is organized as follow. Section 2 recalls some definitions about MS logic and MS-transductions. These notions are familiar to the reader of the paper [7] in this journal. Section 3 deals with MS queries and MS-optimization functions on binary trees, either balanced or not. Section 4 presents the transformation of a binary tree into a 3-balanced one in a framework that is suitable for logical manipulations. Section 5 gives the application to graphs of bounded clique-width. Section 6 gives some comparison with others work, and an open question.

2. Definitions

2.1. Monadic second-order logic

We let R be a finite set of relation symbols, each of them, say r, being given with an *arity* $\rho(r)$ in $\mathbb{N}_+$. We denote by $\mathscr{S}(R)$ the set of finite *R-structures,* i.e., of tuples of the form $S=\langle D_S, (r_S)_{r\in R}\rangle$ where $r_S \subseteq D_S^{\rho(r)}$ for $r\in R$.

For two structures S and S' in $\mathscr{S}(R)$, we let $S\subseteq S'$ (read S is *included in* S') if $D_S \subseteq D_{S'}$, and $r_S \subseteq r_{S'}$ for each r in R.

We recall that *monadic second-order logic* (*MS* logic for short) is first-order logic augmented with (uppercase) variables denoting subsets of the domain of the considered structure, and new atomic formulas of the form $x\in X$ expressing the membership of x in a set X. We denote by $MS(R,\mathscr{X})$ the set of MS formulas over R with free variables in $\mathscr{X}$ (the set of all individual and set variables.)

A property of structures (or of elements and/or of sets of elements of a structure) is MS-*definable* if it can be expressed by an MS formula.

We denote by $S\equiv S'$ the existence of an isomorphism between two structures S and S'. If $L,L'\subseteq\mathscr{S}(R)$, we write $L\equiv L'$ if every structure of L is isomorphic to a structure in L' and vice-versa.

2.2. Graphs

A graph is identified with the $\{edg\}$-structure $G=\langle V_G, edg_G\rangle$ where V_G is the set of vertices and $edg_G\subseteq V_G\times V_G$ is a binary relation on V_G representing the set of edges.

The MS formula $\varphi(X)$ with free variable X:

$$\forall x,y(edg(x,y)\wedge x\in X\Rightarrow y\in X)$$

expresses that a set of vertices X is closed under the edge relation.

The MS formula $\psi(x,y)$ with free variables x and y:

$$\neg(x=y)\wedge\forall X(x\in X\wedge\varphi(X)\Rightarrow y\in X)$$

expresses x and y are distinct vertices and that there is a directed path from x to y.

Let f be a mapping from graphs to positive integers. An *f-annotation* of a graph G is a mapping $\alpha: V_G\to\{0,1\}^*$ such that, for every vertex v, the length of $|\alpha(v)|$ of the word $\alpha(v)$ is at most $f(G)$.

We say that a mapping $g:(\mathscr{P}(V_G))^k\to\mathbb{N}$ is computable from an annotation α of G if there exists an algorithm A by which one can compute $f(X_1,\ldots,X_k)$ from the sets $\{\alpha(v)/v\in X_i\}$, $i=1,\ldots,k$.

We will use these definitions in the following situation: g is fixed, and definable in MS logic (by letting $0=false$, $1=true$, we can handle logical properties as mappings to integers), G ranges over a class of graphs $\mathscr{C}$, an appropriate α can be computed efficiently from G where $f(G)=O(\log(|V_G|))$ or $O(\log(|V_G|)^2)$ and A is the same for all graphs in $\mathscr{C}$.

2.3. MS transduction

A *transduction* of structures is a multivalued mapping: $\mathscr{S}(R) \to \mathscr{S}(R')$, formally handled as a mapping $f : \mathscr{S}(R) \to \mathscr{P}(\mathscr{S}(R'))$, where $\mathscr{P}()$ denotes the power-set operation, such that $S \equiv S'$ implies $f(S) \equiv f(S')$. We say that f as above is *MS compatible* [10] if there exists a total recursive mapping $f^{\#}$: $MS(R', \varnothing) \to MS(R, \varnothing)$ such that $S \models f^{\#}(\varphi)$ iff $S' \models \varphi$ for some $S' \in f(S)$. We call $f^{\#}(\varphi)$ the *backwards translation* of φ relative to f.

We now consider transductions defined by MS formulas. *A parameterless MS-definable* transduction f: $\mathscr{S}(R) \to \mathscr{S}(R')$ is a partial function defined as follows, from $k \in \mathbb{N}_+$ and formulas α in $MS(R, \varnothing), \delta_1, \ldots, \delta_k$ in $MS(R, \{x\}), \theta_{r,i_1,\ldots,i_n}$ in $MS(R, \{x_1, \ldots, x_n\})$, for $r \in R, n = \rho(r), 1 \leqslant i_1, \ldots, i_n \leqslant k$:

(1) $f(S)$ is well defined iff $S \models \alpha$.
(2) Assuming $S \models \alpha$, then $f(S) = T$ where T is constructed as follows:
- $D_T = D_1 \times \{1\} \cup \cdots \cup D_k \times \{k\} \subseteq D_S \times \{1, \ldots, k\}$;
- each D_i is $\{x \in D_S / S \models \delta_i(x)\}$;
- each relation r of T is defined as the union of the sets of tuples of the form $\{((x_1, i_1), \ldots, (x_n, i_n))/S \models \theta_{r,i_1,\ldots,i_n}(x_1, \ldots, x_n)\}$, for all $i_1, \ldots, i_n \in \{1, \ldots, k\}$.

In order to specify k we say that the transduction is *k-copying* (and *noncopying* if $k = 1$).

Let f be a parameterless MS-definable transduction. It is MS-compatible [5,6]. One can even define a backwards translation $f^{\#}(\varphi)$ for MS formulas φ over R' with free variables. If φ has p free variables, then $f^{\#}(\varphi)$ has kp free variables. We refer the reader to [5,6] for details.

We now extend the previous definitions in order to define (by MS formulas) certain transductions: $\mathscr{S}(R) \to \mathscr{S}(R')$ that are not deterministic. Let $p_1, \ldots, p_n$ be n unary relation symbols, that we will call *parameters* ($p_1, \ldots, p_n \notin R \cup R'$). We let $\Pi_R : \mathscr{S}(R \cup \{p_1, \ldots, p_n\}) \to \mathscr{S}(R)$ be the mapping that "forgets" the relations $p_1, \ldots, p_n$. (It is actually a noncopying MS-definable transduction.)

A transduction $f : \mathscr{S}(R) \to \mathscr{S}(R')$ is *MS-definable* (we also say that it is an *MS-transduction*) if there exists an MS-definable subset L of $\mathscr{S}(R \cup \{p_1, \ldots, p_n\})$ and a parameterless MS-definable transduction $g : L \to \mathscr{S}(R')$ such that:

$$f(S) = \{g(S')/S' \in L, \Pi_R(S') = S\}.$$

It is *k-copying* or *noncopying* if g is so. The set $\{S \in \mathscr{S}(R)/f(S) \neq \varnothing\}$ is MS-definable: it is defined by the formula $\exists X_1, \ldots, X_n.\alpha[X_1/p_1, \ldots, X_n/p_n]$ where $\alpha \in MS(R \cup \{p_1, \ldots, p_n\}, \varnothing)$ defines L and X_i/p_i denotes the substitution of X_i for p_i in α. (We replace $p_i(x)$ by $x \in X_i$ for every i and x.)

An MS-transduction f as above is MS-compatible: for every $\varphi \in MS(R', \varnothing)$ one takes $f^{\#}(\varphi)$ equal to

$$\exists X_1, \ldots, X_n.(\alpha \wedge g^{\#}(\varphi))[X_1/p_1, \ldots, X_n/p_n].$$

If $f : \mathscr{S}(R) \to \mathscr{S}(R')$ and $g : \mathscr{S}(R') \to \mathscr{S}(R'')$ are two MS-transductions then $g \circ f$, the transduction $h : \mathscr{S}(R) \to \mathscr{S}(R'')$ defined by

$$h(S) = \cup\{g(S')/S' \in f(S),\ S \in \mathscr{S}(R)\}$$

is an MS-transduction. It is noncopying if f and g are so. See [5,6].

2.4. An example of MS transduction

Here is an example of an MS-transduction from graphs to graphs that associates with a directed graph $G = \langle V_G, edg_G\rangle$ the set of its connected components.

We will use one parameter p; we let

$$L = \{\langle V, edg, p\rangle / p \text{ has one and only one element}\}.$$

The set L is MS (and even first-order) definable.

We let $g : \mathscr{S}(\{edg, p\}) \to \mathscr{S}(\{edg\})$ be the mapping that associates with $\langle V, edg, p\rangle$ the graph $\langle V', edg'\rangle$ such that:

- V' is the set of vertices $x \in V$ that are linked in $\langle V, edg, p\rangle$ by an undirected [1] path to some vertices of p,
- $edg' = edg \cap (V' \times V')$.

The mapping g is a parameterless noncopying transduction defined by the following formulas (we let $k = 1$, δ denote δ_1 and θ_{edg} denote $\theta_{edg,1,1,1}$):

- α is *true*,
- $\theta_{edg}(x_1, x_2)$ is $edg(x_1, x_2)$,
- $\delta(x)$ is $p(x) \vee \exists y[p(y) \wedge \forall X(y \in X \wedge \psi(X) \Rightarrow x \in X)]$,

where $\psi(X)$ is the formula $\forall u, v(u \in X \wedge [edg(u, v) \vee edg(v, u)] \Rightarrow v \in X)$

Note that $\psi(X)$ is true iff the set X does not intersect properly a connected component of the considered graph.

Hence g associates with $\langle V, edg, p\rangle$ the union of the connected components of $G = \langle V, edg\rangle$ containing elements of p.

Hence $f(G) = \{g(G')/G' \in L, \Pi_{edg}(G') = G\}$, because G' is in L iff p is singleton.

3. Monadic second-order queries on trees

A *binary tree* is defined as a finite nonempty subset T of $\{0, 1\}^*$ such that for all u, v (if we denote the prefix order by $\leqslant$):

- $u \in T$ and $v \leqslant u$ implies $v \in T$;
- $u0 \in T$ iff $u1 \in T$.

[1] A path such that edges can be traversed in either direction.

The elements of T are called *nodes*. We call $u0$ the *first (left) successor of u*, $u1$ *its second (right) successor*. The empty word ϵ is the root of T. A maximal node (for $\leqslant$) is called a *leaf*.

We let $u \perp v$ mean that neither $u \leqslant v$ nor $v \leqslant u$.

If T is a binary tree and $u \in T$, we let $T/u = \{v \in \{0,1\}^*/uv \in T\}$ (if it is nonempty, it is a binary tree) and $T \setminus u = \{v \in T/v \leqslant u \text{ or } u \perp v\}$. Note that $u \in T \setminus u$ and that $T \setminus u$ is a binary tree.

The *height* of T is $ht(T) = \mathrm{Max}\{|u|/u \in T\} + 1$. (The tree reduced to ϵ has height 1.)

Let F be a finite set of binary function symbols, and C be a finite set of constants; we denote by $T(F,C)$ the set of (finite) terms written with $F \cup C$.

It is well known that occurrences of the symbols of $F \cup C$ in a term $t \in T(F,C)$ form a binary tree, denoted by $Dom(t)$ (also called the *domain* of t; see e.g [4]).

We denote by $Lab(t)$ the mapping from $Dom(t)$ to $F \cup C$ that associates with each node of $Dom(t)$ the symbol of which it is an occurrence.

For the term $t = f(a, f(a,b))$ we have $Dom(t) = \{\epsilon, 0, 1, 10, 11\}$, $Lab(t)(\epsilon) = Lab(t)(1) = f$, $Lab(t)(0) = Lab(t)(10) = a$, $Lab(t)(11) = b$.

We will rather represent a term in $T(F,C)$ by the relational structure $\|t\|$ defined as follows:

$$\|t\| = \langle N_t, suc_{1,t}(.,.), suc_{2,t}(.,.), (lab_{f,t}(.))_{f \in F \cup C} \rangle.$$

where $N_t = Dom(t)$, $suc_{1,t}(u,v)$ holds iff $v = u0$, $suc_{2,t}(u,v)$ holds iff $v = u1$, $lab_{f,t}(u)$ holds iff $Lab(t)(u) = f$.

We denote by $root_t$ the node of $Dom(t)$ corresponding to the root (the word ϵ). By a *leaf of t* we mean a leaf of the underlying tree $Dom(t)$. We denote by L_t the set of leaves of t.

We will consider MS formulas expressing properties of terms t (represented by the corresponding structures $\|t\|$) and of subsets of their sets of leaves. Since MS formulas do not distinguish between isomorphic structures, $\|t\|$ can be replaced by any structure isomorphic to it.

If $t \in T(F,C)$ and $Z_1, \ldots, Z_k$ are subsets of L_t, we let $t[Z_1, \ldots, Z_k] \in T(F, C \times \{0,1\}^k)$ be defined from t as follows: for each leaf v of t, we replace its label c by $(c, \varepsilon_1, \ldots, \varepsilon_k)$ where $\varepsilon_i = 0$ if $v \notin Z_i$ and $\varepsilon_i = 1$ if $v \in Z_i$.

Let $\varphi(X_1, \ldots, X_k)$ be an MS formula over the relation symbols suc_1, suc_2, lab_f for $f \in F \cup C$, with free variables among $X_1, \ldots, X_k$. One can construct a finite tree automaton[2] A_φ over $F \cup (C \times \{0,1\}^k)$ such that for every $t \in T(F,C)$, for every $Z_1, \ldots, Z_k \subseteq L_t$ we have:

$$t[Z_1, \ldots, Z_k] \in L(A_\varphi) (\subseteq T(F, C \times \{0,1\}^k))$$
$$\text{iff } (\|t\|, Z_1, \ldots, Z_k) \models \varphi.$$

[2] All tree automata will be frontier-to-root deterministic and complete. The set of trees accepted by an automaton A is denoted by $L(A)$.

Proposition 1. *Let F be a finite set of binary function symbols, let C be a finite set of constants, and $P(X_1,\dots,X_k)$ be an MS-property of sets of leaves $X_1,\dots,X_k$. For every $t\in T(F,C)$ one can compute in time $O(ht(t).|t|)$ an $O(ht(t))$-annotation of t from which one can determine $P(Z_1,\dots,Z_k)$ in time $O(ht(t).|\bar{Z}|)$ where $|\bar{Z}| = |Z_1| + \cdots + |Z_k| + 1$*

Of course the algorithm that checks whether $P(Z_1,\dots,Z_k)$ holds true is constructible from F, C and an MS formula φ which specifies P.

Proof. Let v be a leaf of t, labeled by $c\in C$. Let $(f_1,i_1)(f_2,i_2)\dots(f_n,i_n)c$ be its *access word* representing the access path to v from the root of t; we let $f_1,\dots,f_n$ be the function symbols on this path, and $i_1,i_2,\dots,i_n\in\{1,2\}$ represent the left–right branching. (We have $i_p = j_p + 1$ for each $p = 1,\dots,n$ if v is the word $j_1j_2,\dots,j_n\in\{0,1\}^*$)

For an example, if $t = f(g(a,h(g(b,c),c)),a)$ and v is the (unique) leaf with label b, then its access word is $(f,1)(g,2)(h,1)(g,1)b$.

Let A_φ be the finite tree automaton constructed from φ (by the classical result of Doner, Thatcher, Wright; see for instance Thomas [16]) such that for every $t\in T(F,C)$ for all $X_1,\dots,X_k\subseteq L_t$, we have $t[X_1,\dots,X_k]\in L(A_\varphi)$ iff $\|t\|\models\varphi(X_1,\dots,X_k)$.

Let Q be its set of states.

Let us run A_φ over $t_0 = t[\emptyset,\dots,\emptyset]$. Each node u of t (i.e. of t_0) gets one and only one state $q(u)\in Q$.

Let v be a leaf of t, with access word: $(f_1,i_1)(f_2,i_2)\dots(f_n,i_n)c$. The corresponding nodes of the path are, say, $u_1,u_2,\dots,u_n,v$ where u_1 is the root, u_2 its left or right son depending on i_1, etc.

We now let

$$I(v) = (f_1,i_1)q(s_{3-i_1}(u_1))(f_2,i_2)q(s_{3-i_2}(u_2))\dots(f_n,i_n)q(s_{3-i_n}(u_n))cq(v),$$

where we denote by $s_1(u)$ (resp. $s_2(u)$) the left (resp. the right) son of u. The states $q(s_{3-i_1}(u_1)),\dots,q(s_{3-i_n}(u_n))$ are the states of nodes at distance one to the path from the root of t to v.

Then $I(v)$ is the information of size $O(ht(t))$ we attach to v (note that $n\leqslant ht(t)$). Assume A_φ computed once for all, this step takes $O(|t|)$ for running A_φ on t and time $O(ht(t).|L_t|) = O(ht(t).|t|)$ for constructing all words $I(v)$ for $v\in L_t$.

We claim that for every $Z_1,\dots,Z_k\subseteq L_t$ we can determine whether $\|t\|\models\varphi(Z_1,\dots,Z_k)$ by processing the words $I(v)$ for all $v\in Z_1\cup\cdots\cup Z_k$. Let us consider first the simple case where $Z_1 = \{v_1\}$, $Z_2 = \{v_2\}$, $Z_i = \emptyset$, for $i > 2$.

In order to know whether $t' = t[Z_1,\dots,Z_k]\in L(A_\varphi)$ we have to run A_φ on t'. However, the run of A_φ on t' differs from the one on t only on the paths between the root and v_1, and the root and v_2. We can compute the states of the run of A_φ on these paths of t' as follows:

1. Looking at the longest common prefix of $I(v_1)$ and $I(v_2)$ we can determine the node w at which the paths differ.

2. Using the table of A_φ and the states $q(s_{3-i_j}(u_j))$ for u_j between v_1 and w and between v_2 and w, (these states are available from $I(v_1)$ and $I(v_2)$) we can determine the states of the run of A_φ on t' between v_1 and w, and v_2 and w (w excluded).
3. We obtain the state of w from the two states at the sons of w.
4. By going on the path between w and the root of t' as in step 2, we obtain the state at the root of t' for the run of A_φ.

And whence the conclusion.

All this can be done in time $O(n_1 + n_2)$ where n_i is the distance of v_i to the root.

In the general case we have to do a similar computation with the (at most) $|\bar{Z}|$ paths from the leaves in $Z_1 \cup \cdots \cup Z_k$ to the root instead of with the two paths from v_1 and v_2 to the root.

In time $O(ht(t).|\bar{Z}|)$ we can build the tree of these paths and traverse it bottom up in order to compute the state at $root_{t'}$ for the run of A_φ on t'. □

For the next proposition, we let again $\varphi(X_1,\ldots,X_k)$ be an MS formula with free variables among $\{X_1,\ldots,X_k\}$ and we let for fixed $t \in T(F,C)$ and $Z_i \subseteq L_t$:

$$\mathrm{Min}(\varphi)(Z_1,\ldots,Z_{k-1}) = \mathrm{Min}\,\{|Z_k|/Z_k \subseteq L_t, \|t\| \models \varphi(Z_1,\ldots,Z_k)\}.$$

We call $\mathrm{Min}(\varphi)$ an *MS-optimization function.*

Proposition 2. *Let F be a finite set of binary function symbols, let C be a finite set of constants, let $f(Z_1,\ldots,Z_{k-1})$ be an MS-optimization function on trees in $T(F,C)$, where $Z_1,\ldots,Z_{k-1}$ are sets of leaves. For every $t \in T(F,C)$ one can compute in time $O(ht(t).\log(|t|).|t|)$ an $O(ht(t).\log(|t|))$-annotation of t from which we can compute $f(Z_1,\ldots,Z_{k-1})$ in time $O(ht(t).|\bar{Z}|.\log(|t|))$ where $Z_1,\ldots,Z_{k-1} \subseteq L_t$ (and as in Proposition 1 we let $|\bar{Z}| = |Z_1| + \cdots + |Z_{k-1}| + 1$).*

(We denote by $|t|$ the size of t, which is $2|L_t| - 1$ since $Dom(t)$ is a binary tree).

Proof. We first consider the case when $k = 1$, hence where $\mathrm{Min}(\varphi)$ has no argument.

We can assume that $\|t\| \models \exists X.\varphi(X)$, this can be checked by Proposition 1, hence we can consider that $\mathrm{Min}(\varphi)$ is well defined. We can also take $\mathrm{Min}(\varphi) = \infty$ if t has no set of leaves satisfying φ.

We construct A_φ.

For every $Z \subseteq L_t$ the run of A_φ on the subtree of $t[Z]$ issued from a node u of t depends only on $L_{t/u} \cap Z$, i.e. on the set of leaves in Z that are below u.

For every node u of t we let $M_t(u,q)$ be the minimum cardinality of $Z \subseteq L_{t/u}$ such that $q(u) = q$ for the run of A_φ on $t[Z]$. We take it equal to ∞ if no such Z do exist.

Hence $\mathrm{Min}(\varphi) = \mathrm{Min}\,\{M_t(root_t,q)/q$ is an accepting state of $A_\varphi\}$.

The values $M_t(u,q)$ can be computed bottom-up in the tree t as follows.

If u is a leaf then $M_t(u,q)$ is 0, 1 or ∞ and this value can be obtained from the table of A_φ and the label c of u.

If u is an internal node labeled by f we have

$$M_t(u,q) = \mathrm{Min}\left\{\begin{array}{l} M_t(s_1(u),\ q_1) + M_t(s_2(u), q_2)/ \\ \qquad\qquad \text{we have in } A_\varphi \text{ a transition} \\ \qquad\qquad (q_1, q_2) \to q \text{ for symbol } f \end{array}\right\}. \tag{1}$$

Hence, we can compute $\mathrm{Min}(\varphi)$ by a bottom-up traversal of t.

For each node u we compute the total mapping: $Q \to \mathbb{N} \cup \{\infty\}$ (with finite domain), that associates $M_t(u,q)$ with each $q \in Q$. The size of the piece of information associated to each node is thus at most $|Q|.\log(|t/u|) \leqslant |Q|.\log(|t|)$. The time is thus $O(|t|.\log(|t|))$ for fixed formula φ and automaton A_φ. We can consider this computation as that of an infinite tree automaton with set of states $Q \times (\mathbb{N} \cup \{\infty|\})$, and transition table defined by (1).

We consider now the general case of $\varphi(X_1, \ldots, X_k)$. We let A_φ be the automaton associated with this formula. We let $t \in T(F, C)$. For every $Z_1, \ldots, Z_{k-1}$, every node u of t we let $M_{t,\bar{Z}}(u,q)$ (where $\bar{Z} = (Z_1, \ldots, Z_{k-1})$) be the minimum cardinality of $Z \subseteq L_{t/u}$ such that the run of A_φ on $t[Z_1, \ldots, Z_{k-1}, Z]$ yields q as state of u. We take ∞ as value if no such Z do exist.

The values $M_{t,\bar{Z}}(u,q)$ satisfy the same computation rules as in (1) above. The dependence on $\bar{Z}$ is handled as in Proposition 1.

We first compute the values $M_{t,\bar{\emptyset}}(u,q)$ for all u and q (where $\bar{\emptyset} = (\emptyset, \emptyset, \ldots, \emptyset)$). We store them along access words of leaves as in Proposition 1.

For $\bar{Z} = (Z_1, \ldots, Z_{k-1})$, $Z_1 \cup \cdots \cup Z_{k-1} \neq \emptyset$, the values of $M_{t,\bar{Z}}(u,q)$ differ from those of $M_{t,\bar{\emptyset}}(u,q)$ only at the nodes u on the paths between the root and the leaves in $Z_1 \cup \cdots \cup Z_{k-1}$. These new values can be obtain from the pieces of information attached to the leaves and the transition table of A_φ. Since they have total size at most $O(ht(t).\log(|t|))$ (instead of $O(ht(t))$ as in Proposition 1) we use a global time $O(ht(t).|\bar{Z}|.\log(|t|))$. □

Remark. We will be interested in *classes of balanced trees*, i.e., in classes of trees t such that $ht(t) = O(\log(|t|))$. In these cases we get for Proposition 1 processing time $O(|t|.\log(|t|))$ and query time $O(\log(|t|)|\bar{Z}|)$. For Proposition 2 we get processing time $O(|t|.\log^2(|t|))$ and query time $O(\log^2(|t|)|\bar{Z}|)$.

In many cases we will consider queries such that each set Z_i consists of a single vertex. In these cases, the factor $|\bar{Z}|$ disappears in the evaluation of query times.

An example is the distance function on a tree or a graph, that is $\mathrm{Min}(\varphi)$ where $\varphi(Z_1, Z_2, Z_3)$ means: "Z_1 is $\{x_1\}$, $Z_2 = \{x_2\}$, Z_3 induces a connected subgraph containing x_1 and x_2, for some x_1 in Z_1 and x_2 in Z_2".

4. Balanced trees and terms

It is known (see [1]) that every tree which has tree-width at most 1, also has a tree decomposition of width 2 (which is thus not optimal regarding width) and of height $O(\log(n))$, where n is the number of nodes of the tree.

We give a proof of this result which fits our purpose to deal with MS formulas. We first give a few lemmas for binary trees, and then we apply them to terms. We recall that binary trees are defined as subsets T of $\{0,1\}^*$.

From the definitions we have, if $u \in T$ and u is not a leaf:

$$T = T \setminus u \cup u.T/u0 \cup u.T/u1 \quad \text{hence}$$

$$|T| = |T \setminus u| + |T/u0| + |T/u1| \tag{2}$$

since the sets $T \setminus u$, $u.T/u0$ and $u.T/u1$ are disjoint. We recall that $u \in T \setminus u$ and that $T \setminus u$ is T minus the strict descendants of u.

Lemma 1. *Let T be a tree with n nodes, $n \geqslant 3$, $n = 2p+1$, let $l \in T$ be an internal node (i.e., a node that is not a leaf).*

(i) *l has a successor l' such that $|T/l'| \leqslant p$;*
(ii) *If $|T \setminus l| > p$, there is a node s such that $\epsilon \leqslant s < s' \leqslant l$, where s' is the successor of s on the path from the root ϵ to l, $|T \setminus s| \leqslant p$ and $|T/s'| \leqslant p+1$*

Proof. (i) We have by (2) $2p+1 = |T \setminus l| + |T/l0| + |T/l1|$. Hence at least one of $|T/l0|$ and $|T/l1|$ is less than or equal to p.

(ii) We let s be the longest prefix of l such that $|T \setminus s| \leqslant p$. Hence $s < l$, and we let s' be the successor of s such that $s' \leqslant l$ and s'' be the other successor. We have $|T \setminus s'| > p$ and $2p+1 = |T \setminus s'| + |T/s'| - 1$. Hence $|T/s'| \leqslant 2p+1+1-(p+1) = p+1$. □

Definition 1. Let T be a binary tree as in Lemma 1 and l be one of its internal nodes then:

1. If $|T \setminus l| \leqslant p$ we let $Cut(T,l) = l$, $Main(T,l) = T/l'$ where l' is a successor (say the first one) such that $|T/l'| \leqslant p$; we let $Rem(T,l) = T/l''$ where l'' is the other successor and $Ctx(T,l) = T \setminus l$.
2. If $|T \setminus l| > p$ we let $Cut(T,l)$ be the node s defined by (ii) of Lemma 1, $Main(T,l) = T/s'$, $Rem(T,l) = T/s''$ where s'' is the other successor of s and $Ctx(T,l) = T \setminus s$.

In both cases we say that l is 'good' if $|Rem(T,l)| \leqslant p+1$.

Remark. In both cases, $|Ctx(T,l)| \leqslant p$, $|Main(T,l)| \leqslant p+1$ and $Cut(T,l)$ is an internal node.

Lemma 2. *Let T be a binary tree with at least 3 nodes. It has a good node l.*

Proof. One can find a longest $s \in T$ such that $|T \setminus s| \leqslant p$. Let s' and s'' its two successors: $|T \setminus s'| > p$ hence $|T/s'| \leqslant p+1$ (as in case (ii) of Lemma 1) and $|T/s''| \leqslant p+1$ with same argument.

Hence we take $l = s$ as 'good' node. It satisfies case 1 of Definition 1. □

We now apply these lemmas to terms in $T(F,C)$ where we recall that all symbols in F are binary.

The idea is as follows. Let $t \in T(F,C)$ and $T = Dom(t)$ be the corresponding binary tree. Let $s \in Dom(t)$ be an internal node. We can write $t = t_1[t_2/u]$ where $T \setminus s = Dom(t_1)$, $T/s = Dom(t_2)$, $t_1 \in T(F, C \cup \{u\})$ where u is a special nullary symbol with s as unique occurrence, and $t_1[t_2/u]$ denotes the result of the substitution of t_2 for u in t_1. To complete the definition of t_1 and t_2 we let

$$Lab(t_1)(x) = Lab(t)(x) \quad \text{if } x \in T \setminus s,\ x \neq s,$$

$$Lab(t_1)(s) = u,$$

$$Lab(t_2)(x) = Lab(t)(sx) \quad \text{if } x \in T/s.$$

(We recall that s, x are words.)

We define a *context* as a term in $T(F, C \cup \{u\})$ with one and only one occurrence of u. The trivial context is u denoted by Id (we think of Id as denoting the identity function). We denote by $Ctxt(F,C)$ the set of these terms.

On contexts we have the following operations: if $c \in Ctxt(F,C)$, $t \in T(F,C)$ and $f \in F$ then $f(c,t) \in Ctxt(F,C)$ and $f(t,c) \in Ctxt(F,C)$.

We let also $c \circ t \in T(F,C)$ denote $c[t/u]$ for $c \in Ctxt(F,C)$ and $t \in T(F,C)$ and $c \circ c' \in Ctxt(F,C)$ denote $c[c'/u]$ for $c, c' \in Ctxt(F,C)$.

We can thus build terms with the operations of $F \cup \{\circ\}$ and the constants of $C \cup \{Id\}$.

Definition 2. We let $eval : T(F \cup \{\circ\}, C \cup Id) \to T(F,C) \cup Ctxt(F,C)$ be the partial function defined as follows:

- $eval(Id) = u \in Ctxt(F,C)$,
- if $c \in C$ then $eval(c) = c \in T(F,C)$,
- if $f \in F$, sp_1 and sp_2 belong to $T(F \cup \{\circ\}, C \cup \{Id\})$ then $eval(f(sp_1, sp_2)) = f(eval(sp_1), eval(sp_2))$. If both $eval(sp_1)$ and $eval(sp_2)$ are in $T(F,C)$ then $eval(f(sp_1, sp_2))$ is in $T(F,C)$, if one and only one of them is in $Ctxt(T,F)$ then $eval(f(sp_1, sp_2))$ is in $Ctxt(F,C)$, otherwise it is not defined (sp_1 and sp_2 are then both contexts),
- if sp_1 and sp_2 belong to $T(F \cup \{\circ\}, C \cup \{Id\})$, $eval(sp_1) = c$ is in $Ctxt(F,C)$ and $eval(sp_2) = t$ then $eval(sp_1 \circ sp_2)$ is $c[t/u]$. The object $c[t/u]$ is in $Ctxt(T,F)$ if $t \in Ctxt(T,F)$, it is in $T(F,C)$ if $t \in T(F,C)$.

We let $SPE(F,C)$ denote the set of *special terms*, i.e. of those for which $eval$ is well defined. See the Fig. 1 for an example of a special term, and Fig. 2 for its evaluation.

By the *size* of a term, we mean the number of nodes of its underlying tree.

Theorem 1. *For every term $t \in T(F,C)$ or every context $c \in Ctxt(F,C)$ of size $n = 2p+1$, one can build a term $t' \in SPE(F,C)$ that is equivalent to t or to c, i.e. such that $eval(t') = t$ (or c), and of height at most $3\log(p) + 3$ for the case of a term and $3\log(p) + 5$ for the case of a context.*[3] *This term can be constructed in time* $O(|t| \log(|t|))$.

[3] Logarithms are always in base 2.

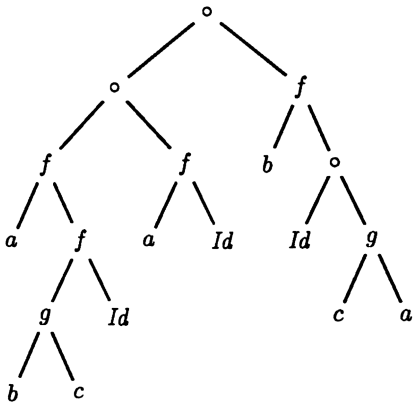

Fig. 1. A special term.

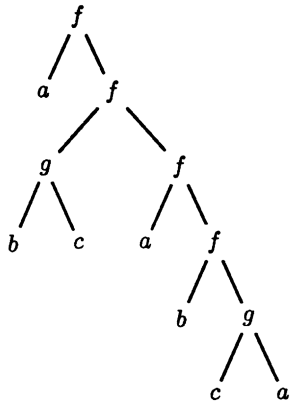

Fig. 2. Evaluation of the special term of Fig. 1.

Proof. We will make the proof by induction on p simultaneously for terms and contexts.

If $p \leqslant 1$, the result holds trivially.

Let t be a term or a context of size $2p+1$, we let l be:

- a good node of t if t is a term (we use Lemma 2 to find it)
- the node which is the occurrence of u if t is a context.

Let s be the node $Cut(t,l)$, then $t = c' \circ t'$ where $Dom(c') = T \setminus s$; then $t' = f(t_1, t_2)$ for some $f \in F$ where either

$$Dom(t_1) = Main(t,l) \quad \text{and}$$

$$Dom(t_2) = Rem(t,l)$$

or vice-versa (see Definition 1).

By using induction we have special terms $\overline{c'}$, $\overline{t_1}$, $\overline{t_2}$, respectively, equivalent to c, t_1 and t_2 satisfying the conditions on heights of the theorem.

We let then

$$\bar{t} = \overline{c'} \circ f(\overline{t_1}, \overline{t_2})$$

and we claim it satisfies the desired conditions.

It is clear that $eval(\bar{t}) = t$.

If t is a term, then by the definition we are in the good case, so $|c'|$, $|t_1|$ and $|t_2|$ are at most $p+1$. So by induction we have:

- $ht(\overline{c'}) \leqslant 3\log(p/2) + 5 = 3\log(p) + 2$,
- $ht(\overline{t_1}) \leqslant 3\log(p/2) + 3 = 3\log(p)$,
- $ht(\overline{t_2}) \leqslant 3\log(p/2) + 3 = 3\log(p)$.

So $ht(\bar{t}) \leqslant \mathrm{Max}(3\log(p) + 2, 3\log(p) + 1) + 1$

We have

$$ht(\bar{t}) \leqslant 3\log(p) + 3.$$

If t is a context, then we only know that $|c'| \leqslant p$, $|t_1| \leqslant p+1$ and $t_2 \leqslant 2p+1$ (or vice-versa). So by induction we have:

- $ht(\overline{c'}) \leqslant 3\log(p/2) + 5 = 3\log(p) + 2$,
- $ht(t_1) \leqslant 3\log(p/2) + 3 = 3\log(p)$,
- $ht(t_2) \leqslant 3\log(p) + 3$.

So $ht(\bar{t}) \leqslant \mathrm{Max}(3\log(p) + 2, 3\log(p) + 1, 3\log(p) + 3 + 1) + 1 = 3\log(p) + 5$.

For the time complexity of the construction, we note that we can find the cut node in time $O(|t|)$, and that we can apply again the same algorithm to three trees of size at most $|t|/2$ (or if we are not in the good case, then the bigger subtree will be in the good case, and we will have to find another cut node (in time $O(|t|)$) and we will have to apply the same algorithm to 5 subtrees of size at most $|t/2|$). So the total time is in $O(|t|\log(|t|))$. □

We now consider the mapping *eval* from the logical point of view. We let $SPE(F,C)_{\mathrm{term}}$ be the set of special terms that evaluate to terms (and not to contexts).

Theorem 2. *The mapping* $eval : SPE(F,C)_{\mathrm{term}} \to T(F,C)$ *is an MS-transduction.*

Proof. We recall that a term t is handled as a relational structure $\|t\|$. Hence we need prove that:

(1) $\{\|t\| / t \in SPE(F,C)_{\mathrm{term}}\}$ is MS-definable
(2) the mapping $\|t\| \mapsto \|eval(t)\|$ (for $t \in SPE(F,C)_{\mathrm{term}}$) is an MS-transduction.

We first consider the second point.

We will express this mapping as the composition of two MS-transductions:

$$\|t\| \mapsto S(t)$$

and

$$S(t) \mapsto \|eval(t)\|,$$

where $S(t)$ is an intermediate structure, intuitively a tree with certain edges labeled by ϵ that we will contract in the final step to get $\|eval(t)\|$.

Let $t \in SPE(F, C)_{\text{term}}$. Since $t \in T(F', C')$ where $F' = F \cup \{\circ\}$ and $C' = C \cup \{Id\}$ the structure representing it is

$$\|t\| = \langle N_t, suc_{1,t}, suc_{2,t}, (lab_{f,t})_{f \in F' \cup C'} \rangle.$$

We let $C_t \subseteq N_t$ be the set of nodes w such that $eval(t/w)$ is a context. Since t evaluates to a term, the root is not in C_t. We let $S(t)$ be the following structure: $S(t) = \langle D, suc'_{1,t}, suc'_{2,t}, \epsilon, (lab_{f,t})_{f \in F \cup C} \rangle$ where ϵ is a new binary relation and we let

$$D = N_t \cup (C_t \times \{i\}) \quad (i \text{ is a label, not an integer}),$$

$$suc'_{1,t} = suc_{1,t} \cap \left(\left(\bigcup_{f \in F} lab_{f,t} \right) \times N_t \right),$$

$$suc'_{2,t} = suc_{2,t} \cap \left(\left(\bigcup_{f \in F} lab_{f,t} \right) \times N_t \right).$$

Intuitively, each node w in C_t is "duplicated" and its copy is (w, i) (where i means: "input to the context t/w").

Note that relations $lab_{\circ,t}$ and $lab_{Id,t}$ do not exist any longer in $S(t)$.

The binary relation ϵ is defined as follows: $\epsilon(x, y)$ holds iff

(i) either $suc_{1,t}(x, y)$ and $lab_{\circ,t}(x)$ hold,
(ii) or $x = (x', i)$ for some $x' \in C_t$ and $z \in N_t$ such that $lab_{\circ,t}(z)$, $suc_{1,t}(z, x')$ and $suc_{2,t}(z, y)$ hold,
(iii) or $x = (x', i)$, $y = (y', i)$ for some $x', y' \in C_t$ such that $lab_{o,t}(y')$ and $suc_{2,t}(y', x')$,
(iv) or $x = (x', i)$, $y = (y', i)$ for some $x', y' \in C_t$ and $j \in \{1, 2\}$ such that $lab_{f,t}(y')$ and $suc_{j,t}(y', x')$ hold,
(v) or $y' = (x, i)$ and $x \in C_t$ and $lab_{Id,t}(x)$ holds.

All cases of the definition of the relation ϵ are illustrated in Figs. 3 and 4. Here are a few comments about those figures:

- the edges marked 1, 2 correspond to pairs satisfying suc_1, suc_2, respectively, in Fig. 3 and satisfying suc'_1, suc'_2 in Fig. 4,
- a new element (x, i) for $x \in C_t$ is represented by $\textcircled{i}$, close to the element x,
- the labels $\circ$ and Id do not exist any longer in Fig. 4.

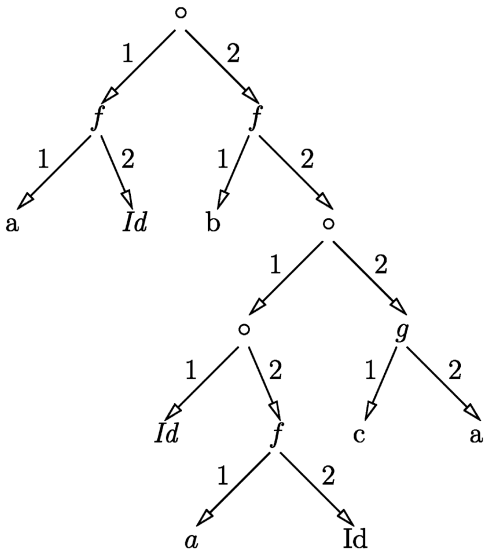

Fig. 3. The structure $\|t\|$ for a special term t.

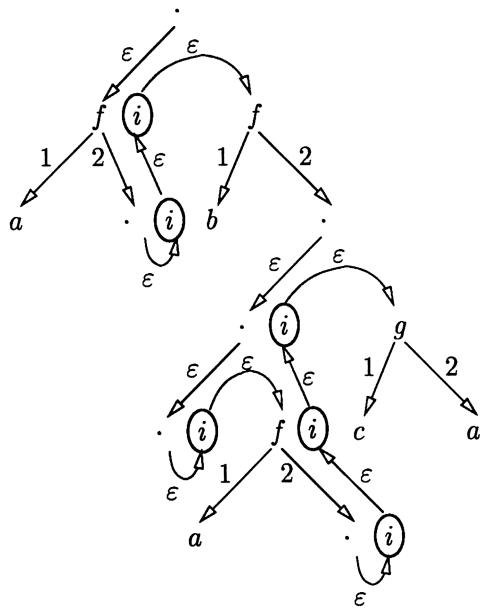

Fig. 4. The structure $S(t)$ for t as in Fig. 3.

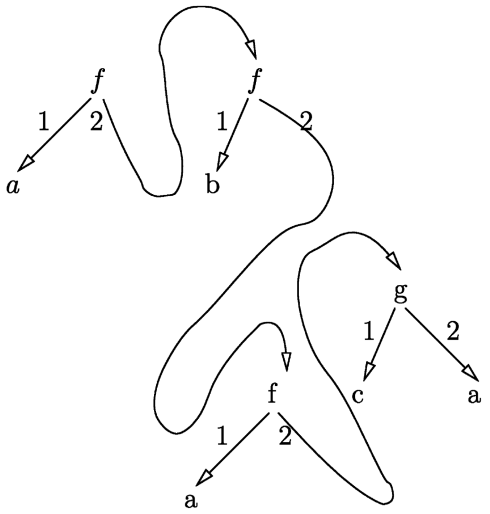

Fig. 5. The final term for Fig. 3.

In order to prove that the transformation that maps $\|t\|$ to $S(t)$ is an MS-transduction, it is enough to check that the set C_t is MS-definable in $\|t\|$.

This is true since a node w of N_t belongs to C_t iff there is in $\|t\|$ a path from w to a leaf labeled by Id, that does not use any edge $x \xrightarrow{1} y$ where x is labeled by $\circ$.

Since the notion of a path is MS-definable this property of w can be expressed by an MS-formula.

Furthermore, if $t \in T(F \cup \{\circ\}, C \cup \{Id\})$ then $t \in SPE(F, C)_{\text{term}}$ iff

(i) the root is not in C_t,
(ii) there is no node labeled by $f \in F$ with its two successors both in C_t.

From the above remark this property is MS-expressible.

Hence the mapping $\|t\| \mapsto S(t)$ for $t \in SPE(F, C)_{\text{term}}$ is an MS-transduction. It remains to prove that the mapping $S(t) \mapsto \|eval(t)\|$ is also a MS-transduction.

But this mapping is nothing but the simultaneous contraction of all ϵ-edges, which is an MS-transduction. The two successor relations are suc'_1 and suc'_2 (see Fig. 5). □

5. Efficient implementation of graph queries

We consider loop-free simple undirected graphs. Graphs with loops and directed edges can be considered similarly.

We first review the graph expressions upon which the complexity measure named *clique-width* is defined.

Let $\mathscr{C}$ be a countable set of labels,

Definitions.

- A *labeled* graph is a triple $G=(V,E,\gamma)$ where V is the set of vertices, E is the edge relation and γ is a mapping from V to $\mathscr{C}$. A labeling function γ is denoted by $\gamma(G)$ whenever the revelant graph G must be specified. We say that G is *C-labeled* if C is a finite subset of $\mathscr{C}$ and $\gamma(G)(V)\subseteq C$.
- If $G_1=(V_1,E_1,\gamma_1)$ and $G_2=(V_2,E_2,\gamma_2)$ are two C-labeled graphs with $V_1\cap V_2=\emptyset$ then $G=G_1\oplus G_2$ is the labeled graph $G=\langle V,E,\gamma\rangle$ defined by:
  - $V=V_1\cup V_2$,
  - $E=E_1\cup E_2$,
  - if $v\in V_1$ then $\gamma(v)=\gamma_1(v)$,
  - if $v\in V_2$ then $\gamma(v)=\gamma_2(v)$.
- If $G=(V,E,\gamma)$ is a C-labeled graph, $p,q\in C$ then $add_{p,q}(G)$ is the C-labeled graph $G'=(V,E',\gamma)$ with
$$E'=E\cup\{\{v,w\}/v,w\in V, v\neq w, \gamma(v)=p \quad \text{and} \quad \gamma(w)=q\}.$$
- If $G=(V,E,\gamma)$ is a C-labeled graph, $p,q\in C$ then $ren_{p\to q}(G)$ is the C-labeled graph $G'=(V,E,\gamma')$ with for all v, $\gamma'(v)=\gamma(v)$ if $\gamma(v)\neq p$, and $\gamma'(v)=q$ if $\gamma(v)=p$.
- if $p\in C$ we denote c_p the C-labeled graph $G=(\{v\},\emptyset,\gamma)$ such that $\gamma(v)=p$ (for any object v).
- A graph (V,E) is considered as labeled with all vertices having the same label.

We now define the graphs of clique-width at most k as those that can be constructed by successive applications of the operations $\oplus, add_{p,q}, ren_{p\to q}$ where p,q are in a set C of labels of size at most k.

We denote by $val(t)$ the graph which is the result of the evaluation of a term t over the operations and constants $\oplus, add_{p,q}, ren_{p\to q}, c_p$. It is not a loss of generality to assume that $C\subseteq\{1,\ldots,k\}$. We denote by F_k the set $\{\oplus, add_{p,q}, ren_{p\to q}/p,q\in\{1,\ldots,k\}, p\neq q\}$ and by C_k the set $\{c_1,c_2,\ldots,c_k\}$. Hence $cwd(G)\leqslant k$ iff $G=val(t)$ for some $t\in T(F_k,C_k)$.

Basic properties of graphs of clique-width at most k can be found in Courcelle and Olariu [9].

We cannot apply directly the results of the previous section to terms in $T(F_k,C_k)$ because some of the operations (namely $add_{p,q}, ren_{p\to q}$) are unary.

However we can replace the set $\{\oplus, add_{p,q}, ren_{p\to q}, c_p/p,q\in C, p\neq q\}$ by a finite set of binary operations and constants, built by combining these operations and constants, and that define the same graphs.

We do that as follows, for each fixed finite set C.

For every composition λ of unary operations (i.e. operations of the forms $add_{p,q}$ and $ren_{p\to q}$) we introduce a new binary operation say $\oplus_\lambda$ defined by:

$$G_1\oplus_\lambda G_2=\lambda(G_1\oplus G_2).$$

Although there are infinitely many sequences λ, the graphs $\lambda(c_p)$ have a single vertex, hence each of them is actually of the form c_q for some q.

Furthermore it follows from Lemma 3 below that there are only finitely many different operations $\oplus_\lambda$. Hence, the operations defining graphs of clique-width at most k can be replaced by finitely many binary operations defining exactly the same graphs (together with the constants c_p).

Lemma 3. *There are finitely many operations $\oplus_\lambda$ for each fixed set C of labels.*

Proof. Let $\bar{C} = \{\bar{c}/c \in C\}$. Let G_C be the labeled graph with $\bar{C}$ as set of vertices, no edge and each vertex $\bar{c}$ has label c. Let λ and λ' be two compositions of unary operations such that $\lambda(G_C) = \lambda'(G_C)$. Then for every graph G, $\lambda(G) = \lambda'(G)$, by the following fact:

Fact 1. *Let $H=(V,E,\gamma)$ be a C-labeled graph, λ be a composition of unary operations, let $H' = (V,E',\gamma') = \lambda(H)$ and $(\bar{C},E_\lambda,\gamma_\lambda) = \lambda(G_C)$ then*:

- $\gamma'(x) = c$ *iff* $\gamma(x) = c'$ *and* $\gamma_\lambda(\overline{c'}) = c$,
- $(x,y) \in E'$ *iff* $(x,y) \in E$ *or* $\gamma(x) = c$, $\gamma(y) = c'$ *and* $(\bar{c},\overline{c'}) \in E_\lambda$, *for some* $c,c' \in C$, $c \neq c'$.

Proof. This is true if $\lambda = Id$, and one can easily see that if this is true for a particular λ then it is true for all compositions of the forms $add_{p,q} \circ \lambda$ and $ren_{p\to q} \circ \lambda$ □

As there are only finitely many labeled graphs of size $|C|$, there are only finitely many different functions obtained by composition of unary operations among $add_{p,q}$ and $ren_{p\to q}$, for $p,q \in C$.

We let F'_k denote the finite set of operations $\oplus_\lambda$ defined as above for $C = C_k$. Hence $val(T(F_k,C_k)) = val(T(F'_k,C_k))$. □

5.1. Main theorems

Theorem 3. *Let $P(X_1,\ldots,X_n)$ be an MS-property of graphs. For every graph G of cwd at most k, given as $val(t)$ for some $t \in T(F_k,C_k)$ one can compute in time* $O(s.\log(s))$ *where $s = |V_G|$ an* $O(\log(s))$*-annotation of G from which one can compute in time* $O(\log(s).|\bar{Z}|)$ *whether $P(Z_1,\ldots,Z_n)$ holds in G (where $|\bar{Z}| = |Z_1| + \cdots + |Z_n| + 1$)*

Let P be described by the MS-formula $\varphi(Z_1,\ldots,Z_k)$. Of course as in Proposition 1 of Section 3 the algorithm that checks whether $\|G\| \models \varphi(Z_1,\ldots,Z_k)$ is the same for all graphs of clique-width k and is constructible from φ and k

Proof. From the given term $t \in T(F_k,C_k)$ such that $val(t) = G$ we construct $t' \in T(F'_k,C_k)$ such that $val(t') = val(t)$.

Note that all operations in F'_k are binary.

We can construct a balanced term t'' in $SPE(F'_k,C_k)$ such that $eval(t'') = t'$ by applying Theorem 1 to t'.

Since *val* and *eval* are MS-transductions, so is their composition [6]. Hence every graph G of clique-width at most k is the image of a balanced term t'' belonging to $SPE(F'_k, C'_k)$ under an MS-transduction.

Let us now consider φ. By backwards translation the property it defines can be expressed by an MS-formula φ'' on t''. Then we build labels for the leaves by using Proposition 1. As each leaf of t'' corresponds to a vertex of G and the height of t'' is $O(\log(s))$, the property holds directly. □

As in Section 3 there is another result:

Theorem 4. *Let $f(X_1,\ldots,X_{n-1})$ be an MS-optimization function on graphs. For every graph G of clique-width at most k, given as $val(t)$ for some $t \in T(F_k, C_k)$, one can compute in time $O(\log^2(s).s)$ where $s = |V_G|$ an $O(\log^2(s))$-annotation of G from which one can compute $f(Z_1,\ldots,Z_{n-1})$ in time $O(|\bar{Z}|.\log^2(s))$ (where as in Theorem 3 we let $|\bar{Z}| = |Z_1| + \cdots + |Z_{n-1}| + 1$).*

The proof is similar to the previous one, but uses Proposition 2.

5.2. Monadic second-order logic with edge set quantifications

In an MS formula expressing a property of a graph G defined as a structure $\langle V_G, edg_G\rangle$, the set variables necessarily denote sets of vertices.

One might need to use variables denoting sets of edges, and this is actually necessary for expressing by an MS formula Hamiltonicity (to take only this example).

We can also represent an undirected graph G by the structure $I(G) = \langle V_G \cup E_G, inc_G\rangle$ where E_G is the set of edges and inc_G is the binary relation such that $inc_G(e,x)$ holds iff e is an edge and x is one of its vertices. Hence $I(G)$ can be seen as a directed bipartite graph sometimes called the *incidence graph* of G.

Monadic second-order logic is more expressive if the considered graphs G are represented by $I(G)$ rather than as in the previous section. We will note this variant by MS_2.

However, the mapping that associates $I(val(t))$ with $t \in T(F_k, C_k)$ is not an MS transduction. Hence Theorems 3 and 4 do not extend to MS_2 formulas. However, if we consider graphs of bounded tree-width (as opposed to bounded clique-width) we obtain the following result (graphs are simple and undirected).

Theorem 5. *Let $P(X_1,\ldots,X_n)$ be a graph property, and $f(X_1,\ldots,X_n)$ an MS-optimization function both expressed by MS-formulas over the relation symbol inc, and let $k \in \mathbb{N}$.*

1. *For each graph G of tree-width at most k and of size s, one can compute in time $O(s.\log(s))$ an $O(\log(s))$-annotation of the graph $I(G)$ from which one can determine $P(Z_1,\ldots,Z_n)$ in time $O(\log(s).|\bar{Z}|)$ where $|\bar{Z}| = |Z_1| + \cdots + |Z_n| + 1$.*
2. *One can compute in time $O(s.\log^2(s))$ an $O(\log^2(s))$-annotation of the graph $I(G)$ from which one can compute $f(Z_1,\ldots,Z_{n-1})$ in time $O(|\bar{Z}|.\log^2(n))$*

Proof (sketch). The proof is an immediate adaptation of those of Theorems 3 and 4 based on the following two facts:

First we need not be given a tree-decomposition of G (or an algebraic term representing it) because it can be constructed in time $O(s)$ (for fixed k) where s is the number of vertices of G (we also recall that the number of edges is $O(s)$ for G of tree-width at most k, since we consider simple graphs) see [1].

Second, graph operations can be defined (see [6]) so that the mapping from a term t to the structure $I(G)$ where G is the value of this term is an MS transduction. Hence the proofs of Theorems 3 and 4 go through. We omit further details. □

5.3. The special case of distance

The distance between two vertices is an MS definable optimization function, hence the last theorem is applicable to it. However, in some cases, an information of size $O(\log(n))$ attached to each vertex is enough.

Here is one of these cases.

We fix an integer k. We consider graphs G built from "basic" connected graphs H with at most k vertices, each of them having two distinguished distinct vertices called the *begin* and the *end* vertices denoted, respectively, by $b(H)$ and $e(H)$. Then G is a connected graph consisting of m such graphs $H_1,\ldots,H_m$, where $e(H_i)=b(H_{i+1})$ for all $i=1,\ldots,m-1$ and these graphs are otherwise disjoint. Hence, G is a sort of chain of connected graphs, each of size at most k. We let $b(G)=b(H_1)$.

The key observation is that for a vertex x in H_i (where i is minimal so) and y in H_j, $i<j$, $x\neq y$ then $d(x,y)=d(y,b(G))-d(e(H_i),b(G))+d(x,e(H_i))$.

Hence, it is enough to store, for each vertex x, its distance to $b(G)$, the distance $d(e(H_i),b(G))$, and its distances to the at most k vertices in the element H of the chain to which it belongs. This can be done using for each vertex a piece of information of size $O(\log(n))$ where n is the number of vertices of G.

A new result by Gavoille [12] shows that for graphs of pathwidth 2 (such graphs have bounded clique-width) we may need space $O(\log(n)^2)$, hence that even for distance, Theorem 4 is optimal. The graphs used to prove this fact are built as follows: the vertices are $1,2,\ldots,n$, $1',\ldots,p'$, and the edges link i and $i+1$, j' and $j'+1$, and i and $f(i)'$ (where f is a partial, strictly increasing mapping from $\{1,2,\ldots,n\}$ to $\{1,2,\ldots,p\}$) for $i\in\{1,\ldots,n\}$ and $j\in\{1,\ldots,p\}$. The proof transfers a similar result for trees obtained in [13].

6. Conclusion

We first compare our results with those of Hagerup [14] who extends previous results by Chaudhuri and Zaroliagis [2,3] concerning querying distances in graphs. As we do in this paper, he considers MS queries.

His work differs from our in three respects. First, he considers graphs of bounded tree-width, whereas we consider graphs of bounded clique-width, so that we cover more classes of graphs.

Second, he builds a "global data structure", as we do in Propositions 1 and 2 of Section 3, whereas our ultimate goal is to distribute information on all vertices. We do not see how his global information can be distributed on vertices, as we do in Section 5.1.

This may explain, at least partially, the fact that his data structure is more efficient than ours: its initialization time is in $O(nI_k(n))$ where $k \geqslant 1$ is an arbitrary constant, and I_k are functions define as:

- $I_0(n) = \lceil n/2 \rceil$ for $n \in \mathbb{N}$
- $I_k(n) = \text{Min}\{i \in \mathbb{N} / I_{k-1}^{(i)}(n) = 1\}$ for $k \geqslant 1$ and $n \geqslant 0$

where $I_{k-1}^{(i)}$ denote the i-fold iteration of I_{k-1}.

Third, his method is implementable by a parallel algorithm to build the data structure, and makes it possible to handle modifications of the considered graph.

By Theorems 1, 2 and Lemma 3 we have the following:

For every k there exists a finite set of labels Λ_k and an MS-transduction τ from binary trees with nodes labeled in Λ_k (denoted $T(\Lambda_k)$) to graphs such that:

(1) $\tau(T(\Lambda_k))$ is the set of graphs of $cwd \leqslant k$,
(2) every graph of $cwd \leqslant k$ is the value under τ of a 3-balanced tree in $T(\Lambda_k)$.

It follows from the proof of Theorem 5.8 of [6] that there exists a function $f(k)$ such that every graph G of $cwd \leqslant k$ is $val(t)$ for some $t \in T(F'_{f(k)}, C_{f(k)})$ (cf. Section 5 for F'_k) such that t is 3-balanced.

Question. Find a construction of t not using the logical tools of [6] that could give a good (if not optimal) function f.

References

[1] H.L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, Theoret. Comput. Sci. 209 (1998) 1–45.
[2] S. Chaudhuri, C.D. Zaroliagis, Shortest paths in digraphs of small treewidth, Part II: optimal parallel algorithms, Theoret. Comput. Sci. 203 (1998) 205–223.
[3] S. Chaudhuri, C.D. Zaroliagis, Shortest paths in digraphs of small treewidth, Part I: sequential algorithms, Algorithmica 27 (2000) 212–226.
[4] B. Courcelle, Fundamental properties of infinite trees, Theoret. Comput. Sci. 25 (1983) 95–169.
[5] B. Courcelle, Monadic-second order graph transductions: a survey, Theoret. Comput. Sci. 126 (1994) 53–75.
[6] B. Courcelle, The expression of graph properties and graph transformations in monadic second-order logic, in: G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, Foundations, Vol. I, World Scientific, Singapore, 1997, pp. 313–400 (Chapter 5).
[7] B. Courcelle, J. Makowsky, U. Rotics, On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic, Discrete Appl. Math. 108 (2001) 23–52.
[8] B. Courcelle, M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Theoret. Comput. Sci. 109 (1993) 49–82.

[9] B. Courcelle, S. Olariu, Upper bounds to the clique-width of graphs, Discrete Appl. Math. 101 (2000) 77–114.
[10] B. Courcelle, I. Walukiewicz, Monadic second-order logic, graph coverings and unfoldings of transition systems, Ann. Pure Appl. Logic 92 (1998) 35–62.
[11] J. Engelfriet, G. Rozenberg, Node replacement graph grammars, in: G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, Foundations, Vol. I, World Scientific, Singapore, 1997, pp. 1–94 (Chapter 1).
[12] C. Gavoille, Private communication and paper in preparation.
[13] C. Gavoille, D. Peleg, S. Pérennes, R. Raz, Distance labeling in graphs, in: Proceedings of the 12th Symposium on Discrete Algorithms (SODA), ACM-SIAM, 2001, pp. 210–213.
[14] T. Hagerup, Dynamic algorithms for graphs of bounded treewidth, Algorithmica 27 (2000) 292–315.
[15] J. Spinrad, Implicit graph representation, Book in preparation, http://www.vuse.vanderbilt.edu/~spin/persinfo.html.
[16] W. Thomas, Automata on infinite objects, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier Science Publishers., Amsterdam, 1990, pp. 133–191 (Chapter 4).