Linear delay enumeration and monadic second-order logic

Bruno Courcelle
Bordeaux University and CNRS, LaBRI

March 28, 2007

Abstract

The results of a query expressed by a monadic second-order formula on a tree, on a graph or on a relational structure of tree-width at most k can be enumerated with a delay between two outputs proportional to the size of the next output. This is possible by using a preprocessing that takes time $O(n \cdot \log(n))$ where n is the number of vertices or elements. One can also output directly the i-th element with respect to a fixed ordering, however in more than linear time in its size. These results extend to graphs of bounded clique-width. We also consider the enumeration of finite parts of recognizable sets of terms specified by parameters like size, height or Strahler number.

Keywords : Monadic second-order logic, tree-width, enumeration, query, DAG, monadic second-order transduction, recognizable set of terms, tree automaton, unfolding, random generation.

Support : This work has been supported by the GRAAL project of the "Agence Nationale pour la Recherche".

Email: courcell@labri.fr

Postal address : LaBRI, Bordeaux University, 351 Cours de la Libération, F-33405 Talence, France.

1 Introduction

We are interested in the problem of enumerating the results of a query in a relational structure expressed in a logical language, actually monadic second-order logic in this article.

For a fixed formula, and given a relational structure S that can represent a word, a tree, a graph or a hypergraph subject to a condition like to have

tree-width bounded by a fixed integer k, we want an algorithm that builds a data structure over S that makes it possible to enumerate without repetition the set of results of the query, in such a way that the time taken to output the next result is as small as possible. For a query expressed in monadic second-order logic (MS logic in short), the results are p-tuples of subsets, where p is the number of free variables (assumed to be set variables). Hence the results have variable sizes. We aim at obtaining a delay between two outputs that is linear in the size of the next output, and a known delay depending on S for reporting the end of the enumeration.

The preprocessing should not consist of the construction of the set of results followed by a straightforward listing. The following result (a simplified version of Corollary 3 in Section 5) is representative of what we obtain. We denote by $Sat(G, \varphi(X_1, ..., X_p))$ the sets of p-tuples of sets that satisfy φ in G. This is the set of results of the query defined by φ in G.

Theorem : Let $\mathcal{C}$ be a set of graphs of tree-width at most k. For every monadic second-order formula $\varphi(X_1, ..., X_p)$, there exists an algorithm that takes as input a graph G in $\mathcal{C}$ and, after a preprocessing using time $O(n \cdot \log(n))$ where n is the number of vertices of G, that enumerates the set $Sat(G, \varphi(X_1, ..., X_p)) = \{B_1, ..., B_m\}$ with linear delay. The preprocessing defines a linear ordering of the set of results and the constructed data structure makes it possible to compute directly the i-th element B_i in time $O(\mid B_i \mid \cdot \log(n))$ where $\mid B_i \mid$ is its size.

Our main tool is the well-known translation of monadic second-order formulas on binary trees into tree-automata. It is applicable to graphs and relational structures of bounded tree-width via a translation of MS formulas on graphs or structures into MS formulas on labelled trees encoding tree-decompositions, followed by another translation into finite tree-automata. Another main tool is the notion of an *AND/OR directed graph without circuits* (in short an *AND/OR-DAG*). Such a graph embeds a set of trees (with sharings of isomorphic subtrees). The set $EMB(G, x)$ of trees embedded in G at vertex x is defined recursively as follows : if x has outdegree 0, then $EMB(G, x) = \{x\}$, if x is an OR-vertex then $EMB(G, x)$ is the set of trees $x(T)$ for T in $EMB(G, y)$ where y is a son of x, if x is an AND-vertex with sons $y_1, ..., y_n$, then $EMB(G, x)$ is the set of trees $x(T_1, ..., T_n)$ for T_i in $EMB(G, y_i)$. We first prove (Section 2) that we can enumerate these trees with linear delay.

From a tree-automaton and a given binary tree t, we build an AND/OR-DAG of size linear in the size of the given tree t. This DAG embeds all annotated trees representing the desired results. The annotations are p-tuples of Boolean values attached to nodes that indicate to which components of the considered p-tuple of sets a node belongs. The trees embedded in this DAG can be enumerated with linear delay. However, these trees contain a lot of useless information. In order to achieve linear delay, where linearity is evaluated with respect to the size of the output and not with respect to the size of the given tree, we perform a transformation quite close to the ε-reduction of finite automata (Section 4). It

eliminates the nodes with associated label $\lambda = (False, ..., False)$, because they do not belong to the result represented by the considered annotation. This last step requires time $O(n \cdot h)$, where n is the size of the tree t and h is its height, all other steps taking linear time in the size of the structures (with huge constants, which is common and actually unavoidable as proved by Frick and Grohe [9] if one wishes to get results for general MS formulas). However, we can improve this result by replacing h by $\log(n)$. For doing this we use a result of Courcelle and Vanicat [5] showing that a tree can be reorganized into a *balanced* one having height $O(\log(n))$ for n nodes. In the application to graphs of bounded tree-width (in Section 5), we use tree-decompositions of height $O(\log(n))$ for n vertices, which are not optimal in terms of tree-width. Yet another improvement is obtained if we input a tree t by means of a DAG which "unfolds" into t. Such a DAG is obtained from t by fusing its identical subtrees.

In Section 6, we also discuss the problem of enumerating finite parts of $T(F, A)$, the set of terms written with a set F of function symbols and a set A of constants. We denote by $\mid t \mid$ the size of a term and by $ht(t)$ its height. A term in $T(F, A)$ of height at most h can be seen as a labelling with symbols from $F \cup A$ of the nodes of a single ordered tree of height h, namely the tree $\mathbf{T}(h, k)$ of words over $\{1, ..., k\}$ of length at most $h-1$, where k is the maximum arity of a symbol in F. For every recognizable subset L of $T(F, A)$, there is a single MS formula that defines the labellings corresponding to the terms in L for any height h. It follows that the improvement of the above theorem based on representing terms by DAGs can also be used for enumerating with linear delay, for any h the terms of height at most h belonging to a recognizable set, because $\mathbf{T}(h, k)$ can be represented by a DAG with h vertices and hk edges.

We also give a more direct proof using the result of Section 2 and based on the enumeration of the set of trees embedded in an AND/OR-graph that is a kind of unfolding of a deterministic tree-automaton recognizing L, where each state is splitted into several states incorporating information like the size of the trees recognized by taking them as accepting states.

In all cases the data structures make it possible to generate directly the i-th element (a result of a query, a term or a graph). However the time complexity is (slightly) larger than in the case of enumeration because the data structure cannot be updated as it can be during the enumeration. We call this the *direct generation problem*. From a direct generation algorithm and the knowledge of the number of elements, one can define *random generation algorithms* with equal probabilities for all elements in the sets.

The article is organized as follows. Definitions are given in Section 2 together with the basic enumeration algorithm for the set of trees embedded in an AND/OR-DAG. The enumeration algorithm for monadic second-order queries on words is given in Section 3. The central results about binary trees and terms are in Section 4. Applications to graphs of bounded tree-width and bounded clique-width are given in Section 5. The enumeration of sets of words and terms is considered in Section 6. A table comparing the various results is given with

a conclusion in Section 7. An appendix reviews monadic second-order logic, monadic second-order transductions and clique-width.

2 Linear delay enumeration of trees embedded in a graph

In this section we define linear delay enumeration, AND/OR-DAGs and we establish the basic linear delay enumeration algorithm of the set of trees embedded in an AND/OR-DAG.

Let A be a set linearly ordered as $\{a_1, ..., a_p\}$. We assume that each element a of A has a size $\mid a \mid$ which is a positive integer, say the length of a word or more generally, the number of bits used to encode a. An algorithm enumerates A *with linear delay* if after a preprocessing phase, it outputs $a_1, ..., a_p$ in this order, and for each $i > 0$ it outputs a_i within time $O(\mid a_i \mid)$ (after the preceding output) and reports the end of the enumeration, i.e., discovers that the last element has been output, within bounded time. The value p is not necessarily known from the preprocessing. If $h : A \longrightarrow B$ is a bijection such that $\mid h(a) \mid = \Theta(\mid a \mid)$ for every $a \in A$ and $h(a)$ is computable in time $O(\mid a \mid)$, and if A is enumerable with linear delay, then so is B.

Notation : In complexity evaluations, we will write $O(n \cdot \log(n))$ instead of $O(n \cdot (\log(n) + 1))$, neglecting the fact that $\log(1) = 0$. All logarithms are in base 2. All graphs, words, trees and relational structures will be finite. We will denote by $Card(X)$ the cardinality of a set X and by $\mid u \mid$ the length of a word or the size of a term u. On sequences of different lengths, the elements of which are linearly ordered, we let the lexicographic order include the prefix order.

We prove in this section that the set of trees embedded in AND/OR-DAGs can be enumerated with linear delay, and with a preprocessing time that is linear in the size of the dag.

Definition 1 : *Depth First Traversal of a DAG.*

Let G be a *DAG*, i.e., a *directed graph without circuits*. Its vertex set is denoted by V_G. We say that a vertex x is *a son* of u if there is an edge from u to x, which we denote by $u \longrightarrow x$. A vertex is a *leaf* if it has no son. A DAG G may have multiple edges. It is *locally ordered* if it is equipped with a partial order on its edge set E_G that linearly orders each set of edges outgoing from a same vertex. The corresponding strict partial order is denoted by $<$. When implemented, a DAG is always linearly ordered since its vertices and edges are represented by distinct bit sequences. Hence, there is no loss of generality in

assuming the existence of a linear order on the set $V_G \cup E_G$. Hence every DAG is locally ordered.

For every vertex x of G, we define $DFT(G, x)$, a sequence of vertices starting and ending with x and such that consecutive vertices are neighbours. We call it a *depth-first traversal* of the subgraph $G \downarrow x$, defined as the union of all directed paths originating at x. It is defined recursively, with as auxiliary argument, a set V of vertices intended to be the set of those *already visited*. The variable V is global and is updated during the computation. We define $DFT(G, V, y)$ as follows :

1. If y is a leaf or if $y \in V$, then $DFT(G, V, y) = (y)$, and y is put in V (unless it is already in V) and said then to be *visited*.

2. Else, y has outgoing edges that we enumerate as $e_1 < ... < e_n$, with respective end vertices $z_1, ..., z_n$ (a vertex may occur twice or more in this list), we define :

$$DFT(G, V, y) = (y) \cdot DFT(G, V, z_1) \cdot (y) \cdot DFT(G, V_1, z_2) \cdot (y) \cdot ... \\ \cdot (y) \cdot DFT(G, V_{n-1}, z_n) \cdot (y)$$

where V_1 is V augmented with the set of vertices visited during the execution of $DFT(G, V, z_1)$, $(y \notin V_1)$,

V_2 is V_1 augmented with the set of vertices visited during the execution of $DFT(G, V_1, z_2)$, $(y \notin V_2)$,...

and V_{n-1} is V_{n-2} augmented with the set of vertices visited during the execution of $DFT(G, V_{n-2}, z_n)$.

3. To conclude the execution of $DFT(G, V, y)$, the set of visited vertices is updated by $V := V_{n-1} \cup \{y\}$. It contains y, $z_1, ..., z_n$ and their descendants.

We denote by $\cdot$ the concatenation of sequences. The calls to $DFT(G, V, z_1)$, $DFT(G, V_1, z_2), ..., DFT(G, V_{n-1}, z_n)$ are executed in this order.

We let then $DFT(G, x) = DFT(G, \varnothing, x)$. It is clear that every vertex $y \neq x$ in $G \downarrow x$ has $\deg(y)$ occurrences in $DFT(G, x)$, i.e. is visited $\deg(y)$ times, where $\deg(y)$ is the *degree* of y. The vertex x is visited $\deg^+(x) + 1$ times, where $\deg^+(x)$ is its *outdegree* i.e., the number of outgoing edges. (We denote by $\deg^-(x)$ the *indegree* of x). Every edge is traversed twice, first in its direction, and the second time in the opposite one.

The construction of $DFT(G, x)$ by this procedure, which we will call the depth-first traversal of $G \downarrow x$ takes linear time in the number of edges of $G \downarrow x$. We assume that the DAG G is given in such a way that the first outgoing edge from a vertex and the i-th one among those with same origin are accessible in constant time.

Definitions 2 : *AND/OR*-DAGs *and the trees they embed.*

(a) We let $\mathcal{D}$ be the class of locally ordered DAGs G, called *AND/OR-DAGs*, such that :

(a1) the vertex set V_G is partitionned into two sets V_G^{and} and V_G^{or} called the sets of *AND-vertices* and of *OR-vertices*,

(a2) there are no two edges with same origin x and same target if x is an OR-vertex (equivalently, two such edges are fused and considered as a single one in traversal algorithms and degree evaluations),

(a3) the *leaves*, i.e., the vertices of outdegree 0 are AND-vertices.

Such a DAG is an *OR-DAG* if the AND vertices are all of outdegree 1 or 0 and an *AND-DAG* if the OR vertices are all of outdegree 1.

(b) The *height* of a vertex x in G is the maximal length of a directed path in $G \downarrow x$.

(c) An AND-DAG G is called a *tree* if :

(c1) $G = G \downarrow r$ for a (unique) vertex r called its *root* and denoted by $Root(G)$ and

(c2) whenever x and y are the ends of two distinct edges with origin a same AND-vertex, then $V_{G\downarrow x} \cap V_{G\downarrow y} = \varnothing$.

Every tree in this sense is a directed tree in the usual sense.

(d) *Tree embeddings*

Let T and $G \in \mathcal{D}$, where T is a tree, and let x be a vertex of G. An *embedding of T in G at x* is a mapping h that maps V_T^{or} into V_G^{or}, V_T^{and} into V_G^{and}, E_T into E_G, that preserves incidences and the linear orderings of the sets of edges with a same origin and furthermore :

(d1) $x = h(Root(T))$,

(d2) if u is an AND-vertex of T then h is a bijection of the set of edges outgoing from u in T onto the set of those outgoing from $h(u)$ in G ; it follows that the outdegrees of u and $h(u)$ are the same, and the leaves of T are mapped to leaves of G.

If u is an ancestor of v in T, then $h(u)$ is an ancestor of $h(v)$ and $h(u) \neq h(v)$. But the mapping h need not be injective on the set $V_T^{or} \cup V_T^{and}$. However it is if G satisfies Condition (c2) (even without being a tree). We prove this as follows. If u and v are incomparable in T, then u is in $T \downarrow u'$ and v is in $T \downarrow v'$ where u' and v' are two distinct sons of a vertex w that must be an AND-vertex. Hence $h(w)$ is an AND-vertex, $h(u)$ is in $G \downarrow h(u')$, $h(v)$ is in $G \downarrow h(v')$. The two edges $w \longrightarrow u'$ and $w \longrightarrow v'$ of T are mapped to two distinct edges $h(w) \longrightarrow h(u')$ and $h(w) \longrightarrow h(v')$ of G. Condition (c2) implies that $V_{G\downarrow h(u')} \cap V_{G\downarrow h(v')} = \varnothing$, hence $h(u) \neq h(v)$.

We denote by $EMB(G, x)$ the set of trees (up to isomorphism) embedded in G at x, and by $EMB(G)$ the set of those embedded in G at some x.

If G is an AND-DAG, then $EMB(G, x)$ consists of a single tree (up to isomorphism), also called the *unfolding of G from x*, and denoted by $Unf(G, x)$.

(e) *A linear notation for embedded trees.*

We write a tree T as $r(T_1, ..., T_n)$ if r is its root and $T_i = T \downarrow y_i$ where $y_1, ..., y_n$ are the sons of r, numbered by increasing order of the edges with origin r and respective ends $y_1, ..., y_n$. Clearly, each T_i is a tree with root y_i. By using recursively this notation, we obtain a linear notation for trees. It does not distinguish the AND-vertices from the OR-vertices; vertices of both types can be of outdegree 1.

If h is an embedding of a tree T in G, and T is linearly denoted as above by $r(..., ..., ...)$, then, by replacing in this notation each node u of T by its image $h(u)$, we obtain a linear notation that represents simultaneously T (or any tree isomorphic to T) and its embedding in G.

For an example, let the AND-DAG G have edges $x \longrightarrow y$, $y \longrightarrow u$, $x \longrightarrow z$, $z \longrightarrow u$. The set $EMB(G,x)$ has a unique element denoted by $x(y(u),z(u))$.

For another example, consider the DAG of Figure 1, where the OR-vertices are x and y. The tree $p(q(r(u)),q'(s(v,w)))$ embeds into this DAG by the embedding k such that $k(p) = f, k(q) = k(q') = x$, etc... The linear notation $f(x(y(a)),x(h(b,c)))$ represents this embedding.

(f) *Linear orderings of the sets* $EMB(G,x)$.

We define on each set $EMB(G,x)$ a strict ordering $\prec$ as follows :

$T \prec T'$ if and only if :

(f1) either x is an OR-vertex with sequence of sons $y_1,...,y_n$, ordered according to the ordering of edges outgoing from x, $T = x(U)$, $T' = x(U')$, $U \in EMB(G,y_i)$, $U' \in EMB(G,y_j)$ and $i < j$, or $i = j$ and $U \prec U'$ in $EMB(G,y_i)$.

(f2) either x is an AND-vertex with sequence of sons $y_1,...,y_n$ as above, $T = x(T_1,...,T_n)$, $T' = x(T'_1,...,T'_n)$ and :

either $T_n \prec T'_n$,
or $T_n = T'_n$ and $T_{n-1} \prec T'_{n-1}$,
or $T_n = T'_n$ and $T_{n-1} = T'_{n-1}$ and $T_{n-2} \prec T'_{n-2}$,
or ... and $T_2 = T'_2$ and $T_1 \prec T'_1$.

Remarks 1 : (1) Consider the DAG G with AND-vertices a,b,c, with OR-vertices x and y and edges $x \longrightarrow y, x \longrightarrow c, y \longrightarrow a, y \longrightarrow b, y \longrightarrow c$. Then $EMB(G,x) = \{xya, xyb, xyc, xc\}$, $EMB(G,y) = \{ya, yb, yc, c\}$.We do not have $EMB(G,y) \subseteq EMB(G,x)$ as one might think. That is we do not "evaluate" OR-vertices as set unions. (For readability, parentheses surrounding a single subtree are omitted. We write xya instead of $x(y(a))$). It follows that two sets $EMB(G,x)$ for distinct vertices x are disjoint.

(2) By using the linear notation of Definition 2 (e), one can define alternatively the set of trees $EMB(G,x)$ by :

1. either x is a leaf and $EMB(G,x) = \{x\}$,
2. or x is an OR-vertex and :

$$EMB(G,x) = \{x(T) \mid T \in EMB(G,y), y \text{ is a son of } x\},$$

3. or x is an AND-vertex with sons $y_1,...,y_n$ as in (f2) and :

$$EMB(G,x) = \{x(T_1,...,T_n) \mid T_i \in EMB(G,y_i) \text{ for } i = 1,...,n\}.$$

Lemma 1 : The relation $\prec$ is a strict linear ordering of each set $EMB(G,x)$.

As an illustration, we note that the $\prec$-smallest element of $EMB(G,x)$, denoted by $F(x)$, is defined recursively as follows :

$F(x) = x$ if x is a leaf,

$F(x) = x(F(y_1),...,F(y_n))$, if x is an AND-vertex with sons $y_1,...,y_n$ as in (f2),

$F(x) = x(F(y_1))$, if x is an OR-vertex with sons $y_1,...,y_n$ as in (f1).

Let G be a DAG in $\mathcal{D}$ and x a vertex of G. We define $N(x) = Card(EMB(G, x))$. By assuming that arithmetic operations can be performed in unit time, independently on how large the values of arguments are, we get the following lemma.

Lemma 2 : (1) The numbers $N(x)$ can be computed by the following rule :
$N(x) = 1$ if x is a leaf,
$N(x) = N(y_1) \cdot ... \cdot N(y_n)$ if x is an AND-vertex with sons $y_1, ..., y_n$,
$N(x) = N(y_1) + ... + N(y_n)$ if x is an OR-vertex with sons $y_1, ..., y_n$.
(2) The labelling of all vertices y of $G \downarrow x$ by the corresponding integers $N(y)$ can be done in time $O(Card(E_{G\downarrow x}))$.

Proof : (1) This is clear from the definitions. We use here condition (a2).
(2) The rules specify $N(x)$ in a unique way, this is clear by induction on the height of x. The value $N(y)$ can be attached to each vertex y of $G \downarrow x$ during a depth-first traversal of $G \downarrow x$, which gives the result. □

For every vertex y in G we denote by $E(i, y)$ the i-th element of $EMB(G, y)$. The function E depends on the ordering $<$ of the set of edges.

Lemma 3 : (1) $E(1, y) = y$ if y is a leaf ; $E(i, y)$ is undefined if y is a leaf and $i > 1$.
(2) $E(i, y) = y(E(i-N(z_1)-...-N(z_k), z_{k+1}))$ if y is an OR-vertex with sons $z_1 < ... < z_n$, k is the largest integer in $\{0, 1, ..., n\}$ such that $N(z_1)+...+N(z_k) < i$; $E(i, y)$ is undefined if y is so but no such k does exist, which means that $i > N(z_1) + ... + N(z_n) = N(y)$.
(3) $E(i, y) = y(E_1, ..., E_n)$ if y is an AND-vertex with sons $z_1 < ... < z_n$, and the following hold :
$i = i_1+(i_2-1)N(z_1)+(i_3-1)N(z_1)N(z_2)+...+(i_n-1)N(z_1)N(z_2)...N(z_{n-1})$,
with $1 \leq i_1 \leq N(z_1)$, $1 \leq i_2 \leq N(z_2)$, ... , $1 \leq i_n \leq N(z_n)$ and
$E_1 = E(i_1, z_1)$, ... , $E_n = E(i_n, z_n)$.
$E(i, y)$ is undefined if y is an AND-vertex with sons $z_1, ..., z_n$ as in (f2) and the above conditions cannot be realized, which means that $i > N(z_1)\cdot...\cdot N(z_n) = N(y)$.
The sequence $i_1, ..., i_n$ is unique when it exists.

Proof : Clear from the recursive definition of $EMB(G, x)$ and elementary properties of the lexicographic ordering of a Cartesian product $C_1, ..., C_n$ where each set C_i has $N(z_i)$ elements.□

We will denote by $Select^{OR}$ the mapping that associates $(i - N(z_1) - ... - N(z_k), z_{k+1})$ with each (i, y) satisfying Case (2), by $Select^{AND}$ the mapping that associates $(i_1, z_1, i_2, z_2, i_3, z_3, ..., i_n, z_n)$ with each (i, y) satisfying Case (3) and by $\deg^{OR}(G)$ the maximum outdegree of an OR-vertex. The size of a tree E is $\mid E \mid = Card(V_E)$.

Theorem 1 : Let G be an AND/OR-DAG and x a vertex.

(1) After a preprocessing taking time $O(Card(E_{G\downarrow x}))$, one can enumerate the set $EMB(G,x)$ with linear delay.

(2) After a preprocessing taking time $O(Card(E_{G\downarrow x}))$, one can compute for any i the tree $E(i,x)$ in time $O(\mid E(i,x) \mid \cdot \log(\deg^{OR}(G)))$.

Proof : (1) The characterization of Lemma 3 yields a recursive procedure for defining $E(i,x)$. For enumerating the set $EMB(G,x)$, it suffices to compute successively $E(i,x)$ for $i = 1, ..., N(x)$.

The preprocessing consists in the labelling defined in Lemma 2. However, we must examine the time taken for computing $Select^{AND}(i,y)$ and $Select^{OR}(i,y)$ whenever they are needed.

For $Select^{AND}(i,y)$ where y has sons $z_1, ..., z_n$ as in (f2), and since the values $N(z_1), ..., N(z_n)$ are available, one obtains the sequence $(i_1, z_1, i_2, z_2, i_3, z_3, ..., i_n, z_n)$ in time $O(n)$ by n Euclidian divisions. Since this computation is intended to set up n outgoing edges in the resulting tree, it fits inside the desired linear time bound in terms of the size of the result.

This is not the same for $Select^{OR}(i,y)$. Its computation in the case where it produces $(i - N(z_1) - ... - N(z_k), z_{k+1})$ takes time $O(k)$. This time is not always proportional to the size of the resulting tree. However, $Select^{OR}(i,y)$ can be computed in constant time from $Select^{OR}(i-1,y)$. Since $Select^{OR}(i,y)$ is requested only after $Select^{OR}(i-1,y)$ has been also requested, we can build a table for the function $Select^{OR}$. The entries of the table are computed only when needed and immediately inserted in view of later use. They are not computed during a preprocessing. Thus we obtain a linear delay enumeration algorithm. Setting up a similar table for $Select^{AND}$ can also be done for accelerating later computations but is not necessary for achieving linear delay.

(2) The progressive construction of a table for $Select^{OR}$ does not work if one only wants $E(i,y)$ for random values i. In this case the remedy is to build for each OR-vertex y a binary search tree of height $h_y = \lceil \log(\deg^+(y)) \rceil$ for obtaining $Select^{OR}(i,y)$ in time $O(h_y)$. This can be done for each y in time $O(\deg^+(y))$ during the traversal of G that computes and installs the values $N(y)$. We obtain thus with a preprocessing time of same order as for (1) a generation time $O(\mid E(i,x) \mid \cdot \log(\deg^{OR}(G)))$. □

Example 1 : Figure 1 shows an AND/OR-DAG with its vertices labelled by the values of function N. The OR-vertices are x and y. The enumeration yields the following sequence of trees, written with the linear notation of Definition 2 (e). With each tree we indicate the values of $Select^{OR}$ entered in the table. We assume that the table is initialized with $Select^{OR}(1,w) = (1,w')$ for every vertex w, where w' is its first son.

$1 : f(xg(a,b,ya), xg(a,b,ya))$

$2 : f(xg(a,b,yc), xg(a,b,ya))$;

$Select^{OR}(2,y) = (1,c)$; $Select^{OR}(2,x) = (2,g)$

$3 : f(xya, xg(a,b,ya))$; $Select^{OR}(3,x) = (1,y)$,

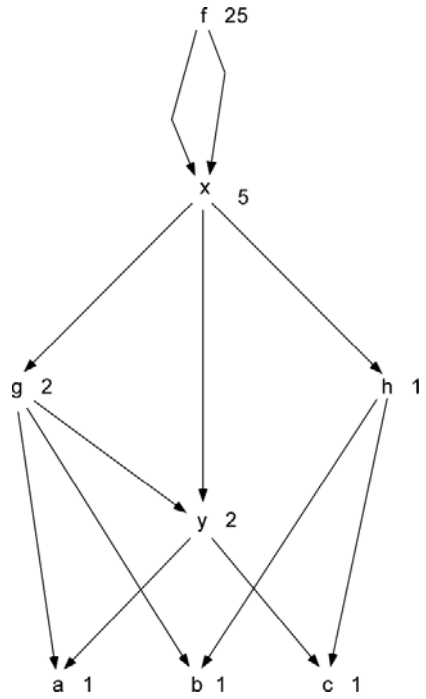

Figure 1: An AND/OR-DAG and the values $N(y)$ at its vertices y.

$4 : f(xyc, xg(a,b,ya))$; $Select^{OR}(4,x) = (2,y)$,
$5 : f(xh(b,c), xg(a,b,ya))$; $Select^{OR}(5,x) = (1,h)$,
$6 : f(xg(a,b,ya), xg(a,b,yc))$,
$7 : f(xg(a,b,yc), xg(a,b,yc))$,
$8 : ....$

Remarks 2 : (1) Theorem 1 extends to the enumeration of $A = \bigcup\{EMB(G,x) \mid x \in X\}$ where X is a set of vertices. The sets $EMB(G,x)$ are pairwise disjoint as observed in Remark 1 (1). To enumerate A, it suffices to add a new OR-vertex r, edges from r to all vertices of X and a linear order on these edges, yielding an AND/OR-DAG G' that is locally ordered. The enumeration of A reduces to that of $EMB(G', r)$.

(2) *Space requirements in the algorithms of Theorem 1*: The space needed to store the function $Select^{OR}$ is $O(\sum\{N(y) \mid y \in V^{or}_{G\downarrow x}\})$. However, if G satisfies Condition (c2) of Definition 2 (without being necessarily a tree), then we need not store $Select^{OR}(i,y)$ for all i, but only (for each y) for the last value i for which $Select^{OR}(i,y)$ has been used (because embeddings of trees are injective). In this case the required space is only $O(Card(V^{or}_{G\downarrow x}))$ for these values, hence $O(Card(E_{G\downarrow x}))$ in total. For the direct generation algorithm, we need space $O(\sum\{\deg^+(y) \mid y \in V^{or}_{G\downarrow x}\})$ which is bounded by $O(Card(E_{G\downarrow x}))$.

In the example of Figure 1, Condition (c2) does not hold. Observe that $E(17,f) = f(E(2,x), E(4,x))$, hence $Select^{OR}(2,x)$ and $Select^{OR}(4,x)$ are needed simultaneously. So we need to store all $Select^{OR}$ unless we compute each value when needed. In this case, by using the technique of (2) of Theorem 1, we achieve delay $O(s \cdot \log(\deg^{OR}(G)))$ where s is the size of the generated tree. This may be considered as a good result if the considered graphs have bounded outdegree.

3 Queries on words

We give a linear delay enumeration algorithm for the set of answers to a monadic second-order query on words. Although these queries could be handled as a particular case of queries on binary trees to be considered next, we consider them first in order to facilitate the presentation of the general construction. All our constructions use DAGs based on finite automata associated with monadic second-order formulas.

Let w be a word in A^+, the set of finite nonempty words over a finite alphabet A. It will be handled as the relational structure S_w with domain D_{S_w} consisting of one element for each occurrence of a letter, a successor relation defining the next occurrence, and unary predicates associated with letters in A which say which letter is at an occurrence. The relational signature of S_w is denoted by σ_A.

Let $\varphi(X_1, ..., X_k)$ be a *monadic second-order formula* (*MS formula*) and $Sat(S_w, \varphi(X_1, ..., X_k))$ be the set of k-tuples of subsets of D_{S_w} that satisfy φ. (Monadic second-order logic is reviewed in the appendix.). We define the size of $(U_1, ..., U_k)$ as $k + Card(U_1) + ... + Card(U_k)$, so that the size of $(\varnothing, ..., \varnothing)$ is not 0. It could also be defined as $1 + Card(U_1 \cup ... \cup U_k)$, this would not change our results regarding linear delay enumeration.

Theorem 2 : Let A be a finite alphabet and $\varphi(X_1, ..., X_k)$ be an MS formula over the signature σ_A. For every word w in A^+ there exists a linear delay enumeration algorithm of the set $Sat(S_w, \varphi(X_1, ..., X_k))$ with preprocessing time $O(\mid w \mid^2)$. There is an algorithm using a preprocessing of same time complexity that generates the i-th result B_i within time $O(\mid B_i \mid \cdot \log(n))$ where $\mid B_i \mid$ is the size of B_i. The results are enumerated or generated with respect to a fixed lexicographic order.

Proof : The set $Sat(S_w, \varphi(X_1, ..., X_k))$ is represented in a well-known way (see [17]) by a regular language $L_{\varphi(X_1,...,X_k)} \subseteq C^+$, where C is the alphabet $A \times \mathbf{B}^k$, $\mathbf{B} = \{True, False\}$. For $w \in A^+$ and subsets $U_1, ..., U_k$ of D_{S_w}, we let $w[U_1, ..., U_k]$ be the word in C^+ obtained by replacing in w each occurrence u of a letter a by (a, α), where $\alpha = (b_1, ..., b_k)$, $b_i = False$ if $u \notin U_i$, and $b_i = True$ if $u \in U_i$. Hence $w[U_1, ..., U_k]$ encodes w and the k-tuple $(U_1, ..., U_k)$.

The set $L_{\varphi(X_1,...,X_k)}$ of all words $w[U_1, ..., U_k]$ for $w \in A^+$ and $(U_1, ..., U_k) \in Sat(S_w, \varphi(X_1, ..., X_k))$ is recognizable, and is recognized by a finite deterministic automaton $\mathcal{A}$ effectively constructible from $\varphi(X_1, ..., X_k)$. Let Q be its set of states. For every w in A^+ of length n, we build a DAG $G_{\mathcal{A}}(w)$ as follows :

1. Its set of vertices is $Q \times [0, n]$ where $[0, n]$ denotes $\{0, 1, ..., n\}$.

2. The edges are from $(q, i-1)$ to (p, i) with label (i, α) in $\{1, ..., n\} \times \mathbf{B}^k$ whenever there is in $\mathcal{A}$ a transition from q to p associated with letter (a, α) where a is the i-th letter of w. (We replace letter a by the corresponding occurrence i in the word).

This DAG is an OR-DAG, where all vertices except the leaves are OR-vertices. It has $(n+1) \cdot Card(Q)$ vertices and at most $n \cdot Card(Q) \cdot 2^k$ edges (exactly this number if the automaton is complete). It can be constructed in time $O(n)$.

Let q_0 be the initial state of $\mathcal{A}$, and Q_{Acc} be its set of final (accepting) states. In the DAG $G_{\mathcal{A}}(w)$, all paths going from $(q_0, 0)$ to a vertex (p, n) for some p in Q_{Acc} have length n. They are in bijection with the elements of the set $Sat(S_w, \varphi(X_1, ..., X_k))$ (because the automaton $\mathcal{A}$ is deterministic). By applying Theorem 1 to $G_{\mathcal{A}}(w)$, one can enumerate with linear delay this set of paths. Hence one obtains a set of sequences, each of length n, that represents $Sat(S_w, \varphi(X_1, ..., X_k))$. However, in these sequences, the components (i, λ) where $\lambda = (False, ..., False)$ are useless. They correspond to occurrences in the word which belong to no component of the considered tuple. One can delete them from the outputs, but the enumeration delay is then no longer linear.

Considering the DAG $G_{\mathcal{A}}(w)$ as a finite-state automaton, we will apply to it the well-known transformation called *ε-reduction*. We will mainly replace sequences of edges with labels of the form (i, λ) by direct edges. This transformation can be described as follows :

First step : We remove (or mark as "removed") all vertices of $G_{\mathcal{A}}(w)$ which are not on paths from the vertex $(q_0, 0)$ to a vertex (p, n) for any p in Q_{Acc}. We obtain a DAG $G_1(w)$.

Second step : For every transition from $(p, i-1)$ to (q, i) with label (i, λ), we replace this label by ε.

Third step : For every state $(q, i), i < n-1$, from which there is a sequence of ε-transitions to (p, n) for some p in Q_{Acc}, we create a "direct" ε-transition from (q, i) to (p, n).

Fourth step : For every q, i, p, r, j such that there is a sequence of ε-transitions from (q, i) to (p, j) and there is a transition from (p, j) to $(r, j+1)$ with label $(j+1, \alpha)$, we create a "direct" transition from (q, i) to $(r, j+1)$ with label $(j+1, \alpha)$.

Fifth step : We delete all ε-transitions except those with target (p, n) for some p in Q_{Acc}.

We let $G_2(w)$ be the AND/OR-DAG obtained in this way. (The AND-vertices are those of the form (p, n) for some p in Q_{Acc}). The trees in $EMB(G_2(w), (q_0, 0))$ are actually paths. Their edges are labelled by pairs (j, α) with $\alpha \neq \lambda$ and $0 < j \leq n$, except for the last edges which can be labelled by ε. The length of a path is the cardinality of $U = U_1 \cup ... \cup U_k$ up to 1, where $(U_1, ..., U_k)$ is the represented element of $Sat(S_w, \varphi(X_1, ..., X_k))$. (The length is one more than the cardinality of U when the last edge of such a path is labelled by ε). These paths can be enumerated with linear delay, which gives a linear delay enumeration of $Sat(S_w, \varphi(X_1, ..., X_k))$.

In the fourth step, there are at most $n - i - 1$ pairs (p, j) associated with each (q, i) (because the automaton $\mathcal{A}$ is deterministic, so that no two transitions with label (i, λ) have same origin). It follows that at most $(2^k - 1)(n - i - 1)$ "direct" transitions with source (q, i) are created. In total, the DAG $G_2(w)$ has less than $n^2 \cdot Card(Q) \cdot 2^k$ edges, and outdegree bounded by $n \cdot 2^k$.

The first two steps can be performed in time $O(n)$ in the course of depth-first traversals as defined in Section 2. Step 4 may produce a DAG with $O(n^2)$ edges.

Steps 3 to 5 can be done in time $O(n^2)$ as follows. We let $L(i)$ be the set of 4-tuples $(q, r, \alpha, j+1)$ where q, r, α, j are defined in Step 4, including the case empty sequences of ε-transitions, so that we can have in $L(i)$ some 4-tuples with $j = i$, corresponding to the transitions of $G_1(w)$ that are not $(i+1, \lambda)$-transitions. We put also in $L(i)$ the triple (q, p, n) whenever there is a direct edge from (q, i) to (p, n) as specified in Step 3. Each set $L(i-1)$ can be computed from $L(i)$ in time $a \cdot Card(L(i))$ for a constant a depending on k and the number of states. Hence, we obtain the DAG $G_2(w)$ by computing successively $L(n-1), L(n-2), ..., L(0)$ in total time $a \cdot (\sum\{Card(L(i)) \mid 0 \leq i <$

$n\}) = O(n^2)$. We can apply Theorem 1, and we obtain an enumeration with finite delay. The number of results is known from the preprocessing.

The results with their complexity evaluations follow from Theorem 1, using the fact that $\deg^{OR}(G) \leq n \cdot 2^k$.

An element of $Sat(S_w, \varphi(X_1, ..., X_k))$ is encoded by a sequence $(i_1, \alpha_1), ..., (i_p, \alpha_p)$ with $0 < i_1 < ... < i_p \leq n$ and $\alpha_1, ..., \alpha_p \in \mathbf{B}^k - \{\lambda\}$. Let us order pairs by $(i, \alpha) < (j, \beta)$ if and only if, either $i < j$, or $i = j$ and $\alpha <_{lex} \beta$, where $<_{lex}$ is the lexicographic order on $\mathbf{B}^k$, and let $\varepsilon < (i, \alpha)$ for every i and α. We obtain a lexicographic linear order on sequences $(i_1, \alpha_1), ..., (i_p, \alpha_p)$. By Theorem 1, the set $Sat(S_w, \varphi(X_1, ..., X_k))$ can be enumerated in this order, provided the edges of the DAG $G_2(w)$ are treated according to the order $<$ on their labels.The i-th element of $Sat(S_w, \varphi(X_1, ..., X_k))$ can be output in linear time in its size. Its elements can be randomly generated with respect to the same order. $\square$

Remarks 3 : (1) We can roughly evaluate the efficiency of this algorithm as follows: for time $p = b \cdot n^2$ spent in preprocessing for a word of length n, we may enumerate $2^{k \cdot n} = 2^{a \cdot p^{1/2}}$ results with linear delay.

(2) *Required space* : Condition (c2) holds trivially. The required space is thus the size of the DAG $G_2(w)$, hence $O(n^2)$. For the direct generation algorithm, one needs space $O(n \cdot \log(n))$ to handle the selection at OR-vertices, thus in total we need space $O(n^2)$.

Example 2 : We consider words with a single letter a and the MS formula with first-order variables x, y, z expressing that $x < y < z$ where $<$ is the natural order on occurrences of letters. This formula is MS and not first-order because the structure S_w uses a successor relation and not an order relation. The corresponding language is described by the regular expression $\lambda^*\alpha\lambda^*\beta\lambda^*\gamma\lambda^*$ where :

$$\alpha = (True, False, False),\ \beta = (False, True, False),$$
$$\gamma = (False, False, True),\ \lambda = (False, False, False),$$

and we replace (a, μ) by μ for each μ in $\mathbf{B}^3$. The triple α (resp. β, γ, λ) means that the corresponding occurrence is the value of x (resp. of y, of z, of none of x, y, z). For a word of length n, the preprocessing time takes time $\Theta(n^2)$, because the resulting automaton has $\Theta(n^2)$ edges (see below its description). The delay between two results is constant (since the results are triples of occurrences in the word), and there are $\Theta(n^3)$ results. So the construction of the total set of satisfying triples cannot be done during a preprocessing step taking only time $O(n^2)$. The enumeration algorithm cannot be replaced by the trivial listing of a set constructed during a preprocessing phase taking time $O(n^2)$.

We now describe the construction of $G_2(w)$. We use a deterministic automaton $\mathcal{A}$ for the language $\lambda^*\alpha\lambda^*\beta\lambda^*\gamma\lambda^*$ with states p, q, r, s , p initial, s accepting, transitions $p \longrightarrow q$ for letter α, $q \longrightarrow r$ for letter β, $r \longrightarrow s$ for letter γ, and loops on each state for letter λ. It is not complete.

The DAG $G_A(w)$ can be described as follows : Its vertices are pi, qi, ri, si for $i = 0, ..., n$. The edges are $pi \longrightarrow q(i+1)$ labelled by $(i+1, \alpha)$, $qi \longrightarrow r(i+1)$ labelled by $(i+1, \beta)$, $ri \longrightarrow s(i+1)$ labelled by $(i+1, \gamma)$, and $xi \longrightarrow x(i+1)$ labelled by λ, for every x in $\{p, q, r, s\}$ and for all relevant integers i. The first step eliminates vertices $q0, r0, r1, s0, s1, s2$ and $p(n-2), p(n-1), pn, q(n-1), qn, rn$.

The final DAG $G_2(w)$ consists of the following edges :

$pi \longrightarrow qj$ labelled by (j, α) for $0 \leq i < j < n-1$,

$qi \longrightarrow rj$ labelled by (j, β) for $1 \leq i < j < n$,

$ri \longrightarrow sj$ labelled by (j, γ) for $2 \leq i < j \leq n$,

$si \longrightarrow sn$ labelled by ε for $3 \leq i < n$.

This example does not witness the maximal efficiency described in Remark 3 (1). □

4 Queries on binary trees and terms

This section contains the main results : linear delay enumeration algorithms for MS queries on terms given directly or as unfoldings of DAGs.

A *(functional) signature* is a pair (F, A) of a finite set F of function symbols of positive arity ($\rho(f)$ is the arity of f) and a finite set A of constants. We denote by $T(F, A)$ the set of terms built over it, by $Occ_x(t)$ the set of occurrences of a symbol x in $t \in T(F, A)$, and by $Occ_A(t), Occ_F(t)$ the sets of occurrences of symbols in A and in F and we let $Occ(t) = Occ_A(t) \cup Occ_F(t)$. (Occurrences may be defined as sequences of integers or in another way). We denote by $\mid t \mid = Card(Occ(t))$ the *size* of t in $T(F, A)$ and by $ht(t)$ its *height*, i.e., the number of nodes of a longest path in t (considered as a tree) from the root to a leaf. Hence $ht(a) = 1$, if a is a constant.

Each term t in $T(F, A)$ can be represented by a relational structure S_t whose domain D_{S_t} is $Occ(t)$. If p is the maximum arity of a symbol in F, the structure S_t has p binary (functional) relations for pointing to the i-th *son* of each occurrence ($1 \leq i \leq p$) and unary relations relating symbols and their occurrences. We denote by $\tau_{F,A}$ the relational signature of S_t. (Examples of formulas using this signature are given in the appendix, see Example 7).

Let $\varphi(X_1, ..., X_k)$ be an MS formula over $\tau_{F,A}$. As in Section 2, we will use $\mathbf{B} = \{True, False\}$ and λ to denote $(False, ..., False) \in \mathbf{B}^k$. We want to enumerate the set $Sat(S_t, \varphi(X_1, ..., X_k))$ of k-tuples of subsets of D_{S_t} that satisfy φ. A tuple $(U_1, ..., U_k)$ in this set will be handled as the set of pairs (u, α) where $u \in U_1 \cup ... \cup U_k$, $\alpha = (b_1, ..., b_k) \in \mathbf{B}^k - \{\lambda\}$ with $b_i = False$ if $u \notin U_i$ and $b_i = True$ if $u \in U_i$. We define the *size* of a tuple $(U_1, ..., U_k)$ as $1 + Card(U_1 \cup ... \cup U_k)$. In complexity evaluations, we consider that k is fixed and that each element of D_{S_t} is a data of fixed size, as done by Durand and Grandjean in [8].

For $t \in T(F,A)$ and subsets $U_1, ..., U_k$ of $Occ(t)$, we let $t[U_1, ..., U_k]$ be the term in $T(F \times \mathbf{B}^k, A \times \mathbf{B}^k)$ called an *annotation* of t obtained by replacing at each occurrence u of a symbol x in $F \cup A$ this symbol by (x, α) where $\alpha = (b_1, ..., b_k)$, $b_i = False$ if $u \notin U_i$, and $b_i = True$ if $u \in U_i$. Hence the term $t[U_1, ..., U_k]$ encodes t (its *underlying* term) and the k-tuple $(U_1, ..., U_k)$. We denote by $NL(t[U_1, ..., U_k])$ the set of pairs (u, α) such that $\alpha \in \mathbf{B}^k - \{\lambda\}$ and $u \in Occ_{(a,\alpha)}(t[U_1, ..., U_k])$ for some $a \in A$.

Assumptions : (a) We assume that F has only binary function symbols. The terms in $T(F,A)$ are considered as *binary trees.* Furthermore, we assume that $\varphi(X_1, ..., X_k)$ is written in such a way that the sets $X_1, ..., X_k$ which satisfy φ are sets of occurrences of constants, called below *leaves*, by using the terminology of trees.

(b) We assume that $Sat(S_t, \varphi(X_1, ..., X_k))$ is not empty, and does contain the k-tuple $(\varnothing, ..., \varnothing)$. In all cases we will consider, this condition can be decided in linear time, and this test can be included in the preprocessing phase. If $Sat(S_t, \varphi(X_1, ..., X_k))$ contains $(\varnothing, ..., \varnothing)$, we can output this result and continue with the modified formula $\varphi(X_1, ..., X_k) \wedge (\bigvee_{1 \leq i \leq k} X_i \neq \varnothing)$.

Assumption (a) will be lifted later. It simplifies the main proof. Assumption (b) will be made without loss of generality.

We will use *complete deterministic bottom-up finite* tree-automata (see the book on line [3] for tree automata) simply called *automata* in the sequel. (Using *complete* automata is not essential but simplifies the formal description of constructions). We recall that the transitions of an automaton $\mathcal{A}$ (for a binary signature) are pairs (a, q) in $A \times Q$ and 4-tuples (q_1, q_2, f, q) in $Q \times Q \times F \times Q$ that define total mappings from A and from $Q \times Q \times F$ to Q. A *run* of $\mathcal{A}$ on a term t is a total mapping $run_{\mathcal{A}}$ from $Occ(t)$ to Q that satisfies the transitions. It is *accepting* if $run_{\mathcal{A}}(Root(t))$ is an *accepting state.*

We let $L_{\varphi(X_1,...,X_k)}[t]$ be the finite set of annotated terms $t[U_1, ..., U_k]$ for all k-tuples $(U_1, ..., U_k)$ in $Sat(S_t, \varphi(X_1, ..., X_k))$ (it may be empty). Our problem is thus to enumerate the set $\{NL(\overline{t}) \mid \overline{t} \in L_{\varphi(X_1,...,X_k)}[t]\}$. We define $L_{\varphi(X_1,...,X_k)}$ as the union of the sets $L_{\varphi(X_1,...,X_k)}[t]$ for all $t \in T(F,A)$. This set is recognizable by a classical result of Doner, Thatcher and Wright [6, 16] (see the book chapter by Thomas [17]). An automaton $\mathcal{A}$ recognizing it can be built from $\varphi(X_1, ..., X_k)$.

For every term t in $T(F,A)$, we now define an AND/OR-DAG $G_{\mathcal{A}}(t)$ that embeds the terms $t[U_1, ..., U_k]$ for all $(U_1, ..., U_k)$ in $Sat(S_t, \varphi(X_1, ..., X_k))$.

Definition 3 : *The AND/OR-DAG associated with a term t.*

We let Q be the set of states of $\mathcal{A}$ and Q_{Acc} be its set of accepting states.

We define V^{and}, the set of AND-vertices of $G_{\mathcal{A}}(t)$ as $(Occ_A(t) \times \mathbf{B}^k) \cup (Occ_F(t) \times Q \times Q)$. We define V^{or}, the set of OR-vertices of $G_{\mathcal{A}}(t)$ as $Occ(t) \times Q$.

We define the edges as follows :

1. If u is an occurrence of a constant a, then we define edges:
   $(u,q) \longrightarrow (u,\alpha)$ for all α such that $((a,\alpha),q)$ is a transition of $\mathcal{A}$.
2. If u is an occurrence of a function symbol f, then we define edges :
   $(u,q) \longrightarrow (u,q_1,q_2)$ for all q_1,q_2 such that (q_1,q_2,f,q) is a transition of $\mathcal{A}$.

All these edges go from an OR-vertex to an AND-vertex.

3. We also define edges from AND-vertices to OR-vertices :
   $(u,q_1,q_2) \longrightarrow (u_1,q_1)$ and $(u,q_1,q_2) \longrightarrow (u_2,q_2)$ if u_1 and u_2 are the first and second sons of u.

This graph is a directed graph without circuits by construction. Its leaves are AND-vertices. A linear order on edges outgoing from a vertex (u,q) can be defined from the lexicographical ordering on $\mathbf{B}^k$ and a lexicographic ordering on $Q\times Q$ based on a linear order on Q. We also define the edge $(u,q_1,q_2) \longrightarrow (u_1,q_1)$ as smaller than the edge $(u,q_1,q_2) \longrightarrow (u_2,q_2)$.

Remark 4 : Without being a tree (because an OR-vertex may have indegree more than 1) this DAG satisfies Condition (c2) of Definition 2 : if in this graph we have a directed path from x to y, then the first components of x and y are occurrences u and v in the term t, and u is an ancestor of v in t or is equal to v. Furthermore, an AND-vertex x with two sons y_1 and y_2 is of the form (u,q_1,q_2), the sons $y_1=(u_1,q_1)$ and $y_2=(u_2,q_2)$ correspond to the two sons u_1 and u_2 of u in t. The vertices in $G_{\mathcal{A}}(t)\downarrow y_1$ and in $G_{\mathcal{A}}(t)\downarrow y_2$ correspond to occurrences in the subterms $t\downarrow u_1$ and $t\downarrow u_2$ of t rooted at u_1 and u_2. Hence they have no vertex in common.

Assuming that t has n occurrences of constants and $n-1$ occurrences of binary symbols, we note that $G_{\mathcal{A}}(t)$ has $n\cdot 2^k+(n-1)\cdot Card(Q)^2+ \quad (2n-1)\cdot Card(Q)=O(n)$ vertices, $(3\cdot n-2)\cdot Card(Q)^2+n\cdot 2^k=O(n)$ edges and can be constructed from t in time $O(n)$.

Lemma 4 : Let $t\in T(F,A)$ and $T\in EMB(G_{\mathcal{A}}(t),(Root(t),p))$. The constants labelling the leaves of T define a term $\overline{t}$ in $T(F,A\times\mathbf{B}^k)$ with underlying term t. The OR-vertices of T are pairs in $Occ(t)\times Q$ and these pairs define the unique run of $\mathcal{A}$ on $\overline{t}$ and futhermore $p=run_{\mathcal{A}}(Root(\overline{t}))$. Conversely, on every term $\overline{t}$ in $T(F,A\times\mathbf{B}^k)$ the automaton $\mathcal{A}$ has a unique run associated with a unique tree $T\in EMB(G_{\mathcal{A}}(t),(Root(t),p))$ for some $p\in Q$.

Proof : Clear from the definitions. □

It follows that the set $EMB^{Acc}(G_{\mathcal{A}}(t))$ defined as the union of the sets $EMB(G_{\mathcal{A}}(t),(Root(t),p))$ for p in Q_{Acc} is in bijection with $L_{\varphi(X_1,\ldots,X_k)}[t]$. Hence, the set $L_{\varphi(X_1,\ldots,X_k)}[t]$ can be enumerated with linear delay by Theorem 1 (and the remark following it). But as in the case of words, we do not obtain a linear delay enumeration algorithm for the set $Sat(S_t,\varphi(X_1,\ldots,X_k))$. To obtain one, we will again perform a kind of ε-reduction.

A leaf of $G_{\mathcal{A}}(t)$, or of a tree embedded in $G_{\mathcal{A}}(t)$ is a *λ-leaf* if it is of the form (u, λ), where $\lambda = (False, ..., False)$.

If T is embedded in G we denote by $NL(T)$ the set of its leaves that are not λ-leaves. The set $Sat(S_t, \varphi(X_1, ..., X_k))$ is in bijection with $NL(EMB^{Acc}(G_{\mathcal{A}}(t)))$ defined as the set of sets $NL(T)$ for trees T in $EMB^{Acc}(G_{\mathcal{A}}(t))$. This latter bijection preserves sizes up to constant multiplicative factors. Our objective is now to transform $G_{\mathcal{A}}(t)$ into an AND/OR-DAG H with a vertex r such that :

$$NL(EMB(H, r)) = NL(EMB^{Acc}(G_{\mathcal{A}}(t)))$$

and the size of a tree T in $EMB(H, r)$ is proportional to $NL(T)$.

Definitions 4 : *Transformations of* $G_{\mathcal{A}}(t)$.

First step : We add to $G_{\mathcal{A}}(t)$ a new OR-vertex r and edges from r to the vertices $(Root(t), p)$ for all accepting states p and we obtain an AND/OR-DAG $G_1(t)$. Thus the enumeration problem reduces to enumerating $EMB(G_1(t), r)$. The construction of $G_1(t)$ can be done from $G_{\mathcal{A}}(t)$ in constant time.

Second step : Let us define a *λ-tree* as a tree T in $EMB(G_1(t), u)$ for some u, having only λ-leaves. Let us say that a vertex x of $G_1(t)$ is of type **A** (for *always*) if all the trees in $EMB(G_1(t), x)$ are λ-trees. It is of type **N** (for *never*) if no tree in $EMB(G_1(t), x)$ is a λ-tree. It is of type **S** (for *sometimes*) otherwise.

The second step consists in marking the vertices of $G_1(t) \downarrow r$ with their types **N**,**S** or **A**. This can be done during a depth-first traversal of $G_1(t) \downarrow r$ starting and ending at r, hence in time $O(\mid t \mid)$, using the procedure DFT described in Section 2, and the following computation rules for types. The type of a λ-leaf is **A**, that of another leaf is **N**. The type of an OR-vertex is **N** if all its sons have type **N**, **A** if all its sons have type **A** and **S** in all other cases. The type of an AND-vertex is **N** if one of its sons has type **N**, **A** if all its sons have type **A** and **S** in all other cases.

One can also delete the λ-leaves because the information they contain is now included in the types of OR-vertices. Some OR-vertices may become leaves.

It r is of type **A**, then one can report that $(\varnothing, ..., \varnothing)$ is the only satisfying assignment and stop.

It r is of type **S**, then $(\varnothing, ..., \varnothing)$ will be one result, say the first one.

Third step :

1. For each AND-vertex of the form (u, q_1, q_2) we do the following concerning its two sons (u_1, q_1) and (u_2, q_2) and its father (u, q) :

(a) if (u_1, q_1) and (u_2, q_2) are both of type **A** we delete the AND-vertex (u, q_1, q_2) ;

(b) otherwise

if (u_1, q_1) is of type **S** or **A** we add an ε-edge (i.e., an edge labelled by ε) from (u, q) to (u_2, q_2) ;

if (u_2, q_2) is of type **S** or **A** we add an ε-edge from (u, q) to (u_1, q_1) ;

in both cases, if (u_1, q_1) or (u_2, q_2) is of type **A** we delete the AND-vertex (u, q_1, q_2).

2. We delete the vertices of type **A**, and then, the vertices not reachable from r by a directed path. We let $G_2(t)$ be the graph obtained in this way in time $O(\mid t \mid)$.

Claim 1 : $G_2(t)$ is a locally ordered AND/OR dag.

Proof : Can an OR-vertex of $G_2(t)$ be a leaf ? No because this vertex would be of type **A** (because a vertex having a son not of type **A** is never deleted) and all vertices of type **A** are finally deleted.

For each u at most one of the nodes (u, q) has type **S** or **A** because this implies that there is a run of $\mathcal{A}$ on the term $(t \downarrow u)[\varnothing, ..., \varnothing]$ yielding state q at u, but $\mathcal{A}$ is deterministic hence, there is at most one such state q. It follows that we do not create parallel edges, the origin of which is an OR-vertex (in case (b), if we would create two parallel edges from (u, q) to (u_2, q_2) this would mean that two pairs (u_1, q_1) and (u_1, q_1') are of type **S** or **A**.) Since the occurrences of t can be linearly ordered, we have a linear order on the set of edges. Hence, $G_2(t)$ is locally ordered.

Condition (c2) of Definition 2 holds for $G_1(t)$, hence also for $G_2(t)$ as one checks easily. □

Claim 2 : $NL(EMB(G_2(t), r)) = NL(EMB(G_1(t), r))$.

Proof : The deleted vertices are all of type **A** ; the corresponding subtrees are λ-trees and do not contribute to the sets in $NL(EMB(G_1(t), r))$. They need not be explored, hence they can be deleted.

Deleting them does not block explorations because in case (b) of the third step, if (u_1, q_1) is of type **A** we add an ε-edge from (u, q) to (u_2, q_2). Hence, the subgraph $G_1(t) \downarrow (u_2, q_2)$ will be explored from (u, q) through the ε-edge from (u, q) to (u_2, q_2). (And similarly if we exchange (u_1, q_1) and (u_2, q_2).) It follows that $NL(EMB(G_2(t), r)) \supseteq NL(EMB(G_1(t), r))$.The other inclusion is proved by a similar argument.□

Claim 3 : The trees in $EMB(G_2(t), r)$ have no λ-leaves.

Proof : Clear because we have deleted all vertices of type **A**.□

Since a tree T in $EMB(G_2(t), r)$ may contain "long" paths consisting of ε-edges, its size is not proportional to that of $NL(T)$. Hence we need a fourth step which, as in the case of words, consists in replacing sequences of ε-edges by direct edges.

Fourth step : The ε-edges we want to eliminate are all between OR-vertices and are of the form $(v, p) \longrightarrow (u, q)$ where v is the father of u.

(1) For every AND-vertex that is a leaf (u, α) we do the following :

For every OR-vertex (v, p) such that there is a directed path of ε-edges from (v, p) to (u, q) and an edge from (u, q) to (u, α) we create an edge from (v, p) to (u, α).

(2) For every AND-vertex with two sons, hence of the form (u, q_1, q_2) we do the following :

For every OR-vertex (v, p) such that there is a directed path of ε-edges from (v, p) to (u, q) and an edge from (u, q) to (u, q_1, q_2) we create an edge from (v, p) to (u, q_1, q_2).

(3) We delete all ε-edges.

We denote by $G_3(t)$ the AND/OR-DAG constructed by this step. Like $G_2(t)$ it is locally ordered from the global linear ordering on its set of vertices.

Claim 4 : The number of edges created by this step is $O(\mid t \mid \cdot ht(t))$. The fourth step can be performed in time $O(\mid t \mid \cdot ht(t))$.

Proof : We create edges from (v, p) to AND-vertices (u, q_1, q_2) or (u, α) only if v is an ancestor of u. Hence, we create at most $Card(Q) \cdot d(u)$ edges where $d(u)$ is the distance of u to the root of t. The number of AND-vertices is $O(\mid t \mid)$ and $d(u) \leq ht(t)$, hence in total at most $O(\mid t \mid \cdot ht(t))$ edges are created. This fourth step can be performed in time $O(\mid t \mid \cdot ht(t))$ by processing $G_2(t)$ from the leaves to the root r, as we did in the proof Theorem 2 to compute the sets $L(n-1), ..., L(0)$. Here is the method : for each occurrence u we define the set $M(u)$ consisting of all pairs (q, w) such that q is a state, w is an AND-vertex and there is an edge from the OR-vertex (u, q) to w, which is either an edge of $G_2(t)$ or a edge added by Step 4. If u has sons v and v'(in t) then $M(u)$ can be computed in time proportional to $Card(M(v)) + Card(M(v'))$. The total time is thus proportional to the sum of cardinalities of the sets $M(u)$ for all occurrences, hence to the number of edges of $G_3(t)$. We obtain thus the bound $O(\mid t \mid \cdot ht(t))$. □

Claim 5 : $NL(EMB(G_3(t), r)) = NL(EMB(G_2(t), r))$

Proof : Clear from the construction. □

Proposition 1 : Let A be a finite set of constants, F be a finite set of binary function symbols and $\varphi(X_1, ..., X_k)$ be an MS formula over the signature $\tau_{F,A}$ such that $Sat(S_t, \varphi(X_1, ..., X_k)) \subseteq \mathcal{P}(Occ_A(t))^k$ for every t in $T(F, A)$.

(1) For every term t in $T(F, A)$, there exists a linear delay enumeration algorithm of the set $Sat(S_t, \varphi(X_1, ..., X_k)) = \{B_1, ..., B_m\}$ with preprocessing time $O(\mid t \mid \cdot ht(t))$.

(2) For each i, the element B_i can be computed in time $O(\mid B_i \mid \cdot \log(\mid t \mid))$, after a preprocessing taking time $O(\mid t \mid \cdot ht(t))$.

Proof : Assumption (a) before Definition 3 is satisfied. We use the above described construction. In time $O(\mid t \mid \cdot ht(t))$ we can transform $G_{\mathcal{A}}(t)$ into an AND/OR-DAG $G_3(t)$ with a root r such that $NL(EMB(G_3(t), r)) = NL(EMB^{Acc}(G_{\mathcal{A}}(t)))$. Furthermore, the size of a tree in $EMB(G_3(t), r)$ is proportional to the size of the k-tuple in $Sat(S_t, \varphi(X_1, ..., X_k))$ that it characterizes since we have eliminated all λ-leaves and all ε-edges introduced at intermediate steps of the construction. This gives the desired result with help of Theorem 1 and the observation that :

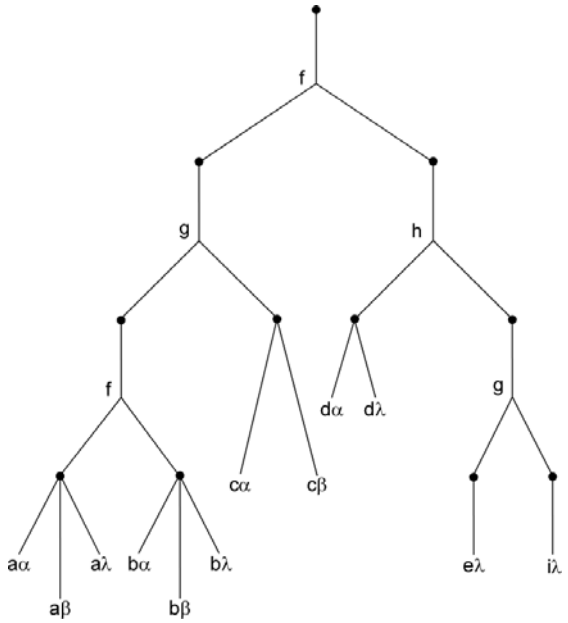

Figure 2: An AND/OR-DAG $G_{\mathcal{A}}(t)$

$$\deg^{OR}(G_3(t)) \leq \mid t \mid \cdot(Card(Q) + 2^k).\ \square$$

Example 3 : We consider a term $f(g(f(a,b),c),h(d,g(e,i))))$. Figure 2 shows a part of a DAG $G_{\mathcal{A}}(t)$. The OR vertices are black dots. Figure 3 shows the **N,S,A** types of OR vertices, the created ε-edges (curved lines) and with dotted lines, the edges deleted at Steps 2,3,4. Figure 4 shows the resulting AND/OR-DAG $G_3(t)$.

Some of the 36 results of the query represented by this DAG are :
$\{a\alpha, b\alpha, c\alpha, d\alpha\}$, $\{a\beta, b\alpha, c\alpha, d\alpha\}$, $\{a\alpha, b\beta, c\alpha, d\alpha\}$,
$\{a\alpha, b\alpha, c\alpha\}$, $\{a\beta, b\alpha, c\alpha\}$, $\{a\alpha, b\beta, c\alpha\}$, $\{a\beta, b\beta, c\alpha\}$,
$\{b\alpha, c\beta, d\alpha\}$, $\{c\alpha\}$, $\{c\beta\}$.$\square$

In the following theorem, we allow F to contain symbols that are not binary. The signature $\tau_{F,A}$ has in this case p binary relations relating a vertex to its sons, where p is the maximal arity of a symbol in F. Furthermore, the free variables of formulas are not restricted to range over sets of occurrences of constants. We will use transformations of structures defined by monadic second-order formulas called *MS transductions* and reviewed in the appendix.

Definition 5 : *Transfering enumeration results from a class to another.*

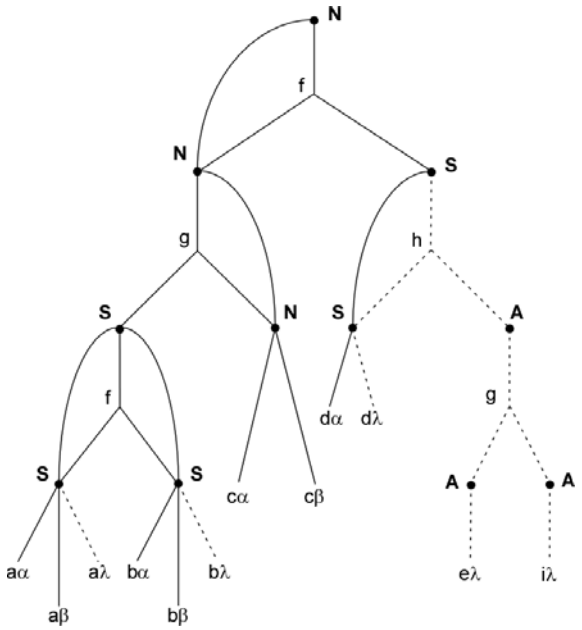

Figure 3: An intermediate step of the transformations

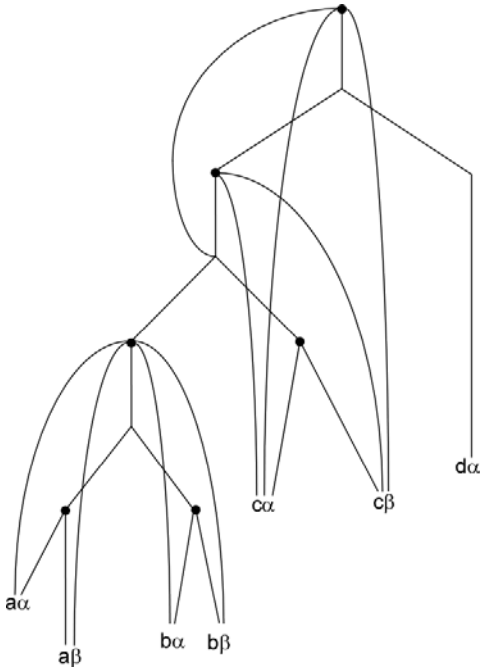

Figure 4: The resulting DAG $G_3(t)$

Let $\mathcal{C}$ and $\mathcal{D}$ be two classes of *concrete* relational structures (not of relational structures up to isomorphism). Let us assume we have a total injective mapping $\gamma : \mathcal{C} \longrightarrow \mathcal{D}$ and a noncopying MS transduction $\delta : \mathcal{D} \longrightarrow \mathcal{C}$ satisfying the following conditions (we denote by D_S the domain of a structure S) :

(1) If $T = \gamma(S)$, then $D_S \subseteq D_T$.

(2) For every S in $\mathcal{C}$, we have $\delta(\gamma(S)) = S$.

We denote this by $\mathcal{C} \rightleftarrows^{\gamma}_{\delta} \mathcal{D}$. From definitions and properties of MS transductions, in particular from the "backwards translation lemma" applied to the transduction δ (see [4] or the appendix for background) we have the following facts :

(3) If $T = \gamma(S)$, then $D_S = \{x \in D_T \mid T \models \theta(x)\}$ for some MS formula $\theta(x)$.

(4) For every MS formula $\varphi(X_1, ..., X_k)$ one can construct an MS formula $\psi(X_1, ..., X_k)$ such that for every S in $\mathcal{C}$ we have, because $S = \delta(\gamma(S))$:

$$Sat(S, \varphi(X_1, ..., X_k)) = Sat(\gamma(S), \psi(X_1, ..., X_k)).$$

If we have $\mathcal{C} \rightleftarrows^{\gamma}_{\delta} \mathcal{D}$ and if the results of an MS query defined by ψ as in (4) for structures $\gamma(S)$ can be enumerated with linear delay, then the same holds for those of an MS query defined by φ in every S in $\mathcal{C}$. The preprocessing time is the sum of that for computing $\gamma(S)$ and of that for the enumeration of the set $Sat(\gamma(S), \psi(X_1, ..., X_k))$.

Note also that $\mathcal{C} \rightleftarrows^{\gamma' \circ \gamma}_{\delta \circ \delta'} \mathcal{D}$ if $\mathcal{C} \rightleftarrows^{\gamma}_{\delta} \mathcal{F}$ and $\mathcal{F} \rightleftarrows^{\gamma'}_{\delta'} \mathcal{D}$.

Theorem 3 : Let A be a finite set of constants, F a finite set of function symbols and $\varphi(X_1, ..., X_n)$ be an MS formula over the signature $\tau_{F,A}$.

(1) For every term t in $T(F, A)$, there exists a linear delay enumeration algorithm of the set $Sat(S_t, \varphi(X_1, ..., X_k)) = \{B_1, ..., B_m\}$ with preprocessing time $O(\mid t \mid \cdot ht(t))$.

(2) For each i, the element B_i can be computed in time $O(\mid B_i \mid \cdot \log(\mid t \mid))$, after a preprocessing taking time $O(\mid t \mid \cdot ht(t))$.

Proof : In order to use Proposition 1, we introduce a new set of binary function symbol F_* a defined as $\{*\} \cup \{*_f \mid f \in F\}$ and we define recursively an injective mapping $\gamma : T(F, A) \longrightarrow T(F_*, F \cup A)$ as follows :

$\gamma(a) = a$ if $a \in A$,

$\gamma(f(t_1, ..., t_m)) = *_f(f, *(\gamma(t_1), *(\gamma(t_2), ..., *(\gamma(t_{m-1}), \gamma(t_m)))...))$.

This transformation is called *curryfication* in Niehren et al. [14]. Informally, a function symbol f of arity m is replaced by the term $*_f(f, *(x_1, *(x_2, ..., *(x_{m-1}, x_m))...)$. (There is a certain redundancy in this encoding, but it will be useful later.) For an example if $t = f(a, g(b, c), h(d))$, then $\gamma(t) = *_f(f, *(a, *(*_g(g, *(b, c)), *_h(h, d))))$.

The function symbols of F are made into constants. If t has height h, then $\gamma(t)$ has height at most $p \cdot h + (1 - p)$, where p is the maximal arity

of a symbol in F. The inverse of γ is a noncopying MS transduction $\delta : T(F_*, F \cup A) \longrightarrow T(F, A)$. In the appendix, we detail its definition (Example 7). The transformation γ is an MS transduction up to isomorphism.

Hence we have $T(F, A) \rightleftarrows_{\delta}^{\gamma} T(F_*, F \cup A)$. The formula $\psi(X_1, ..., X_k)$ which translates "backwards" (with respect to δ) a formula $\varphi(X_1, ..., X_k)$ satisfies the condition that $X_1, ..., X_k$ can only denote sets of occurrences of constants (because function symbols of the original signature F are made into constants in the new signature $(F_*, F \cup A)$).

The computation of $\gamma(t)$ (or rather of $\gamma(S_t)$) can be done in time $O(\mid t \mid)$ for t in $T(F, A)$ hence we get the result by Proposition 1 and the previous remarks. □

Remarks 5 : (1) The efficiency of this algorithm can be measured (cf. Remark 3 (1)) as follows : for a preprocessing time p, we may enumerate $2^{a \cdot p/\log(p)}$ results, which is more than what gives Theorem 2. To see this, we consider the terms t_n defined by $t_1 = b$, $t_{n+1} = f(t_n, t_n)$. With a preprocessing taking time $p = c \cdot n2^n$, we can obtain $2^{k \cdot 2^n}$ results.

(2) *Required space* : For the algorithms of Proposition 1 and Theorem 3, we observe that in both cases, Condition (c2) holds for the DAGs we construct, the trees of which are to be enummerated. This gives a space requirement for the preprocessing proportional to the size of the DAG to be explored, that is $O(\mid t \mid .ht(t))$. For the direct generation algorithm, one needs the same space, by Remark 2. The reduction used for proving Theorem 3 preserves these values.

Words over an alphabet A can be handled as terms over a signature consisting of a set A of unary function symbols and a constant denoting the empty word. In this case, the height of a word is its length plus 1 and Theorem 3 gives Theorem 2 (we omit some technical details necessary for deriving rigorously Theorem 2 from Theorem 3). Clearly, the construction of Theorem 2 is more direct. However, words can also be handled as terms in $T(\{\bullet\}, A)$ where $\bullet$ is the binary concatenation operation. A word of length n can be expressed by a term of height $\lceil log(n) \rceil$, and Theorem 3 gives for words the following better result than Theorem 2 :

Corollary 1 : Let A be a finite alphabet and $\varphi(X_1, ..., X_k)$ be an MS formula over the signature σ_A.

(1) For every word w in A^+ there exists a linear delay enumeration algorithm of the set $Sat(S_w, \varphi(X_1, ..., X_k))$ using a preprocessing taking time $O(\mid w \mid . \log(\mid w \mid))$.

(2) For each i, the i-th element B_i of this set can be computed in time $O(\mid B_i \mid \cdot \log(\mid w \mid))$. The preprocessing takes time $O(\mid w \mid . \log(\mid w \mid))$.

A similar improvement can be done for the case of terms with function symbols of arbitrary arity. We will use the following result by Courcelle and Vanicat [5].

Proposition 2 : Let A be a finite set of constants, F a finite set of binary function symbols. Let $\circ$ be a new binary symbol and $\#$ be a new constant. There exists a mapping $\mu : T(F, A) \longrightarrow T(F \cup \{\circ\}, A \cup \{\#\})$ such that for every $t \in T(F, A)$ we have :

(1) $\mid \mu(t) \mid \leq 2 \cdot \mid t \mid$, $Occ_x(\mu(t)) = Occ_x(t)$ for every $x \in F \cup A$, and $ht(\mu(t)) \leq 3 \cdot \log(\mid t \mid) + 1$.

(2) Its inverse is a noncopying MS transduction $\nu : T(F \cup \{\circ\}, A \cup \{\#\}) \longrightarrow T(F, A)$.

(3) $\mu(t)$ is computable in time $O(\mid t \mid \cdot \log(\mid t \mid))$.

Proof sketch :

In order to give the definition of ν we define a *context* as a term in $T(F, A \cup \{u\})$ having one and only one occurrence of a special variable u. Then ν is the partial mapping : $T(F \cup \{\circ\}, A \cup \{\#\}) \longrightarrow T(F, A \cup \{u\})$ defined inductively as follows :

$\nu(a) = a$ if $a \in A$,

$\nu(\#) = u$,

$\nu(f(s,t)) = f(\nu(s), \nu(t))$, if $f \in F$, $\nu(s)$ and $\nu(t)$ are both defined, both are in $T(F, A)$ or one is in $T(F, A)$ and the other is a context,

$\nu(\circ(s,t)) = \nu(s)[\nu(t)/u]$, if $\nu(s)$ and $\nu(t)$ are both defined, $\nu(s)$ is a context and $\nu(t)$ is in $T(F, A)$ or is a context, and $\nu(s)[\nu(t)/u]$ denotes the substitution of $\nu(t)$ for the unique occurrence of u in $\nu(s)$,

$\nu(t)$ is undefined if none of the above clauses is applicable.

For an example, if $t = \circ(\circ(f(a, \#), f(b, \#)), f(c, d)))$ then $\nu(t) = f(a, f(b, f(c, d)))$.

The mapping μ is an inverse of ν that transforms a term into a "balanced" one, of logarithmic height in its size. It is defined in [5], Theorem 1. That ν is an MS transduction is Theorem 2 of [5]. The equality $Card(Occ_x(\mu(t))) = Card(Occ_x(t))$ for every $x \in F \cup A$ is clear from the definition of ν. One can designate concretely the occurrences in terms $\mu(t)$ in such a way that one has the equality $Occ_x(\mu(t)) = Occ_x(t)$. $\square$

Theorem 4 : Let A be a finite set of constants, F a finite set of function symbols and $\varphi(X_1, ..., X_k)$ be an MS formula over the signature $\tau_{F,A}$.

(1) For every term t in $T(F, A)$ there exists a linear delay enumeration algorithm of the set $Sat(S_t, \varphi(X_1, ..., X_k))$ with preprocessing time $O(\mid t \mid \cdot \log(\mid t \mid))$.

(2) For each i, the i-th element B_i of this set can be computed in time $O(\mid B_i \mid \cdot \log(\mid t \mid))$, where $\mid B_i \mid$ is its size. The preprocessing takes time $O(\mid t \mid \cdot \log(\mid t \mid))$.

Proof : We have $T(F, A) \rightleftarrows_{\delta}^{\gamma} T(F_*, F \cup A)$ by the proof of Theorem 3, and $T(F_*, F \cup A) \rightleftarrows_{\nu}^{\mu} T(F_* \cup \{\circ\}, F \cup A \cup \{\#\})$ by Proposition 2.

The preprocessing time for $t \in T(F, A)$ is :

$$O(\mid t \mid) + \ O(\mid \gamma(t) \mid \cdot \log(\mid \gamma(t) \mid)) + \ O(\mid \mu(\gamma(t)) \mid \cdot ht(\mu(\gamma(t))))$$

which gives $O(\mid t \mid \cdot \log(\mid t \mid))$ because $\mid \gamma(t) \mid \leq k \cdot \mid t \mid$ and by (2) of Proposition 2 we have $\mid \mu(\gamma(t)) \mid \leq 2 \cdot \mid \gamma(t) \mid = O(\mid t \mid)$ and $ht(\mu(\gamma(t))) \leq 3 \cdot \log(\mid \gamma(t) \mid) + 1 = O(\log(\mid t \mid))$.□

Remark 6 : *Required space* : The reduction used for proving Theorem 4 preserves the values obtained for Theorem 3. The space requirement is $O(\mid t \mid \cdot \log(\mid t \mid))$ in both cases.

By using Proposition 2, we have reduced the preprocessing time by reorganizing any tree into a *balanced* one, i.e., a tree of height that is logarithmic in its size. This applies to all trees. We now describe another way to reduce the preprocessing time. It does not apply to all trees, but only to those having "few distinct subtrees". We recall from Definition 2 (d) that the unfolding of an AND-DAG G starting from a vertex x yields a tree denoted by $Unf(G, x)$ which may have exponentially more vertices than G.

Definitions 6 :

(a) We denote by $D(F, A)$ the set of AND-DAGs H satisfying the following conditions :

(a1) All vertices are AND-vertices and are reachable from a unique vertex called the *root*, denoted by $Root(H)$.

(a2) Each vertex x has a label in $F \cup A$, the arity of this label is equal to the outdegree of x.

(b) Every H in $D(F, A)$ unfolds into a term $Unf(H, Root(H))$ in $T(F, A)$ denoted in a simpler way by $Unf(H)$. A DAG H in $D(F, A)$ is defined as locally ordered and the ordering of edges with origin a same vertex defines the order of arguments of function symbols in the term $Unf(H)$.

(c) If all symbols in F have arity 2 we can apply to DAGs in $D(F, A)$ the constructions of Definitions 3. That is, for a complete deterministic automaton $\mathcal{A}$ recognizing a subset of $T(F, A \times \mathbf{B}^k)$ and a DAG H in $D(F, A)$, we construct an AND/OR-DAG $G_{\mathcal{A}}(H)$ by slight modifications in the construction of $G_{\mathcal{A}}(t)$ given in Definition 3.

We let $V_{H,A}$ be the set of vertices of H having a label in A, and $V_{H,F}$ be the set of those having a label in F. We let Q be the set of states of $\mathcal{A}$, and Q_{Acc} be its set of accepting states.

We define the set of AND-vertices of $G_{\mathcal{A}}(H)$ as $(V_{H,A} \times \mathbf{B}^k) \cup (V_{H,F} \times Q \times Q)$ and its set of OR-vertices as $V_H \times Q$. The edges are as follows :

1. If u in $V_{H,A}$ has label a we define edges

$(u, q) \longrightarrow (u, \alpha)$ for all α such that $((a, \alpha), q)$ is a transition of $\mathcal{A}$.

2. If u in $V_{H,F}$ has label f we define edges :

2.1. $(u, q) \longrightarrow (u, q_1, q_2)$ for all q_1, q_2 such that (q_1, q_2, f, q) is a transition of $\mathcal{A}$. We also define edges from AND-vertices to OR-vertices :

2.2. $(u, q_1, q_2) \longrightarrow (u_1, q_1)$, defined as the first outgoing edge, and

2.3. $(u, q_1, q_2) \longrightarrow (u_2, q_2)$, defined as the second outgoing edge, where u_1 and u_2 are the first and second sons of u. (They can be identical, this is why we specify a first and a second edge).

This graph has no circuit by construction. It is locally ordered (we use linear orders on the sets Q and $\mathbf{B}^k$ to order edges going out of OR-vertices), hence it is an AND/OR-DAG.

Lemma 5 : Let $H \in D(F, A), t = Unf(H)$ and $T \in EMB(G_{\mathcal{A}}(H), (Root(H), p))$. The constants labelling the leaves of T define a term $\overline{t}$ in $T(F, A \times \mathbf{B}^k)$ with underlying term t. The OR-vertices of T are pairs in $Occ(t) \times Q$ and these pairs define the unique run of $\mathcal{A}$ on $\overline{t}$ and futhermore $p = run_{\mathcal{A}}(Root(\overline{t}))$. Conversely, the unique run of $\mathcal{A}$ on a term $\overline{t}$ in $T(F, A \times \mathbf{B}^k)$ is associated in this way with a unique tree $T \in EMB(G_{\mathcal{A}}(H), (Root(H), p))$ for some $p \in Q$.

Our objective is to enumerate the set $\{NL(\overline{t}) \mid \overline{t} \in L_{\varphi(X_1,...,X_k)}[t]\}$ where $t = Unf(H)$. There is however a difficulty. In a pair $(u, \alpha) \in NL(\overline{t})$, the occurrence u is not just a vertex of H because a vertex of H yields (in general) several occurrences in t. Hence, we must specify in some way *and compute* the occurrences u in the pairs (u, α) in $NL(\overline{t})$. We could enumerate with linear delay the set $L_{\varphi(X_1,...,X_k)}[t]$ but its elements may contain a lot of useless information and the size of $\overline{t}$ is not $O(\mid NL(\overline{t}) \mid)$. We will actually delete the useless parts from the terms $\overline{t}$.

We let $\perp$ be a new constant. For each $\overline{t}$ in $T(F, A \times \mathbf{B}^k)$, we define a term $\overline{t}_\perp$ in $T(F, A \times (\mathbf{B}^k - \{\lambda\}) \cup \{\perp\})$ by replacing every subterm of $\overline{t}$ that belongs to $T(F, A \times \{\lambda\})$ by $\perp$. Informally, $\overline{t}_\perp$ is the union of all paths in $\overline{t}$ from the root to a "useful" leaf, i.e., one that is not an occurrence of (a, λ). We may have $\overline{t}_\perp = \perp$ if the encoded tuple is $(\varnothing, ..., \varnothing)$.

Example 4 : We consider terms in $T(\{f\}, \{a\})$ where f is binary. We let $\varphi(X)$ express that in a term $t = f(t_1, t_2)$ the set X has exactly two elements, one in $Occ_a(t_1)$ and the other in $Occ_a(t_2)$. We consider the term $t = f(f(f(a, a), f(a, a)), f(f(a, a), f(a, a)))$ described by the DAG $H = x_f \rightrightarrows y_f \rightrightarrows z_f \rightrightarrows u_a$. (The subscripts in x_f, y_f, z_f, u_a indicate the labels of vertices x, y, z, u). We let $\alpha = (True)$ and $\lambda = (False)$.

Three of the terms $\overline{t}_\perp$ associated with the 16 terms $\overline{t}$ in $L_{\varphi(X_1,...,X_k)}[t]$ are:
$f(f(\perp, f(\perp, a)), f(\perp, f(\perp, a)))$,
$f(f(f(\perp, a), \perp), f(\perp, f(\perp, a)))$,
$f(f(\perp, f(a, \perp)), f(\perp, f(\perp, a)))$,
where for clarity, we write a instead of (a, α).□

We will enumerate the set $SAT(S_{Unf(H)}, \varphi(X_1, ..., X_n))$ defined as $\{\overline{t}_\perp \mid \overline{t} \in L_{\varphi(X_1,...,X_k)}[Unf(H)]\}$. We take the size of $\overline{t}_\perp$ as size measure for the results of our desired enumeration algorithm (the notation $SAT(S_{Unf(H)}, \varphi(...))$ emphasizes the change of size measure.) This is justified because the term t is not

given directly : it must be computed from H. The notion of linear delay enumeration is less demanding in this case because the sizes of answers are larger. We get the following analog of Theorem 3, that uses a simpler and quicker preprocessing.

Theorem 5 : Let (F, A) be a signature, and $\varphi(X_1, ..., X_k)$ be an MS formula over the relational signature $\tau_{F,A}$.

(1) For every H in $D(F, A)$ there exists a linear delay enumeration algorithm of the set $SAT(S_{Unf(H)}, \varphi(X_1, ..., X_k))$. The preprocessing takes time $O(Card(V_H))$.

(2) For each i, the i-th element t_i of this set can be computed in time $O(\mid t_i \mid)$. The preprocessing takes time $O(Card(V_H))$.

Proof, First part: As for Theorem 3 we first consider the particular case where the symbols in F are binary and the variables $X_1, ..., X_k$ can only denote sets of leaves. We modify as follows the constructions of Definition 4. We apply the first two steps to $G_{\mathcal{A}}(H)$ instead of to $G_{\mathcal{A}}(t)$. Then we remove all vertices not accessible from the root r by a directed path. We denote by $G_1(H)$ the DAG obtained in this way. By Lemma 5 we have a bijection of $EMB(G_1(H), r)$ onto $L_{\varphi(X_1,...,X_k)}[t]$ where $t = Unf(H)$. This bijection can be defined as the restriction of a mapping b from $\bigcup\{EMB(G_1(H), w) \mid w \in V_{G_1(H)}\}$ into $T(F, A \times \mathbf{B}^k)$ defined by the following recursion, where $lab(u)$ denotes the label in $F \cup A$ of a vertex u of H, and $T \in EMB(G_1(H), w)$:

1. If w is a leaf, $T = w = (u, \alpha)$ for some $u \in V_H$, $\alpha \in \mathbf{B}^k$. Then $b(T) = (a, \alpha)$ where $a = lab(u) \in A$.

2. If w is an AND-vertex of the form (u, q_1, q_2) with outgoing edges $(u, q_1, q_2) \longrightarrow (u_1, q_1)$, $(u, q_1, q_2) \longrightarrow (u_2, q_2)$, then $T = w(T_1, T_2)$ for some $T_1 \in EMB(G_1(H), (u_1, q_1))$ and $T_2 \in EMB(G_1(H), (u_2, q_2))$. Then $b(T) = f(b(T_1), b(T_2))$ where $f = lab(u) \in F$.

3. If w is an OR-vertex then $T = w(T')$ with $T' \in EMB(G_1(H), w')$ for some edge $w \longrightarrow w'$ in $G_1(H)$ and then $b(T) = b(T')$.

In Lemma 5, for $T \in EMB(G_{\mathcal{A}}(H), (Root(H), p))$ the associated term $\overline{t}$ is $b(T)$. Then we transform $G_1(H)$ as in the third step of Definition 4 except that the created edges are labelled by 1 or 2 instead of by ε, and there will be no ε-reduction step. Here is the construction.

(a) For each AND-vertex of the form (u, q_1, q_2) we do the following concerning its two sons (u_1, q_1) and (u_2, q_2) and its father (u, q) :

(a1) if (u_1, q_1) and (u_2, q_2) are both of type **A** we delete (u, q_1, q_2) ;

(a2) otherwise

if (u_1, q_1) is of type **S** or **A**, then we add an edge labelled by 2 from (u, q) to (u_2, q_2) ;

if (u_2, q_2) is of type **S** or **A**, then we add an edge labelled by 1 from (u, q) to (u_1, q_1) ;

in both cases, if (u_1, q_1) or (u_2, q_2) is of type **A** we delete (u, q_1, q_2).

(b) We delete the vertices of type **A**, and then, the vertices not reachable from the root by a directed path.

By Assumption (b) made before Definition 3, there is no vertex $(Root(H), q)$ of type **A** or **S** where q is accepting, because this would indicate that $(\varnothing, ..., \varnothing)$ is an answer to the considered query, which Assumption (b) excludes.

We let $G_2(H)$ be the graph obtained in this way in time $O(Card(V_H))$. We now comment some cases of this construction. In Case (a1), (u, q_1, q_2) has type **A**, there is a unique tree T in $EMB(G_1(H), (u, q_1, q_2))$ and $b(T) \in T(F, A \times \{\lambda\})$. The OR-vertex (u, q) is of type **A** or **S**. Its type indicates that there is a tree T in $EMB(G_1(H), (u, q))$ such that $b(T) \in T(F, A \times \{\lambda\})$. We loose no information by deleting (u, q_1, q_2).

In Case (a2), if an edge labelled by 2 is added from (u, q) to (u_2, q_2) this means that there is a tree T_1 in $EMB(G_1(H), (u_1, q_1))$ such that $b(T_1)$ is in $T(F, A \times \{\lambda\})$. Hence, $EMB(G_1(H), (u, q))$ contains trees of the form $T = (u, q)((u, q_1, q_2)(T_1, T_2))$, such that $T_1 \in EMB(G_1(H), (u_1, q_1))$, $b(T_1) \in T(F, A \times \{\lambda\})$, $T_2 \in EMB(G_1(H), (u_2, q_2))$. For such T_1 and those T_2 such that $b(T_2) \notin T(F, A \times \{\lambda\})$, we obtain $b(T) = f(\bot, b(T_2))$ where $f = lab(u)$. We will use the labelled edges to produce directly occurrences of the constant $\bot$, without needing to explore λ-subtrees. As observed in Claim 1 we do not create two parallel edges with same label whose origin is an OR-vertex (in Case (a2)). However, we may create two parallel edges, one with label 1 and one with label 2. See for an example the right part of Figure 5.

We claim that there is a bijection of $EMB(G_2(H), r)$ onto $SAT(S_{Unf(H)}, \varphi(X_1, ..., X_k))$ where r is the root of $G_3(H)$. We define a mapping $\overline{b}$ of $\bigcup\{EMB(G_2(H), w) \mid w \in V_{G_3(H)}\}$ into $T(F, (A \times (\mathbf{B}^k - \{\lambda\})) \cup \{\bot\})$ that will give the desired bijection. Its definition is similar to that of b given above.

1. If w is a leaf, $T = w = (u, \alpha)$ for some $u \in V_H$, $\alpha \in \mathbf{B}^k - \{\lambda\}$. Then $\overline{b}(T) = (lab(u), \alpha)$.

2. If w is an AND-vertex of the form $w = (u, q_1, q_2)$, with edges $(u, q_1, q_2) \longrightarrow (u_1, q_1)$, $(u, q_1, q_2) \longrightarrow (u_2, q_2)$, then $T = w(T_1, T_2)$ for some $T_1 \in EMB(G_2(H), (u_1, q_1))$ and $T_2 \in EMB(G_2(H), (u_2, q_2))$, then $\overline{b}(T) = f(\overline{b}(T_1), \overline{b}(T_2))$ where $f = lab(u)$.

3. If w is an OR-vertex, then $T = w(T')$ with $T' \in EMB(G_2(H), w')$, for some edge $w \longrightarrow w'$ in $G_2(H)$, and we distinguish 3 cases :

3.1 : This edge is labelled by 1, we are in Case (a2) with $w = (u, q)$, $w' = (u_1, q_1)$, (u_2, q_2) is of type **S** or **A** , then we let $\overline{b}(T) = f(\overline{b}(T'), \bot)$ where $f = lab(u)$.

3.2 : This edge is labelled by 2, we are in Case (a2) with $w = (u, q)$, $w' = (u_2, q_2)$, (u_1, q_1) is of type **S** or **A**, then we let $\overline{b}(T) = f(\bot, \overline{b}(T'))$ where $f = lab(u)$.

3.3 : This edge is not labelled, then $\overline{b}(T) = \overline{b}(T')$.

The restriction of $\overline{b}$ to $EMB(G_2(H), r)$ is a bijection onto $SAT(S_{Unf(H)}, \varphi(X_1, ..., X_k))$ for the following reasons. The mapping b defines a bijection of

$EMB(G_1(H), r)$ onto $L_{\varphi(X_1,...,X_k)}[t]$. A tree in $EMB(G_2(H), r)$ is obtained from one in $EMB(G_1(H), r)$ by deleting λ-subtrees, and the edges labelled by 1 and 2 keep track of these deletions. The terms in $SAT(S_{Unf(H)}, \varphi(X_1, ..., X_k))$ are obtained in a similar way from those in $L_{\varphi(X_1,...,X_k)}[t]$ by the replacement of λ-subterms by the constant $\perp$, and the mapping $\overline{b}$ uses the edges labelled by 1 and 2 to produce the appropriate occurrences of these constants.

The preprocessing which consists in building $G_2(H)$ from H and $\mathcal{A}$ takes time $O(Card(V_H))$. (There is no ε-reduction step to perform). We obtain a linear delay algorithm enumerating $EMB(G_2(H), r)$. We obtain also one for $SAT(S_{Unf(H)}, \varphi(X_1, ..., X_k))$ because $\overline{t}_\perp$ is easily computable from T in $EMB(G_2(H), r)$ such that $\overline{b}(T) = \overline{t}_\perp$, and we have $\mid \overline{t}_\perp \mid = \Theta(\mid T \mid)$.

The outdegree of an OR-vertex is bounded by $Card(Q)^2$ in $G_1(H)$, and by $Card(Q)^2 + 2 \cdot Card(Q)$ in $G_2(H)$ (because of Case (a2)). Assertion (2) of Theorem 1 yields a linear delay for the generation of the i-th element.

Example 4 (Continued) : The automaton $\mathcal{A}$ corresponding to $\varphi(X)$ has states p, q, r, s such that for every $\overline{t}$ in $T(\{f\}, \{a\} \times \{\alpha, \lambda\})$ encoding a subset X of $Occ(t)$ where $t \in T(\{f\}, \{a\})$:

$run_{\mathcal{A}}(\overline{t}) = p$ if and only if t has no occurrence in X,

$run_{\mathcal{A}}(\overline{t}) = q$ if and only if t has one occurrence in X which is a leaf,

$run_{\mathcal{A}}(\overline{t}) = r$ if and only if the condition defined by $\varphi(X)$ holds in t, hence r is accepting,

and s is a "sink" state. The transitions are :

$((a, \alpha), q), ((a, \lambda), p)$,

$(p, p, f, p), (p, q, f, q), (q, p, f, q), (q, q, f, r)$,

(x, y, f, s) for any x and y with $(x, y) \notin \{(p, p), (p, q), (q, p), (q, q)\}$.

We consider the same DAG H as before, which unfolds into $t = f(f(f(a, a), f(a, a)), f(f(a, a), f(a, a)))$.

The DAG $G_1(H)$ is shown on the left part of Figure 5. The OR-vertices are black dots, and the associated states p, q, r are shown. The vertices with state p are of type **A**, the others are of type **N**. The DAG $G_2(H)$ is shown to the right with the same conventions. From it one gets the 16 answers to the considered query. □

Proof of Theorem 5, second part:

We now consider the general case where the symbols in F are not necessarily binary and the variables $X_1, ..., X_k$ can denote arbitrary subsets of $Occ(t)$ for $t \in T(F, A)$. A k-tuple $(U_1, ..., U_k)$ of subsets of $Occ(t)$ is encoded by a term $t[U_1, ..., U_k]$ in $T(F \times \mathbf{B}^k, A \times \mathbf{B}^k)$ that we call an annotation of t. The "useful part" of a term s in $T(F \times \mathbf{B}^k, A \times \mathbf{B}^k)$ is the term $s_\perp$ in $T(F \times \mathbf{B}^k, A \times (\mathbf{B}^k - \{\lambda\}) \cup \{\perp\})$ defined recursively as follows :

1. $s_\perp = \perp$ if $s \in T(F \times \{\lambda\}, A \times \{\lambda\})$,
2. $s_\perp = s$ if $s \in A \times (\mathbf{B}^k - \{\lambda\})$,

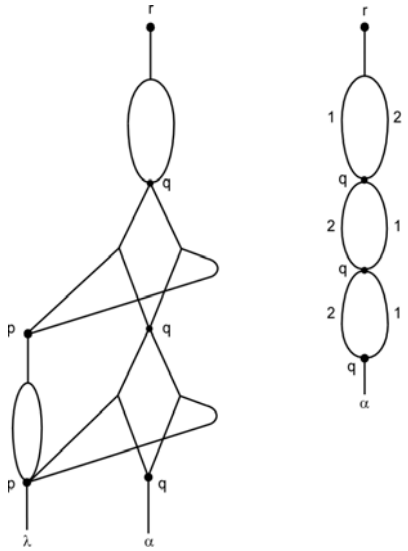

Figure 5: For the DAG H of Example 4.

3. $s_\perp = (f,\beta)(s_{1\perp},...,s_{m\perp})$ if $s = (f,\beta)(s_1,...,s_m)$ and $s \notin T(F\times\{\lambda\}, A\times\{\lambda\})$.

In the last case, we may have $\beta = \lambda$ if some of the terms $s_1,...,s_m$ is not a λ-*term*, i.e., is not in $T(F\times\{\lambda\}, A\times\{\lambda\})$.

For an example let $t = f(a, g(b,b'), h(c,c'), l(d,d'), e)$. Let $k = 1$, let α denote $(True)$ and λ denote $(False)$. Let U_1 consist of the occurrences of g, b, c and let $s = t[U_1]$. Then :

$s_\perp = f_\lambda(\perp, g_\alpha(b_\alpha,\perp), h_\lambda(c_\alpha,\perp),\perp,\perp)$, where we write f_λ for (f,λ), g_α for (g,α), *etc...* for the purpose of readability.

Our objective is to enumerate the set $SAT(S_{Unf(H)}, \varphi(X_1,...,X_k))$ defined as the set of terms $t[U_1,...,U_k]_\perp$ for $t[U_1,...,U_k]$ in $L_{\varphi(X_1,...,X_k)}[t]$. We will apply the technique of Theorem 3. The mapping γ: $T(F,A) \longrightarrow T(F_*, F\cup A)$ extends into γ: $D(F,A) \longrightarrow D(F_*, F\cup A)$ in the obvious way (see Example 5) so that $Unf(\gamma(H)) = \gamma(Unf(H))$. We will use the MS transduction $\delta : T(F_*, F\cup A) \longrightarrow T(F,A)$ which inverses γ on terms. From an MS formula $\varphi(X_1,...,X_k)$ we construct the MS formula $\psi(X_1,...,X_k)$ that translates it "backwards" with respect to δ so that $Sat(S_t, \varphi(X_1,...,X_k)) = Sat(S_{\gamma(t)}, \psi(X_1,...,X_k))$ for every t in $T(F,A)$.

The algorithm works as follows. Given H, we compute $\gamma(H)$ and, by the first part of the proof, we can enumerate with linear delay the set $SAT(S_{Unf(\gamma(H))}, \psi(X_1,...,X_k))$. We let $t = Unf(H)$. From each term $\gamma(t)[U_1,...,U_k]_\perp$ of this set, we can compute $t[U_1,...,U_k]_\perp$ in linear time by means of the mapping

ξ: $T(F_*, (F\cup A)\times(\mathbf{B}^k - \{\lambda\})\cup\{\perp\}) \longrightarrow T(F\times\mathbf{B}^k, A\times(\mathbf{B}^k - \{\lambda\})\cup\{\perp\})$

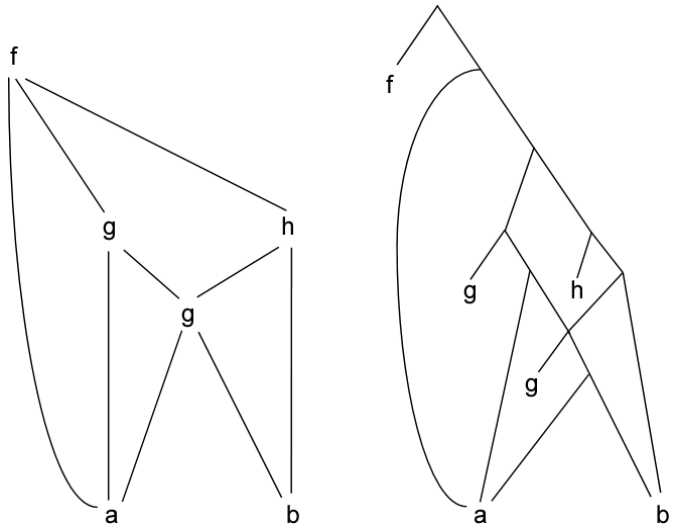

Figure 6: A DAG $H \in D(F, A)$ and the DAG $\gamma(H)$.

defined as follows.

1. If $s \in A \times (\mathbf{B}^k - \{\lambda\}) \cup \{\perp\}$, we let $\xi(s) = s$.

2. If $s = *_f(t_0, *(t_1, *(t_2, ..., *(t_{p-1}, t_p))..))$ where $p = \rho(f)$, then:

2.1. $\xi(s) = (f, \lambda)(\xi(t_1), ..., \xi(t_p))$ if $t_0 = \perp$,

2.2. $\xi(s) = (f, \alpha)(\xi(t_1), ..., \xi(t_p))$ if $t_0 = (f, \alpha)$.

3. If $s = *_f(t_0, *(t_1, *(t_2, ..., *(t_{p-1}, \perp))..))$ with $p < \rho(f)$, then

3.1. $\xi(s) = (f, \lambda)(\xi(t_1), ..., \xi(t_{p-1}), \perp, ..., \perp)$ if $t_0 = \perp$,

3.2. $\xi(s) = (f, \alpha)(\xi(t_1), ..., \xi(t_{p-1}), \perp, ..., \perp)$ if $t_0 = (f, \alpha)$, and, in both cases we have $\rho(f) - p + 1$ times $\perp$.

4. If none of these cases holds, or if is some necessary $\xi(t_1), ..., \xi(t_p)$ is undefined then $\xi(s)$ is undefined.

It is clear (using an induction on t) that if $s = \gamma(t)[U_1, ..., U_k]_\perp$ then $\xi(s)$ is well-defined and equal to $t[U_1, ..., U_k]_\perp$.

We continue with the example of $t = f(a, g(b, b'), h(c, c'), l(d, d'), e)$ and the set U_1 considered above. We have :

$s = \gamma(t)[U_1]_\perp = *_f(\perp, *(\perp, *(*_g(g_\alpha, *(b_\alpha, \perp)), *(*_h(\perp, *(c_\alpha, \perp)), \perp)),$

In this case $\rho(f) = 5, p = 4$. Then $\xi(s) = t[U_1]_\perp = f_\lambda(\perp, g_\alpha(b_\alpha, \perp), h_\lambda(c_\alpha, \perp), \perp, \perp)$.

We note also that the sizes of s and $\xi(s)$ are linearly related. From a linear delay enumeration algorithm for $SAT(S_{Unf(\gamma(H))}, \psi(X_1, ..., X_k))$, we get one for $SAT(S_{Unf(H)}, \ \varphi(X_1, ..., X_k))$.□

Example 5 : The left part of Figure 6 shows a DAG $H \in D(F, A)$ and its right part the corresponding DAG $\gamma(H)$ (where the labels $*$ and $*_f$ are omitted for readability). We have :

$Unf(H) = f(a, g(a, g(a, b)), h(g(a, b), b))$,
$Unf(\gamma(H))$
$= *_f(f, *(a, *(*_g[g, *(a, *_g(g, *(a, b)))], *_h[h, *(*_g(g, *(a, b)), b)])))$
$= \gamma(Unf(H)).\Box$

Remarks 7 : (1) The efficiency of this algorithm can be measured as follows: for a preprocessing time p, we may obtain $2^{2^{a \cdot p}}$ results : it suffices to take the DAGs with maximal sharing that unfolds into the terms t_n defined by $t_1 = b$, $t_{n+1} = f(t_n, t_n)$.

(2) Condition (c2) of Definition 2 is no longer valid for the DAG $G_2(H)$. The space requirement is thus important because one needs to store the function $Select^{OR}$ (see Remark 2(2) in Section 2). The space needed for the linear delay enumeration algorithm is thus $O(N \cdot Card(V_H))$, where N is the number of results. For the direct generation algorithm, we need space $O(Card(E_H))$.

5 Applications to graphs, hypergraphs and relational structures.

We apply the results of the previous section to MS queries in graphs or relational structures that are images of trees under MS transductions. We will reduce the enumeration problems to those considered in Section 4. We refer the reader to [4, 5] or to the appendix for the definition of clique-width and some lemmas.

A set $\mathcal{C}$ of relational structures is *tree-like* if it is the image of a subset of $T(F, A)$ under a noncopying MS-transduction where F is binary. Being tree-like is meaningful only for an infinite set of structures. Every finite set of structures is tree-like in a trivial way. The set of complete graphs is tree-like in this sense. A set of directed or undirected graphs is tree-like if and only if it has bounded clique-width.

Theorem 6 : Let $\mathcal{C}$ be a set of tree-like structures over a finite signature σ with defining MS transduction $\tau : T(F, A) \longrightarrow \mathcal{C}$. Let $\varphi(X_1, ..., X_k)$ be a monadic second-order formula over σ.

(1) For every term $t \in T(F, A)$ defining a structure $S = \tau(S_t)$ there exists an algorithm that enumerates the set $Sat(S, \varphi(X_1, ..., X_k))$ with linear delay after a preprocessing taking time $O(\mid t \mid . \log(\mid t \mid))$.

(2) For each i, the i-th element B_i of this set can be computed in time $O(\mid B_i \mid \cdot \log(\mid t \mid))$. The preprocessing takes time $O(\mid t \mid \cdot \log(\mid t \mid))$.

Proof : We apply the "backwards translation lemma" with respect to the transduction τ : from $\varphi(X_1, ..., X_k)$ one can construct an MS formula $\psi(X_1, ..., X_k)$ such that for every $t \in T(F, A)$ we have, letting $S = \tau(S_t)$:

$$Sat(S, \varphi(X_1, ..., X_k)) = Sat(S_t, \psi(X_1, ..., X_k)).$$

Hence, we are reduced to an enumeration problem and a direct generation problem for the results of MS queries on terms of $T(F, A)$. We finish the proof by using Theorem 4.□

Applications to graphs and hypergraphs

Corollary 2 : (1) Let $\mathcal{C}$ be a set of simple, directed or undirected graphs of clique-width at most p. For every monadic second-order formula $\varphi(X_1, ..., X_k)$ there exists an algorithm that takes as input a clique-width expression t of width p defining a graph G in $\mathcal{C}$ with n vertices and enumerates the set $Sat(G, \varphi(X_1, ..., X_k))$ with linear delay, after a preprocessing using time $O(n \cdot \log(n))$.

(2) If a graph G of clique-width at most p with n vertices is not given by a clique-width expression, the same can be done with a preprocessing taking time $O(n^3)$.

(3) In each of these cases and for each i, the i-th element B_i of $Sat(G, \varphi(X_1, ..., X_k))$ can be computed in time $O(\mid B_i \mid \cdot \log(n))$, where $\mid B_i \mid$ is its size. The preprocessing takes the same time as in (1) and (2) respectively.

Proof : (1) This is an immediate consequence of Theorem 6, because, by Lemma 3 of [5], every clique-width expression of width p defining a graph with n vertices can be transformed into one of same width that defines the same graph and has size $O(n)$, and the mapping that associates with a clique-width expression of width p the graph it defines is a noncopying MS transduction.

(2) With help of the algorithms by Hliněny and Oum [12,15] that take time $O(n^3)$ to construct an $f(p)$-expression for graphs of clique-width at most p, where f is a fixed function.

(3) By (2) of Theorem 6. □

For graphs and hypergraphs of bounded tree-width, one can use MS formulas over incidence graphs which allow quantifications over sets of edges and hyperedges. We call them MS_2 formulas. See Courcelle [4] for details. In this case, the size of the relational structure $Inc(G)$ representing a graph or hypergraph G is the number of vertices plus the number of edges and hyperedges (see the appendix for $Inc(G)$). This size is denoted by $\parallel G \parallel$. The height $ht(D)$ of a tree-decomposition D is defined as the diameter of the underlying tree which is usually undirected. The tree-decompositions of a hypergraph are those of the graph obtained by replacing a hyperedge by edges between any two vertices of this hyperedge. The notion of tree-width follows immediately.

Corollary 3 : (1) Let $\mathcal{D}$ be a set of graphs or hypergraphs of tree-width at most p. For every MS_2 formula $\varphi(X_1, ..., X_k)$ there exists an algorithm that takes as input a tree-decomposition D of width at most p defining a graph or hypergraph G in $\mathcal{D}$ and enumerates the set $Sat(Inc(G), \varphi(X_1, ..., X_k))$ with linear delay. The preprocessing takes time $O(\parallel G \parallel \cdot \log(\parallel G \parallel))$.

(2) The same holds if the graph or hypergraph G is given without its tree-decomposition, after a preprocessing taking the same time.

(3) In each of these cases and for each i, the i-th element B_i of $Sat(Inc(G), \varphi(X_1, ..., X_k))$ can be computed in time $O(\mid B_i \mid \cdot \log(\parallel G \parallel))$, where $\mid B_i \mid$ is its size. The preprocessing takes the same time as in (1) and in (2).

Proof : (1) This is a consequence of Theorem 6 : a tree-decomposition D of width $\leq p$ can be converted into a term over an appropriate alphabet and this term has height $O(ht(D))$. The mapping from such a term for a decomposition of width $\leq p$ to the corresponding graph or hypergraph G, represented by $Inc(G)$ is a noncopying MS transduction. We also use the fact that a tree-decomposition D of width $\leq p$ can be transformed into one of bounded width with $ht(D) = O(\log(\parallel G \parallel))$, via the constructions of Theorem 4.

(2) With help of the algorithm by Bodlaender (see [7]) that constructs in time $O(n)$ a tree-decomposition of width p if there exists one for a graph or hypergraph with n vertices.

(3) By (2) of Theorem 6.□

Remarks 8 : In these algorithms, the main part consists in enumerating or generating answers to queries on terms, and, from them the answers to the considered queries in relational structures, graphs or hypergraphs. The space requirement is as for Theorem 4, which gives $O(\parallel G \parallel \cdot \log(\parallel G \parallel))$.

6 Enumeration of sets of words and terms

Our purpose is here to enumerate sets of words or terms, not answers to queries in one given word or term. However, the techniques will be similar to the ones used in Sections 3 and 4, and will use the algorithm of Theorem 1 that enumerates the set of trees embedded in an AND/OR-DAG.

We will consider languages $L \subseteq A^+$ specified by finite-state automata or context-free grammars. We will consider the problems of enumerating the set $L[n]$ of words in L of length at most n and the set $L^{(n)}$ of words generated by a derivation tree of height at most n in case L is defined by a context-free grammar.

Proposition 3 : (1) Let $L \subseteq A^+$ be a regular language. For every n, the finite language $L[n]$ can be enumerated in lexicographic order with linear delay and preprocessing time and space $O(n)$.

(2) Let $L \subseteq A^+$ be defined by a linear nonambigous context-free grammar. For every n, the finite language $L^{(n)}$ can be enumerated with linear delay and preprocessing time and space $O(n)$.

(3) In each case, the i-th element can be obtained in linear time in its size, with preprocessing time and space $O(n)$.

Proof : (1) Letting \$ be not in A, we can assume that $L\$$ is given by a deterministic finite automaton $\mathcal{A}$ with set of states Q. We construct a DAG

$G_{\mathcal{A}}[n]$ as in the proof of Theorem 2. Its vertices are pairs (q,i) of a state q of $\mathcal{A}$ and some $i \in \{0,...,n\}$ together with a unique accepting state q_{Acc} receiving the transitions on letter \$ that terminate an accepted word. A linear order on $A \cup \$$ is fixed such that \$ is the minimal symbol and a linear order on the edges of $G_{\mathcal{A}}[n]$ follows. The associated lexicographic order on L is the same as on $L\$$. The enumeration of the paths in $G_{\mathcal{A}}[n]$ from $(p,0)$ (p initial) to q_{Acc} i.e., of the words of $L[n]$, can be done with linear delay by Theorem 1. Since there are no ε-transitions to remove, the preprocessing time is $O(n)$. It consists in removing the vertices which are not on a path to q_{Acc}.

(2) A context-free grammar, linear or not, that is unambigous is necessarily *loop-free*. This means that it has no derivation sequence of the form $S \longrightarrow^+ S$ where S is a nonterminal occurring in a derivation of a word of the generated language (otherwise, some word derivable using S would have several derivation trees). It follows that the size of a derivation tree t and the length of the generated word w are related by $\mid w \mid \leq m \cdot \mid t \mid$ and $\mid t \mid \leq (2M+1) \cdot \mid w \mid$ where m is the maximal number of terminal symbols in the right handside of a rule and M is the maximal number of steps of a derivation sequence of the form $S \longrightarrow^+ U$ where S, U are nonterminal symbols.

If such a grammar is linear (at most one nonterminal in each right handside) its derivation sequences (derivation trees with a single path) are described by a regular language. We can use (1) to enumerate derivation sequences d with linear delay. We get from each sequence d the derived word w in time $O(\mid d \mid)$ or equivalently, in time $O(\mid w \mid)$. The words are thus enumerated with linear delay in the lexicographic order on their derivation sequences.

(3) This follows from Assertion (2) of Theorem 1 and Remark 2 (2), because the constructed DAGs have bounded degree (in terms of the size of A). □

Next we consider the case of a recognizable set $L \subseteq T(F,A)$. We denote by $L\{h\}$ the set of terms in L of height at most h. We sketch a proof using only the results of Section 2.

Theorem 7 : Let (F,A) be a signature. If $L \subseteq T(F,A)$ is recognizable, then for each n, the set $L[n]$ and the set $L\{n\}$ can be enumerated with linear delay with a preprocessing taking time $O(n^2)$. The computation of the i-th element t_i according to a fixed linear order can be done with the same preprocessing time, within time $O(\mid t_i \mid \cdot \log(n))$.

Proof: We first consider the case of $L[n]$ and we first assume that all symbols of F have arity 2. We order linearly the set $F \cup A$. We let L be defined by a (complete deterministic) automaton $\mathcal{A}$ with set of states Q and set of accepting states Q_{Acc}. We let $L(q,m)$ be the set of terms t with size equal to m such that $run_{\mathcal{A}}(Root(t)) = q$. For a fixed integer n we construct an AND/OR-graph $G_{\mathcal{A}}[n]$ as follows.

Its set of OR-vertices is $Q \times [n]$, where $[n] = \{1,...,n\}$. Its set of AND-vertices is $(A \times \{1\}) \cup (F \times Q \times Q \times [n] \times [n])$. We define its edges as follows:

$(q,1) \longrightarrow (a,1)$ if (a,q) is a transition of $\mathcal{A}$,

$(q,i) \longrightarrow (f,q_1,q_2,i_1,i_2)$ if $i>1$, (q_1,q_2,f,q) is a transition of $\mathcal{A}$ and $i_1+i_2 = i-1$,
$(f,q_1,q_2,i_1,i_2) \longrightarrow (q_1,i_1)$,
$(f,q_1,q_2,i_1,i_2) \longrightarrow (q_2,i_2)$.

It has $O(n^2)$ vertices and edges, and :

$$deg^{OR}(G) \leq Max\{Card(A), n \cdot Card(F) \cdot Card(Q)^2\} = O(n).$$

We order the edges $(q,1) \longrightarrow (a,1)$ outgoing from $(q,1)$ by using the linear order on A. We also order linearly the set Q. We order the edges $(q,i) \longrightarrow (f,q_1,q_2,i_1,i_2)$ outgoing from (q,i) by lexicographic order on the sequences (f,q_1,q_2,i_1,i_2). Hence $G_{\mathcal{A}}[n]$ is a locally ordered AND/OR-DAG. It is essentially a tree automaton. With (q,m) as accepting state, it recognizes the set $L(q,m)$.

Claim : For every $m \leq n$ and $T \in EMB(G_{\mathcal{A}}[n],(q,m))$, the tree obtained from T by contracting the edges outgoing from OR-vertices (all of outdegree 1) is an element of $L(q,m)$. Conversely every term in $L(q,m)$ is obtained in this way from a unique tree T in $EMB(G_{\mathcal{A}}[n],(q,m))$. The size of T is exactly twice that of t.

It follows that the enumeration of $L[n]$ and the computation of its i-th element with respect to a linear order that we will make precise below reduce to the corresponding problems for $\bigcup\{EMB(G_{\mathcal{A}}[n],(q,m)) \mid m \leq n, q \in Q_{Acc}\}$. We order linearly the set $\{(q,m) \mid m \leq n, q \in Q_{Acc}\}$ by right to left lexicographic order, hence we order the output trees by increasing sizes. We can apply Theorem 1. We obtain a linear delay enumeration algorithm with preprocessing time $O(n^2)$, and a direct generation algorithm with time complexity $O(s \cdot \log(n))$ where s is the size of the result.

We now make precise the order in which terms are enumerated. For t in $T(F,A)$ we define $R(t)$ as the sequence :

$$(\mid t \mid, run_{\mathcal{A}}(Root(t)), First(t), run_{\mathcal{A}}(Root(t_1)), run_{\mathcal{A}}(Root(t_2)), \mid t_1 \mid, t_1, t_2)$$

where t_1 and t_2 are such that $t = First(t)(t_1,t_2)$ if $First(t) \in F$ and $t_1 = t_2 = First(t)$ if $First(t) \in A$. We do not use $\mid t_2 \mid$ in this list because this value is determined from $\mid t \mid$ and $\mid t_1 \mid$ by $\mid t_2 \mid = \mid t \mid - \mid t_1 \mid -1$ if $First(t) \in F$ and is 1 otherwise.

Then, we let $s \prec t$ if and only if $R(s) <_{lex} R(t)$, where $<_{lex}$ is based on the orderings on the set $\mathbf{N}$ of nonnegative integers, on the sets $F \cup A$ and Q, and on $\prec$ itself (for comparisons of the last two components). This is a (well-defined) recursive definition of a strict linear order on $T(F,A)$. The ordering of edges outgoing OR-vertices is such that the terms of $L[n]$ are enumerated according to this linear order.

If F has symbols of arity up to p, the coding γ of the proof of Theorem 3 can be used. Note that $\mid \gamma(t) \mid = \Theta(\mid t \mid)$; the set $\gamma(L)$ is a recognizable subset of $T(F_*, F \cup A)$ and by an easy modification of the above construction, one can enumerate with linear delay the set of terms t in $\gamma(L)$ such that $\mid \gamma^{-1}(t) \mid = n$. From them, one obtains those of $L[n]$, also with linear delay.

One can also enumerate or generate with the same complexity the sets of terms in L having size between m and n. Similar constructions can be done for the set $L\{n\}$ of terms of height at most n of a recognizable subset L of $T(F, A)$. □

Remarks 9 : *Space requirements.* Here Condition (c2) of Definition 2 does not hold. Hence the space needed is proportional to the number of results, (but is not the total size of the set of results). For the direct generation algorithm, the space requirement is $O(n^2)$ (cf. Remark 2 (2)). Its data structure can also be used for an enumeration algorithm with delay $O(s \cdot \log(n))$ where s is the size of the generated result.

Question : Can one generate $L[n]$ according to the natural lexicographic order. This order does not depend on any automaton ?

Other enumeration parameters

The *Strahler number* (also known as *register allocation number)* is defined as follows for a binary tree $t \in T(F, A)$:

$Str(a) = 1$, if $a \in A$,

$Str(f(t_1, t_2)) = Max\{Str(t_1), Str(t_2)\}$ if $Str(t_1) \neq Str(t_2)$,

$= 1 + Str(t_1)$ if $Str(t_1) = Str(t_2)$.

We denote by $L\{h, m\}$ the set of terms in L of height at most h and of Strahler number at most m.

Corollary 4 : If $L \subseteq T(F, A)$ is recognizable, then for all h, m with $m \leq h$, the set $L\{h, m\}$ can be enumerated with linear delay after a preprocessing taking time $O(m^2 \cdot h^2)$. The computation of the i-th element t_i according to a fixed linear order can be done with the same preprocessing time, within time $O(\mid t_i \mid \cdot \log(m \cdot h))$.

Proof : It uses the same technique as for size or height, with $L(q, n)$ replaced by $L(q, n, \ell)$ defined as the set of terms t with $ht(t) = n$, $Str(t) = \ell$ and such that $run_{\mathcal{A}}(Root(t)) = q$. The set of OR-vertices is $Q \times [m] \times [h]$ and the set of AND-vertices is $(A \times \{1\} \times \{1\}) \cup (F \times Q \times Q \times [m] \times [h] \times [m] \times [h])$. The constructions and proofs extend easily. □

We now show how the second assertion of Theorem 7 can be improved by using Theorem 5.

Theorem 8 : Let F be a finite set of function symbols. If $L \subseteq T(F,A)$ is recognizable, then for each h, the set $L\{h\}$ can be enumerated with linear delay with a preprocessing taking time $O(h)$. The computation of the i-th element t_i according to a certain linear order can be done with the same preprocessing time within time $O(\mid t_i \mid)$.

Proof : Let $h, k > 0$, let $\mathbf{T}(h,k)$ be the tree of words over $\{1, ..., k\}$ of length at most $h-1$, ordered by the prefix order. The sons of a vertex are compared lexicographically. Let F be a set of function symbols of arity at most k. A term t in $T(F,A)$ of height at most h can be described by a coloring of the nodes of $\mathbf{T}(h,k)$ by elements in $F \cup A$. (A constant has height 1 and is represented by the empty word.) A node of $\mathbf{T}(h,k)$ with no color is not an occurrence of t. Such a coloring can be defined as an assignment of sets of nodes to the variables of the set $W_{F,A} = \{X_b \mid b \in F \cup A\}$. (One can actually use only $\lceil \log(\mid F \cup A \mid) \rceil$ variables.) We denote by ν_t the assignment which encodes $t \in T(F,A)$. For every recognizable set $L \subseteq T(F,A)$ there exists (by the direction of the theorem of Doner, Thatcher and Wright [6, 16] opposite to the one we have used up to now), an MS formula φ_L with set of free variables $W_{F,A}$ such that for every positive integer h, for every $t \in T(F,A)$:

$$ht(t) \leq h \Longrightarrow (t \in L \Longleftrightarrow (\mathbf{T}(h,k), \nu_t) \models \varphi_L)$$

The MS formula φ_L is independent of h. The tree $\mathbf{T}(h,k)$ is the unfolding of a DAG with h vertices and $(h-1) \cdot k$ edges. For each h and by Theorem 5, we can enumerate with linear delay the set $SAT(\mathbf{T}(h,k), \varphi_L)$. The size of a term s in this set is proportional to that of the term t of L that it encodes (we have $\mid t \mid \leq \mid s \mid \leq k \cdot \mid t \mid$), and the term t can be constructed in linear time from s. (See Example 6). This gives a linear delay enumeration algorithm for $L\{h\}$ with preprocessing time $O(h)$. □

Example 6:

We let $k = 2, h = 5$, and $t = f(a, g(f(g(b)), f(c,d)))$ where $\rho(f) = 2, \rho(g) = 1$. The tree $\mathbf{T}(5,2)$ can be written linearly as :

$$\begin{array}{l}\varepsilon(1\{1[1(1,2), 2(1,2)], 2[1(1,2), 2(1,2)]\},\\ \qquad 2\{1[1(1,2), 2(1,2)], 2[1(1,2), 2(1,2)]\})\end{array}$$

Its annotation $\mathbf{T}(5,2)[\nu_t]$ which encodes t is then

$$\begin{array}{l}\varepsilon_f(1_a\{1_\lambda[1_\lambda(1_\lambda, 2_\lambda), 2_\lambda(1_\lambda, 2_\lambda)], 2_\lambda[1_\lambda(1_\lambda, 2_\lambda), 2_\lambda(1_\lambda, 2_\lambda)]\},\\ \qquad 2_g\{1_f[1_g(1_b, 2_\lambda), 2_f(1_c, 2_d)], 2_\lambda[1_\lambda(1_\lambda, 2_\lambda), 2_\lambda(1_\lambda, 2_\lambda)]\})\end{array}$$

where the subscripts f, a, g *etc...* indicate that the tuple of sets ν_t specifies $f, a, g, ...$ respectively for the corresponding occurrence.

The term $\mathbf{T}(5,2)[\nu_t]_\perp$ is :

$$\varepsilon_f(1_a\{\bot,\bot\}, 2_g(1_f[1_g(1_b,\bot), 2_f(1_c,2_d)],\bot)).$$

from which we get t by removing the occurrences of $\bot$ and replacing ε_f by f, 1_a by a *etc...*

Remarks 10 : (1) If F consists of one binary function symbol and A consists of two constants, then, $L\{h\}$ can contain more than 2^{2^h} terms. The efficiency of this algorithm is thus $2^{2^{a.p}}$.

(2) *Space requirement* : $O(N \cdot h)$ where N is the number of results. Or alternatively, $O(h)$ with an enumeration delay of $O(s.h)$ by using the generation algorithm.

Applications to context-free grammars.

Corollary 5 : Let $L \subseteq A^+$ be defined by a nonambigous context-free grammar. For every n, the finite language $L^{(n)}$ can be enumerated with linear delay and preprocessing time $O(n)$. So can be the finite language $L[n]$ with preprocessing time $O(n^2)$. The computation of the i-th element according to a fixed linear order can be done with the same preprocessing time within time $O(s.\log(n))$ where s is the size of the result.

Proof : We use the observations made in the proof of Proposition 3 (2). The case of $L^{(n)}$ is an immediate consequence of Theorem 8. For $L[n]$ we recall that a classical algorithm can transform the given grammar into one that is nonambigous and has only rules of the forms $S \longrightarrow a$ and $S \longrightarrow TU$ for nonterminal symbols S, T, U and terminal symbol a. For such grammars, a word w is generated by a derivation tree of size $3\cdot \mid w \mid -1$. The result follows then from Theorem 7. □

If the given grammar is ambigous, the methods apply but some words will be output several times. The algorithms of Corollary 5 actually enumerate without repetitions their derivation trees. This technique can also be applied to context-free *graph grammars* [4]. However, context-free graph grammars are usually ambigous. Hence the enumeration will present repetions.

7 Summary, conclusion and open questions

We first summarize our results in a table. The efficiency is discussed above after each theorem. We recall that p is the time spent for preprocessing, and s the size of a result. We denote by t_S a term the value of which, under a fixed MS transduction τ is a structure S (see Theorem 6). For Theorem 5, the size measure of the results is not the same as for Theorem 4.

Results	**Class of structures ; enumerated objects**	**Preprocessing time for linear delay enumeration ; efficiency**	**Time for direct generation**
Theorem 1	DAG G ; embedded trees	$O(Card(E_G))$	$O(s \cdot \log(\deg^{OR}(G)))$
Theorem 2 Corollary 1	word of length n ; MS query	$O(n^2)$; $2^{a \cdot p^{1/2}}$ $O(n \cdot \log(n))$; $2^{a \cdot p/\log(p)}$	$O(s \cdot \log(n))$
Prop. 1 Theorem 3	term $t \in T(F,A)$; MS query	$O(\mid t \mid \cdot ht(t))$ $2^{a \cdot p/\log(p)}$	$O(s \cdot \log(\mid t \mid))$
Theorem 4	term $t \in T(F,A)$; MS query	$O(\mid t \mid \cdot \log(\mid t \mid))$ $2^{a \cdot p/\log(p)}$	$O(s \cdot \log(\mid t \mid))$
Theorem 5	term $t = Unf(H)$, $H \in D(F,A)$; MS query on t	$O(Card(V_H))$ $2^{2^{a \cdot p}}$	$O(s)$
Theorem 6	Tree-like structure S ; MS query	$O(\mid t_S \mid \cdot \log(\mid t_S \mid))$	$O(s \cdot \log(\mid t_S \mid))$
Prop. 3	Regular or linear context-free language L ; $L[n]$ or $L^{(n)}$	$O(n)$	$O(s)$
Theorem 7 Theorem 8	Recognizable set of terms L ; $L[n]$ or $L\{n\}$	$O(n)$ or $O(n^2)$ $2^{2^{a \cdot p}}$	$O(s \cdot \log(n))$

G. Bagan gives in [1] another algorithm for enumerating with linear delay the set of answers to a monadic second-order query on classes of tree-like graphs or structures. He also uses DAGSs like $G_2(t)$ in Definition 4, but his preprocessing is different. A clever data structure makes possible to avoid the ε-reduction step, hence his preprocessing takes only linear time in the size of the structure, assumed to be given by its tree as in Theorem 3. However, his technique does not apply to terms representing by DAGs, as we can do in Theorem 5.

Here are some questions.

1. *Concerning size functions :* Let us define a *size function* is a total mapping $s : T(F,A) \longrightarrow \mathbf{N}$ such that $s(f(t_1,t_2,...,t_k)) \geq s(t_i)$ for all $f, t_1, t_2, ..., t_k$ and i. For which size functions do we have results like Theorem 7 and Corollary 4 ?

2. *Concerning applications to context-free languages :* Can one perform a preprocessing which avoids generating a word several times when the given context-free grammar is ambigous ? Ambigous grammars are considered in [2], but in a different way. Other references on random generations of words of context-free languages are [2,11,13], however these works do not consider the enumeration problem. Can one enumerate with linear delay finite parts of context-free languages L, say $L[n]$ or $L^{(n)}$ in lexicographic order ?

3. *Endless enumeration.* Can one enumerate with linear delay the words of an infinite recognizable or context-free set of words, terms or graphs by increasing lengths, and at which cost in terms of preprocessing time ? One would obtain a program that never stops.

4. *Applications to chemistry.* The enumeration of trees and graphs representing molecules is an important topic to which a lot of work has been devoted. This work is described in the book by Trinajstić et al. [18]. Furthermore the enumerated structures are of low tree-width (at most 2 in most cases). The methods of Section 6 are perhaps applicable to such problems.

Acknowledgement : This work has been inspired by reading the article [8] by A. Durand and E. Grandjean who give a *constant delay enumeration algorithm* for the answers to first-order queries on relational structures of bounded degree. I acknowledge useful discussions and exchanges with F. Olive, J. Niehren, G. Bagan, I. Durand, G. Sénizergues and M. Kanté.

8 References

[1] G. Bagan, MSO queries on tree decomposable structures are computable with linear delay, *Proceedings of Computer Science Logic 2006*, Lec. Notes Comput. Sci. **4207** (2006) 167-181.

[2] A. Bertoni, M. Goldwurm, M. Santini, Random generation for finitely ambigous context-free languages, *Theoretical Informatics and Applications* **35** (2001) 499-535.

[3] H. Comon *et al.*, *Tree Automata Techniques and Applications*, Book online, freely readable :
http://www.grappa.univ-lille3.fr/tata/

[4] B. Courcelle, The expression of graph properties and graph transformations in monadic second-order logic, Chapter 5 of the *Handbook of graph grammars and computing by graph transformations, Vol. 1 : Foundations*, G. Rozenberg ed., World Scientific (New-Jersey, London), 1997, pp. 313-400.

[5] B. Courcelle, R. Vanicat, Query efficient implementations of graphs of bounded clique-width, *Discrete Applied Mathematics* **131** (2003) 129-150

[6] J. Doner, Tree acceptors and some of their applications. *J. Comput. Syst. Sci.* **4** (1970) 406-451

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

[8] A.Durand, E. Grandjean, First-order queries on structures of bounded degree are computable with constant delay, *ACM Transactions on Computational Logic*, to appear.

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

[10] M. Goldwurm, Random generation of words in an algebraic language in linear binary space, *Inf. Proc. Letters* **54** (1995) 229-233.

[11] T. Hickey, J. Cohen, Uniform random generation of strings in a context-free language, *SIAM on Comput.* **12** (1983) 645-655.

[12] P. Hliněny, S. Oum, Finding branch-decompositions and rank-decompositions, Preprint, March 2007.

[13] H. Mairson, Generating words in a context-free language uniformly at random generation, *Inf. Proc. Letters* **49** (1994) 95-99.

[14] J. Niehren, L. Planque, J.-M. Talbot, S. Tison, N-ary queries by tree Automata, *10th Int. Symp. on Data Base Programming Languages*, Lec. Notes Comp. Sci. **3774** (2005) 217-231.

[15] S. Oum, Approximating rank-width and clique-width quickly, 2006, Expanded version of the communication in *Proceedings of WG 2005*, Lec. Notes Comp. Sci. **3787** (2005) 49-58.

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

[17] W. Thomas: Languages, automata and logic, in *Handbook of Formal Languages Volume 3*, G. Rozenberg, A. Salomaa eds., Springer 1997, pp. 389-455.

[18] N. Trinajstić, S. Nikolić, J. Knop, W. Müller, K. Szymanski, *Computational chemical graph theory : Characterization, enumeration and generation of chemical structures by computer methods*, Ellis Horwood, Chichester, England, 1991.

9 Appendix : Monadic second-order logic and clique-width

This appendix reviews Monadic second-order logic and transformations of structures expressed in this language, called *MS transductions*. The reader is referred to the book chapter [4] or to the related article [5].

9.1 Relational structures and monadic second-order logic

Let $R = \{A, B, C, ...\}$ be a finite set of *relation symbols* each of them given with a nonnegative integer $\rho(A)$ called its *arity*. We denote by $\mathcal{STR}(R)$ the set of *finite* R-structures $S = \langle D_S, (A_S)_{A\in R}\rangle$ where $A_S \subseteq D_S^{\rho(A)}$ if $A \in R$ is a relation symbol, and D_S is the *domain* of S. If R consist of relation symbols of arity one or two we say that the structures in $\mathcal{STR}(R)$ are *binary*.

A simple graph G can be defined as an $\{edg\}$-structure $G = \langle V_G, edg_G\rangle$ where V_G is the set of vertices of G and $edg_G \subseteq V_G \times V_G$ is a binary relation representing the edges. For undirected graphs, the relation edg_G is symmetric. If in addition we need vertex labels, we represent them by unary relations. Binary structures can be seen as vertex- and edge-labelled graphs. If we have several binary relations say A, B, C, the corresponding graphs have edges of types A, B, C.

The *incidence graph* of G, undirected is the relational structure $Inc(G) = \langle V_G \cup E_G, inc_G\rangle$ such that $inc_G(x, y)$ holds if and only if $x \in E_G, y \in V_G$ and y is an end of x. For G directed we use $Inc(G) = \langle V_G \cup E_G, inc_{1G}, inc_{2G}\rangle$ such that $inc_{1G}(x, y)$ (resp. $inc_{2G}(x, y)$) holds if and only if $x \in E_G, y \in V_G$ and y is the origin (resp. the target) of x.

We recall that *Monadic Second-order logic* (*MS* logic for short) is the extension of *First-Order logic* (*FO* logic for short) with variables denoting subsets of the domains of the considered structures and new atomic formulas of the form $x \in X$ expressing the membership of x in a set X. (Uppercase letters will denote set variables, lowercase letters will denote first-order variables). We denote by $FO(R, W)$ (resp. by $MS(R, W)$) the set of *First-order* (resp. *Monadic second-order*) formulas written with the set R of relation symbols and having their free variables in a set W consisting of *first-order as well as of set variables*. Hence, we allow first-order formulas with free set variables and written with atomic formulas of the form $x \in X$. In first-order formulas, only first-order variables can be quantified.

As a typical and useful example, we give an MS formula with free variables x and y expressing that (x, y) belongs to the reflexive and transitive closure of a binary relation A :

$$\forall X(x \in X \wedge \forall u, v[(u \in X \wedge A(u, v)) \Longrightarrow v \in X] \ \Longrightarrow y \in X).$$

If the relation A is not given in the structure but defined by an MS formula, then one replaces $A(u, v)$ by this formula with appropriate substitutions of variables.

9.2 Monadic Second-order transductions

We will also use MS formulas to define certain graph transformations. As in language theory, a binary relation $\mathcal{R} \subseteq \mathcal{A}\times\mathcal{B}$ where $\mathcal{A}$ and $\mathcal{B}$ are sets of relational

structures will be called a *transduction* : $\mathcal{A} \to \mathcal{B}$. An *MS transduction* is a transduction specified by MS formulas. It transforms a structure S, given with an n-tuple of subsets of its domain called the *parameters*, into a structure T, the domain of which is a subset of $D_S \times [k]$. ($[k]$ denotes$\{1, ..., k\}$). We will refer to the integer k by saying that the transduction is *k-copying*; if $k = 1$ it is *noncopying.* Furthermore, each such transduction, has an associated *backwards translation*, a mapping that transforms effectively every MS formula φ relative to T, possibly with free variables, into one, say $\varphi^{\#}$, relative to S and having free variables corresponding to those of φ (k times as many actually) and to those denoting the parameters. This new formula expresses in S the property of T defined by φ. We now give some details. In this article we need not use transductions with parameters. Hence we simplify the definitions accordingly. More can be found in [4].

We let R and Q be two finite sets of relation symbols. A (Q, R)-*definition scheme* is a tuple of formulas of the form :

$$\Delta = (\varphi, \psi_1, \cdots, \psi_k, (\theta_w)_{w \in Q*k})$$

where $k > 0, Q * k := \{(q, \vec{j}) \mid q \in Q, \vec{j} \in [k]^{\rho(q)}\}$, $\varphi \in MS(R, \varnothing), \psi_i \in MS(R, \{x_1\})$ for $i = 1, \cdots, k$ and $\theta_w \in MS(R, \{x_1, \cdots, x_{\rho(q)}\})$ for $w = (q, \vec{j}) \in Q * k$. These formulas are intended to define a structure T in $\mathcal{STR}(Q)$ from a structure S in $\mathcal{STR}(R)$.

Let $S \in \mathcal{STR}(R)$. A Q-structure T with domain $D_T \subseteq D_S \times [k]$ is *defined from* S by Δ if :

(i) $S \models \varphi$

(ii) $D_T = \{(d, i) \mid d \in D_S, i \in [k], (S, d) \models \psi_i\}$

(iii) for each q in Q : $q_T = \{((d_1, i_1), \cdots, (d_t, i_t)) \in D_T^t \mid (S, d_1, \cdots, d_t) \models \theta_{(q,\vec{j})}\}$, where $\vec{j} = (i_1, \cdots, i_t)$ and $t = \rho(q)$.

Since T is associated in a unique way with S and Δ whenever $S \models \varphi$, we can use the functional notation $def_\Delta(S)$ for T. The *transduction defined by* Δ is the binary relation $\mathcal{D}_\Delta = \{(S, T) \mid T = def_\Delta(S)\}$. Hence $\mathcal{D}_\Delta \subseteq \mathcal{STR}(R) \times \mathcal{STR}(Q)$. A transduction $f \subseteq \mathcal{STR}(R) \times \mathcal{STR}(Q)$ is an *MS transduction* if it is equal, up to isomorphism of structures, to $\mathcal{D}_\Delta$ for some (Q, R)-definition scheme Δ. A noncopying definition scheme can be written more simply : $\Delta = (\varphi, \psi, (\theta_q)_{q \in Q})$. If the formula ψ is the Boolean constant $True$ and $T = def_\Delta(S)$, then D_T contains $D_S \times \{1\}$, an isomorphic copy of D_S. This transduction defines the domain of T as an extension of that of S (up to isomorphism).

Example 7 :

We give some details of the definition scheme of the transduction δ defined in Section 4 as the inverse of the mapping $\gamma : T(F, A) \longrightarrow T(F_*, F \cup A)$.

We denote by p the maximum arity $\rho(f)$ of a symbol f in F. We let F_* be the set of binary function symbols consisting of $*$ and $*_f$ for each f in F. The mapping γ is defined recursively as follows :

$\gamma(a) = a$ if $a \in A$,

$\gamma(f(t_1, ..., t_m)) = *_f(f, *(\gamma(t_1), *(\gamma(t_2), ..., *(\gamma(t_{m-1}), \gamma(t_m)))...))$ if $m = \rho(f)$ and $t_1, t_2, ..., t_m \in T(F, A)$.

A term t in $T(F, A)$ is represented by the relational structure

$$S_t = \langle D_{S_t}, suc_{1t}, ..., suc_{pt}, (lab_{ct})_{c \in F \cup A} \rangle,$$

where $D_{S_t} = Occ(t)$, $suc_{it}(u, v)$ holds if and only if v is the i-th son of u, $lab_{ct}(u)$ holds if and only if u is an occurrence of c. The corresponding relational signature is $\tau_{F,A}$.

A term t in $T(F_*, F \cup A)$ is represented by the relational structure:

$$S_t = \langle D_{S_t}, suc_{1t}, suc_{2t}, (lab_{ct})_{c \in F_* \cup F \cup A} \rangle.$$

There exists a definition scheme $\Delta = (\varphi, \psi, (\theta_q)_{q \in \tau_{F,A}})$ that defines S_t from S whenever $S = S_{\gamma(t)}$ for some t in $T(F, A)$.

The formula φ is such that $S \models \varphi$ if and only if $S = S_w$ for some w in $T(F_*, F \cup A)$ and, furthermore $w = \gamma(t)$ for some t in $T(F, A)$; its construction is routine with the techniques presented in [4].

The formula $\psi(x_1)$, defined as $\bigvee_{c \in F \cup A} lab_c(x_1)$ specifies the domain of S_t as the set of elements of $D_{S_{\gamma(t)}}$ labelled by symbols in $F \cup A$ (as opposed to by symbols of F_*).

The formulas $\theta_{lab_c}(x_1)$, defined as $lab_c(x_1)$ for each c in $F \cup A$, express that the labels do not change in the transduction δ.

We will use an auxiliary formula $\eta(u, x_2)$ defined as :

$$(u = x_2 \wedge \bigvee_{a \in A} lab_a(x_2)) \vee (suc_1(u, x_2) \wedge \bigvee_{f \in F} lab_f(x_2))).$$

This formula expresses the fact that if $t = a \in A$, then $\gamma(t) = t$ and that, otherwise, the root of $\gamma(t)$ is the left son of the root of t. Then we define $\theta_{suc_i}(x_1, x_2)$ as the formula :

$$\psi(x_1) \wedge \psi(x_1) \wedge \bigvee_{f \in F} \theta_{f,i}(x_1, x_2)$$

where for each f, the formula $\theta_{f,i}$ is defined for $1 \leq i \leq \rho(f)$. We write these formulas for the case where $\rho(f) = 3$. The extension to the general case is straightforward. We obtain the following three formulas:

$\theta_{f,1}(x_1, x_2)$ is the formula :

$$\exists y_1, y_2, u.[lab_{*_f}(y_1) \wedge lab_*(y_2) \wedge suc_1(y_1, x_1) \wedge suc_2(y_1, y_2) \wedge suc_1(y_2, u) \wedge \eta(u, x_2)],$$

$\theta_{f,2}(x_1, x_2)$ is the formula :

$$\exists y_1, y_2, y_3, u.[lab_{*_f}(y_1) \wedge lab_*(y_2) \wedge lab_*(y_3) \wedge suc_1(y_1, x_1) \wedge suc_2(y_1, y_2) \wedge suc_2(y_2, y_3) \wedge suc_1(y_3, u) \wedge \eta(u, x_2)],$$

$\theta_{f,3}(x_1, x_2)$ is the formula:

$$\exists y_1, y_2, y_3, u.[lab_{*_f}(y_1) \wedge lab_*(y_2) \wedge lab_*(y_3) \wedge suc_1(y_1, x_1) \wedge suc_2(y_1, y_2) \wedge suc_2(y_2, y_3) \wedge suc_2(y_3, u) \wedge \eta(u, x_2)].\Box$$

The following lemma says that if $T = def_\Delta(S)$, then the monadic second-order properties of T can be expressed as monadic second-order properties of S. The usefulness of definable transductions is based on it.

Let $\Delta = (\varphi, \psi_1, \cdots, \psi_k, (\theta_w)_{w \in Q*k})$ be a (Q, R)-definition scheme. Let V be a set of set variables. For every variable X in V, for every $i = 1, \cdots, k$, we let X_i be a new variable. We let $V' = \{X_i \mid X \in V, i = 1, \cdots, k\}$. Let S be a structure in $\mathcal{STR}(R)$ with domain D. For every mapping $\eta : V' \longrightarrow \mathcal{P}(D)$, we let $\eta^k : V \longrightarrow \mathcal{P}(D \times [k])$ be defined by $\eta^k(X) = \eta(X_1) \times \{1\} \cup \cdots \cup \eta(X_k) \times \{k\}$.

Backwards Translation Lemma [4] : For every formula β in $MS(Q, V)$ one can construct a formula $\beta^\#$ in $MS(R, V')$ such that, for every S in $\mathcal{STR}(R)$, for every assignment $\eta : V' \longrightarrow S$ we have :

$(S, \eta) \models \beta^\#$ if and only if :
$def_\Delta(S)$ is defined, η^k is a V–assignment in $def_\Delta(S)$,
and $(def_\Delta(S), \eta^k) \models \beta$.

If the definition scheme and the formula β are FO, then the formula $\beta^\#$ is also first-order. Note that, even if $T = def_\Delta(S)$ is well-defined, the mapping η^k is not necessarily a V-assignment in T, because $\eta^k(X)$ may not be a subset of the domain of T which may be a proper subset of $D_S \times [k]$. We call $\beta^\#$ the *backwards translation* of β relative to the transduction def_Δ.

The composition of two transductions is defined as the composition of the corresponding binary relations. If they are both partial functions, then one obtains the composition of these functions.

Proposition 4 [4] : 1) The composition of two MS transductions (resp. noncopying MS transductions) is an MS transduction (resp. a noncopying MS transduction).
2) The inverse image of an MS-definable class of structures under an MS transduction is MS-definable.

In Section 5, we defined a class of tree-like structures as the image of a set of binary terms under a noncopying MS transductions. This is equivalent to taking the more liberal definition of the image of a set of trees under an MS transductions possibly using parameters.

9.3 Clique-width

Clique-width is like tree-width a graph complexity measure. It is defined and studied in [5] and in [4], among other works. Graphs are simple, directed or

not, and loop-free. Let C be a set of k labels. A *C-graph* is a graph G given with a total mapping lab_G from its vertex set to C. Hence G is defined as a triple $\langle V_G, edg_G, lab_G\rangle$. We call $lab_G(v)$ the *label* of a vertex v. The operations on C-graphs are the following ones :

(i) For each $i \in C$, we define a constant $\mathbf{i}$ to denote an isolated vertex labelled by i.

(ii) For $i, j \in C$ with $i \neq j$, we define a unary function $add_{i,j}$ such that :

$$add_{i,j}(\langle V_G, edg_G, lab_G\rangle) = \langle V_G, edg'_G, lab_G\rangle$$

where edg'_G is edg_G augmented with the set of pairs (u, v) such that $lab_G(u) = i$ and $lab_G(v) = j$. In order to add undirected edges, we take :

$$add_{i,j}(add_{j,i}(\langle V_G, edg_G, lab_G\rangle)).$$

(iii) We let also $ren_{i\to j}$ be the unary function such that

$$ren_{i\to j}(\langle V_G, edg_G, lab_G\rangle) = \langle V_G, edg_G, lab'_G\rangle$$

where $lab'_G(v) = j$ if $lab_G(v) = i$, and $lab'_G(v) = lab_G(v)$, otherwise. This mapping renames into j every vertex label i.

(iv) Finally, we use the binary operation $\oplus$ that makes the union of disjoint copies of its arguments. (Hence $G \oplus G \neq G$ and its number of vertices is twice that of G).

A well-formed term t over these symbols will be called a *k-expression* where $k = Card(C)$. Its *value* is a C-graph $G = val(t)$. The set of vertices of $val(t)$ is (or can be defined as) the set $Occ_C(t)$. However, we will also consider that a term t designates any graph isomorphic to $val(t)$. The *clique-width* of a graph G, denoted by *cwd(G)* is the minimal cardinality of C such that G is the value of some C-expression. A set of graphs has bounded clique-width if it has bounded tree-width, and the converse does not hold.

If we need to define graphs with vertex labels from a set L, then we use constant symbols $\mathbf{i}_a$ for i in C and a in L. The labels from L are not modified by any operation, and do not interfere with the other operations. To build a graph with labelled edges we can use the operation $add_{a,i,j}$ to add edges labelled by a from the vertices labelled by i to those labelled by j. The clique-width of a graph depends strongly on edge directions. Cliques and transitive tournaments have clique-width 2 but tournaments have unbounded clique-width.

Proposition 5 [4] : A set of graphs has bounded clique-width if and only if it is the image of a set of binary terms under a noncopying MS transduction.