THE COST OF LINEARZING GRAPH PROPERTIES

GRADINARIU M / TIXEUIL S

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud – LRI

09/2006

Rapport de Recherche N° 1458

The cost of linearizing graph properties

Selma Djelloul and David Soguet

LRI, UMR 8623, Bât 490 Université de Paris-Sud
91405 Orsay Cedex, France
{selma.djelloul,david.soguet}@lri.fr
http://www.lri.fr

Abstract. We consider the deterministic finite tree-automata corresponding to the atomic predicates of the monadic second-order logic for graphs of treewidth at most k. For each of these automata, we give the first non trivial upper bound on its minimum number of states. Our bounds rely on a well specified set of operations that transform a tree-decomposition of width at most k into a proper tree (i.e., a binary tree with no unary branching). Our bounds also rely on a precise labeling system that assigns labels of length $O(k^2)$ to the nodes of the proper tree such that all the information about the tree-decomposition distribute well over the proper tree. Every automaton is described by using either the standard representation by transition table, or by a recursive procedure that captures the behavior of the automaton. In the latter case, the procedure simply writes a record value at each node, during a bottom-up traversal of the tree. Each of these record values corresponds to a unique state. In our approach, there is no need of translations from graph vocabulary to tree vocabulary, as opposed to all the other approaches we are aware of in the same context.

1 Introduction

The notion of treewidth was defined by Robertson and Seymour [RS86] in the context of their investigations in the *graph minor theory*. Many of the *graph minors series* results are discussed in [Joh87] in the context of algorithmic computational complexity. Roughly speaking, treewidth measures how close a graph is to a tree. Courcelle [Cou92] proved that the treewidth of a graph is related to another parameter measuring the *width of the expression of a term in an algebra*. Earlier results [Don65,TW68] established an equivalence between the formulas of the *monadic second-order logic* (MSOL) and finite automata on the terms of free algebras. All these contributions together gave raise to new approaches in algorithmic complexity of graphs. One of these approaches relies on the decidability in linear time of MSOL on graphs of treewidth at most k. MSOL is the extension of first-order logic by second-order variables (denoted here by uppercase letters) that range over subsets of the domain. Corresponding atomic formulas of the form $X(x)$ are introduced for specifying that the element x of the domain belongs to the subset X. MSOL is powerful enough to express various NP-complete properties ([Cou97a,Cou97b]).

The idea behind the decidability in linear time of the formulas of MSOL on graphs of treewidth at most k is the following. For fixed k, all graphs of width at most k can be transformed into $O(n)$-node trees inside which the original graphs can be retrieved. The nodes of such a tree are encoded using a quantity of information that depends on k only. Hence, these trees can be run by finite tree-automata in time $O(n)$.

However, linear-time decidability of MSOL on graphs of treewidth at most k, remains hard to deal with for concrete algorithmic implementations. In particular, no practical bounds are known for the time required to achieve the construction of the automaton, not even on the number of states of the automata corresponding to the atomic predicates. The objective of this paper is to derive such bounds.

The literature suggests two main directions for deriving an algorithm from an MSOL-formula interpreted over graphs of bounded treewidth. One approach is inspired by the work of Feferman-Vaught [FV59] and Shelah [She75]. It is based on the recursive computation of the *boolean reduction sequence* [Cou90]. The second approach interprets a class of bounded treewidth graphs inside the class of labeled binary trees [ALS91]. Our approach is inspired from this latter method.

When dealing with automata in a practical way, the two significant factors are the size of the alphabet, and the size of the state set. In our work, we use a number of predicates that is at most $2(k+1)(k+2)+O(1)$ to interpret a graph of treewidth at most k into a binary tree. Both the transformation of the tree-decomposition into a binary tree, and the labeling system, are specified using a set of well defined operations. We are interested in measuring the impact that our specification has on the size of the automata corresponding to the atomic predicates. It is obvious that reducing the size of the state set of the atomic automata contributes in avoiding the blow-up of the state set of the final automaton. Since, we are able to "write" the automata corresponding to the atomic predicates via a direct interpretation in the tree, without any rewriting of formulas from one vocabulary to another, our work is the first that allows an interpretation of a graph class into trees without the use of the heavy and rigid formalism of *translation schemes*. The reader interested in how such a scheme is defined is referred to [Cou97a,Mak04].

1.1 Our framework

All considered graphs are undirected, have no isolated vertices, but can have loops and multiple edges. Limiting ourselves to undirected graphs is only for the sake of simplicity of the presentation.

We are interested in solving decision problems on input graphs of bounded treewidth. The goal of this paper is to derive bounds on the space resource needed to achieve the construction of the automaton corresponding to the formula expressing a graph property in the monadic second-order logic. Input graphs are assumed to be given together with their tree-decomposition. Note that determining whether a graph has treewidth at most k is NP-complete if k is part of the input, but decidable in linear time for fixed k ([Bod96]). In case of a positive answer, a tree-decomposition is produced by the algorithm in [Bod96]. However, this linear time is not practical because of large constants depending on k. Since the tree-decompositions we need do not have to be optimal, one can use polynomial-time approximation algorithms for treewidth. The best known approximation factor is $O(\sqrt{\log \mathsf{opt}})$ [FHL05].

1.2 Our results

There are two main contributions in our work. The first contribution is a technique for preprocessing the tree-decomposition in order to obtain a labeled binary tree that can be processed by a tree-automaton. This is done by applying a set of well defined operations that transform the tree-decomposition of the graph into a binary tree over which all the information on the tree-decomposition distribute in a well defined layout.

Our second contribution is a description of the automata associated to the atomic predicates. The construction of the final automaton is obtained by the classical inductive scheme where logical connectives are associated to automata-theoretic operations. In our approach, the automatic inductive scheme constructing the final automaton "reads" the graph-formula itself instead of reading a tree-formula obtained by translation from one vocabulary to another. Our description of the automata is obtained via a direct interpretation in the corresponding tree and makes useless the definition of intermediary sub-formulae, as opposed to, e.g., [ALS91]. Moreover, we reduce the construction in [ALS91] into a unique phase of interpretation. We obtain the following result on the number of states, where *DFTA* stands for *deterministic finite tree-automaton*:

Theorem 1.

(i) *There exists a DFTA with at most* $4k + 7$ *states that decides the equality relation on any* k*-bounded treewidth graph.*
(ii) *There exists a DFTA with at most* $2k^2 + 7k + 8$ *states that decides the incidence relation on any* k*-bounded treewidth graph.*

The adjacency relation is not part of our vocabulary. The formula $adj(x, y)$ is expressed by "$V(x) \wedge V(y) \wedge \{\exists e\ (E(e) \wedge inc(x, e) \wedge inc(y, e))\}$", where inc, the incidence relation is an atomic predicate in our vocabulary. The automatic inductive process of constructing automata computes the set of states of the product of two automata as the cartesian product of the two state sets. The (deterministic) automaton corresponding to a formula of the form $\exists X \varphi$ is obtained by projecting (with respect to X) the automaton corresponding to φ. If the set of states of the latter is S, the set of states of the former may be 2^S, the power set of S (see [ALS91] for more details). Hence, the theoretic process of constructing automata yields a number of states for the automaton corresponding to $adj(x, y)$ that can be $\Omega(2^{4k^4})$, which is unacceptable if we hope to go further in the construction of automata corresponding to more complex graph properties. We prove that the minimum number of states of an automaton deciding the adjacency relation on any graph of treewidth at most k, is actually $O(k^2)$. This result is obtained by a direct interpretation of the adjacency relation in the corresponding tree. We believe that many (simple) properties can be interpreted by analyzing what is the exact set of conditions in the tree that is equivalent to the considered graph property.

Theorem 2. *There exists a DFTA with at most $2k^2+9k+10$ states that decides the adjacency relation on any k-bounded treewidth graph.*

2 Definitions and notation

Tree automaton on proper trees. **-** *A proper tree* is a binary tree with no unary branching.

A (DFTA) on a proper tree is a quintuple $M = (S, \Sigma, \delta, s_0, A)$ where:

- S is a finite set of states,
- Σ is a finite set of letters, the alphabet, disjoint from S,
- δ is a transition function $\delta : \ S \times S \times \Sigma \to S$,
- s_0 is the initial state, $s_0 \in S$,
- A is the set of accepting states, $A \subset S$.

Let $M = (S, \Sigma, \delta, s_0, A)$ be a tree-automaton. Let T be a proper tree whose nodes are labeled with elements from Σ. M *executes* T by assigning states to its nodes during a bottom-up traversal. More precisely, a leaf with the label a, is assigned the state $\delta(s_0, s_0, a)$. For every node v, if v has label a, and if s_l and s_r are the states assigned to the children of v, then v is assigned the state $\delta(s_l, s_r, a)$. The automaton M *accepts* T iff the state assigned to its root is in A. A tree-automaton executes a tree in time that is linear in the size of the tree.

Graphs as relational structures. **-** A graph G is represented by the relational structure $\langle D^G, V^G, E^G, P_1^G, \ldots, P_p^G, Inc^G\rangle$ where:

- D^G is the domain of the structure, namely the set of both vertices and edges,
- V and E are unary predicates distinguishing vertices and edges respectively. That is, $V(x)$ holds iff x is a vertex, and $E(x)$ holds iff x is an edge,
- $P_j^G(x)$, $1 \le j \le p$, are unary predicates that give the possibility to define subsets of the domain of distinguished vertices or edges.
- Inc^G is a binary predicate defined by: $Inc^G(x, y)$ holds iff x is a vertex incident with the edge y.

A sequence of predicates defining relational structures for a language $\mathcal{L}$ is called a *vocabulary* for $\mathcal{L}$. If σ is a vocabulary for $\mathcal{L}$, we denote by $\mathcal{L}(\sigma)$ the set of MSOL-formulas expressed over σ. In the sequel, we denote by $\underline{G}$ the relational structure associated to the graph G.

The MSOL language on the relational structures defined above for graphs is strictly more expressive than the one on graph relational structures with only the vertices in the domain. For instance, the property of a graph of being hamiltonian is expressible in the former but not in the latter language.

The treewidth of a graph. - A tree-decomposition of a graph G is a pair $(T, \mathcal{F})$, where T is a tree and $\mathcal{F}$ is a family of subsets of D (the domain of $\underline{G}$) indexed by the nodes of T satisfying:

- $\bigcup_{X_t \in \mathcal{F}} X_t = D$
- For any y such that $E(y)$ holds, there exists a unique $X_t \in \mathcal{F}$ such that $y \in X_t$; and if x satisfies $Inc(x, y)$, then $x \in X_t$.
- For all $x \in D$ the subgraph of T induced by $\{t | x \in X_t\}$ is connected.

The width of a tree-decomposition is $\max\limits_{X_t \in \mathcal{F}} |\{x | x \in X_t, V(x)\ holds\}| - 1$

The graph G has *treewidth* w if w is the smallest integer such that G has a tree-decomposition of width w.
An example is depicted in figure 1 in the appendix.

3 From a tree-decomposition to a proper tree

We derive an algorithm that, given a fixed integer k and a tree-decomposition of a graph G of width at most k, produces a labeled proper tree whose labels encode all the information about the tree-decomposition. The starting point of our work is the paper by Arnborg at *al.* [ALS91]. As far as concrete implementation is concerned, there is a lack, in their work, of the specification of well defined operations to generate the proper trees. We define such a set of operations. This enables us to generate trees with a shape that is more constrained.

All considered tree-decompositions have no bag with a single vertex. Indeed, if such a bag exists then the vertex it contains, appears in at least one neighbor bag (since there are no isolated vertices in the graph). Then, we tansform the tree-decomposition to a one with the same width by fusing the two bags (and contracting the edge between them).

3.1 The preprocessing operations

The terminals coloring - . Let G be a graph of treewidth at most k, and let $(T, (X_t)_{t \in V_T})$ be a tree-decomposition of G of width at most k, rooted at t_0. For each t, all the elements in X_t are assigned a color as follows. Every vertex has a color that is the same over all the bags containing it. Two distinct vertices occurring in adjacent bags receive distinct colors. As mentioned in [ALS91], this can be done using at most $2(k+1)$ colors. We denote by $C_1, \ldots, C_q$ the assigned colors. We arrange the set of these colors in a list fixed arbitrarily. For each $t \neq t_0$, we denote by $S(t)$ the induced list of colors occurring in the parent bag of $X(t)$. We set $S(t_0) = \emptyset$. For each t, we also denote by $ADJ(t)$ all the pairs of adjacent colors. Hence, a pair $\{x, y\}$ of adjacent vertices is tracked by the corresponding pair of adjacent colors in all bags containing both x and y. We use the notation $[i - j]$ to denote that color i is adjacent to color j with $i \leq j$. For each t, we organize $ADJ(t)$ in a list with the property that, a pair $[i - j]$ occurs before the pair $[i\prime - j\prime]$ iff $i \leq i\prime$. A tree-decomposition with such a coloring is said to be *colored at the terminals.*

Let $(T,(X_t)_t)$ be a rooted tree-decomposition colored at the terminals. For each node t, we denote by $c(t)$ the number of children of t. Fix a *BFS* ordering of the nodes of T. If $c(t) \neq 0$, let $t^1, t^2, \ldots$ be the induced ordering of the children of t, from the leftmost to the rightmost. The size of the bag X_t is denoted by $x(t)$. For every bag $X(t)$, we fix an enumeration $a_{1,t}, a_{2,t}, \ldots, a_{x(t),t}$ of its elements.

Quartering a bag - . Let X_t be a bag. Consider a path $L(t)$ with $x(t)-1$ nodes: $l_1(t), \ldots, l_{x(t)-1}$ rooted at $l_1(t)$, and, if $x(t) \geq 3$, $l_{i+1}(t)$ is the left child of $l_i(t)$, $1 \leq i \leq x(t)-2$. If $c(t)=0$, then $l_1(t)$ has also the name $\epsilon(t)$. If $c(t) \neq 0$, add a new vertex denoted $\epsilon(t)$. In this case, we make $l_1(t)$ the left child of $\epsilon(t)$. If $c(t) \geq 2$, let $R(t)$ be a path with $c(t)-1$ nodes: $r_1(t), \ldots, r_{c(t)-1}$. Make $r_1(t)$ the right child of $\epsilon(t)$, and if $c(t) \geq 3$, make $r_{i+1}(t)$ the right child of $r_i(t)$, $1 \leq i \leq c(t)-2$. All the nodes of an $L(t)$-path (resp. $R(t)$-path) are of type l (resp. r). A node of the form $\epsilon(t)$ has type b. It has both types b and l if $c(t)=0$; and it has both types b and r if $c(t)=1$.

The binary tree obtained by applying the previous operation is called *the quarter with respect to the node t* (figure 2 in the appendix).

Fixing the terminals of a quarter consists in adding $|X(t)|$ leaves to the quarter with respect to t as follows. The new nodes are organized in a list that is in bijection with the list $\{a_{1,t}, a_{2,t}, \ldots, a_{x(t),t}\}$. Assign the type t (standing for "terminal") to all these new nodes and make $a_{x(t),t}$ the left child of $l_{x(t)-1}(t)$, and for every i, $1 \leq i \leq x(t)-1$, $a_{i,t}$ the right child of $l_i(t)$. The quarter $Q(t)$ for which the fixing terminals operation has been applied is denoted again by $Q(t)$.

Preprocessing a tree-decomposition $(T,(X_t)_t)$ consists in performing the following:

Step1: Define a terminals coloring.
Step2: For each node t, fill the lists $S(t)$ and $ADJ(t)$.
Step3: For each node t, construct the quarter $Q(t)$ and fix the terminals for $Q(t)$.

3.2 Constructing and labeling the proper tree

In binary words, we number the bits starting from 1. We define how all the quarters are connected together. This is performed using:

The affiliation operator. - Let Q and T be two binary trees. Let q_0 be the root of Q, and let t be a node of T that has not a left (resp. right) child. The left (resp. right) *affiliation* of Q in T at t, is the operation that consists in connecting the root of Q and t making q_0 the left (resp. right) child of t. The resulting binary tree is denoted by $Q \nearrow [T,t]$ (resp. $[T,t] \nwarrow Q$).

Now, the preprocessing phase has been performed and we have a family $(Q(t))_{t \in V_T}$ of quarters with fixed terminals for all $t \in V_T$, together with the corresponding family $(S(t), ADJ(t))_{t \in V_T}$ defined above. The following procedure is then called. It proceeds in a top-down traversal of T. During this traversal, at each node t, two $0,1$-words, $W(t)$ and $A(t)$ are set.

- $W(t)$ has length q. The i^{th} bit of $W(t)$ is 1 iff the color $C_i \in S(t)$.
- $A(t)$ has length $q(q+1)/2$. It is decoded as a sequence of q words of length $q, (q-1), \ldots, 1$ respectively. The first word encodes the adjacency list of color 1 (in $X(t)$), the second word encodes the adjacency list of color 2, etc. Hence, for each i, $0 \leq i \leq (q-1)$ and for each j, $i \leq j \leq (q-1)$, $[(i+1)-(j+1)] \in ADJ(t)$ iff the bit of $A(t)$ at position $(2q-i+1)i/2+(j-i)+1$ is set. This means that, in $X(t)$, there is an adjacency between the colors $(i+1)$ and $(j+1)$.

Start with $t = t_0$ and $B := Q(t_0)$, where t_0 denotes the root of the tree-decomposition. All words $W(t)$ and $A(t)$ are zeroed.

```
quarters2ptree($T, t, Q, B$) {
begin
  For each $j$ in $S(t)$
    set the $j^{th}$ bit of $W(t)$;
  For all $i := 0 \ldots (q-1)$
   For all $j := i \ldots (q-1)$
    if $[(i+1)$-$(j+1)] \in ADJ(t)$ then
      set the bit of $A(t)$ at position $(2q-i+1)i/2+(j-i)+1$;
  if $c(t) \neq 0$ then
   begin
    if $c(t) = 1$ then $B := [B, \epsilon(t)] \nwarrow Q(t^1)$;
    else
     begin
      $B := [B, r_{c(t)-1}] \nwarrow Q(t^{c(t)})$;
      For all $i := 1 \ldots c(t) - 1$
        $B := Q(t^i) \nearrow [B, r_i(t)]$;
     end
   end
 For all $i$ such that $t^i$ is a child of $t$
  quarters2ptree($T, t^i, Q, B$);
end; }
```

The tree depicted in figure 3 of appendix is an example of a proper tree obtained from the tree-decomposition in figure 1.

The previous procedure runs in a time linear in the number of nodes in T. It produces a proper tree B together with the family $(W(t), A(t))_{t \in V_T}$ of $0,1$-words. The number of nodes in B is linear in the number of nodes in T.

The labeling system. - We arrange the $P_j's$ predicates in a list fixed arbitrarily.

The nodes of the proper tree are labeled by words $\bar{w} \in \{0,1\}^{q+q(q+1)/2+p+6}$, where p is the number of the $P_j^{\prime}s$ and q is the number of the colors assigned in the preprocessing phase. The labels are of the form $\bar{t}ve\bar{c}\bar{a}\bar{p}$. We begin with all labels cleared with zero. The proper tree B is processed in a top-down traversal.

- $\bar{t} = t_1t_2t_3t_4$ is the type encoding part. The $t_i^{\prime}s$ bits correspond to the types b, t, l and r respectively. For instance, a node with both types b and r has a label beginning with 1001.
- The bit v (resp. e) is set iff the node corresponds to a vertex (resp. an edge) in the graph. Note that, in case one of v or e is set then, the type encoding part is 0100.
- The part $\bar{p}$ encodes the membership to the $P_j^{\prime}s$ of the vertex/edge the current node represents. The $p_j^{\prime}s$ flags are arranged in the same order as in the list of the $P_j^{\prime}s$.
- The color encoding part $\bar{c}$ has length q. If v is set, and if C_i is the color that the corresponding vertex gets in the preprocessing phase, then the i^{th} bit in $\bar{c}$ is set. If e is set, and if C_i and C_j are the colors of the ends of the corresponding edge (with $i = j$ in case of a loop), then the i^{th} and j^{th} bits of $\bar{c}$ are set. The last case where $\bar{c}$ has bits set to 1 is when the current node corresponds to the root of a quarter $Q(s)$. This case occurs iff the current node has bit t_1 set. In this case, we let $\bar{c} = W(s)$.
- The adjacency encoding part $\bar{a}$ has length $q(q+1)/2$. It maintains, for each node of type b, the lists of adjacency in terms of colors. More precisely, if the current node corresponds to the root of a quarter $Q(s)$, then we let $\bar{a} = A(s)$.

Labeling all the proper tree B is performed in a unique traversal of the tree. At each node, the quantity of information to be encoded is of length at most $(k+1)(2k+5)+p+6$, where k is the treewidth of the input graph. Hence, the labeling of the tree is accomplished in a time linear in the size of B.

4 The atomic automata

In this section, we give the automata that correspond to the atomic formulas expressed on graphs and interpreted in proper trees.
If the formula expresses a property of the graph then it is a *sentence* (a closed formula). Every formula can be (re)written using only the logical connectives $\neg$, $\wedge$ and $\exists$. The theoretic automata-operations that are the complementation, the product and the projection are associated to these three connectives respectively. This suggests that the process of constructing the automaton is automatic as soon as the automata corresponding to the interpretation of the atomic predicates are given. All the individual variables in a formula can be treated as set variables which are constrained to be singleton sets. Hence, we deal with set variables only. All the nodes of the tree are labeled by the labeling system in section 3.2. We prefix the labels by 2 bits. The first bit of this prefix, specifies the node of the tree that interprets one of the singleton set variables. If a node has the second bit in the latter prefix set to 1, then it belongs to the set of nodes that interprets the second set variable.

Remark 1. On any graph $\underline{G}$, atomic formulae of the form $X(x)$ where X is a set variable or one of the fixed sets V, E or P_j, $1 \leq j \leq p$, are decidable by a DFTA of at most 3 states.

The transition table of the corresponding automaton is given below. The first and second columns give the states that have been assigned to the left and right child of the current node. The third an fourth columns are the values of the first and second bit in the label of the current node. The character $-$ is the "don't care" symbol. The value of the transition is taken from the first row that matches.

left	*right*	*bit 1*	*bit 2*	δ
s_0	s_0	0	$-$	s_0
s_2	s_0	0	$-$	s_2
s_0	s_2	0	$-$	s_2
s_0	s_0	1	1	s_2
$-$	$-$	$-$	$-$	s_1

The set of accepting states is $\{s_0, s_2\}$.

Wee need the following operations on 0, 1-words.

- *The projection.* - Let l be a positive integer, $\mathcal{S} \subseteq \{1, \ldots, l\}$ a set of s fixed positions and $\bar{w} \in \{0,1\}^l$. We denote by $pr_{/\mathcal{S}}(w) \in \{0,1\}^s$ the projection of $\bar{w}$ on the dimensions in $\mathcal{S}$.
- If w is a binary word, we denote by $subwrd(w, i, n)$ the sub-word of w of length n starting from the bit at the position i.
- Recall that, in $A(t)$, we set the bit at the position $(2q-i+1)i/2+(j-i)+1$ iff, in the bag X_t, color i is adjacent to color j ($j \geq i$). It is convenient to view this position number as being accessed by the following operator, $\mathtt{Dipl(Base, Jump)_q}$. The $\mathtt{Base}$ argument is a word of length $q(q+1)/2$. The $\mathtt{Jump}$ argument is a non-negative integer. It indicates that the information we want to access in the $\mathtt{Base}$ argument has to be searched immediately after $(\mathtt{2q} - \mathtt{Jump} + 1)\mathtt{Jump}/2$ bit-positions further from the beginning of $\mathtt{Base}$. The value of the $\mathtt{Dipl}$ argument is at least the value of $\mathtt{Jump}$. It is used to perform the final displacement that accesses the desired bit. The amount of this additional displacement is $(\mathtt{Displ} - \mathtt{Jump} + 1)$. For instance, if we want to encode that, in X_t, color 3 is adjacent to color 7, we set, in $\mathtt{A(t)}$, the bit at the position $\mathtt{6(A(t), 2)_q}$. If we want to encode that, in X_t, color 1 is adjacent to color 1, we set, in $\mathtt{A(t)}$, the bit at the position $\mathtt{0(A(t), 0)_q}$.

Sketch of the proofs of theorem 1 and theorem 2. - The labels are of length $l = q + q(q+1)/2 + p + 8$. Let $f : V_B \to \{0,1\}^l$ be the corresponding mapping. For each of the proofs, we define below a set $\mathcal{S}$ of bit-positions. Let $g = pr_{/\mathcal{S}} \circ f$. We consider the resulting labels. The corresponding binary words are renumbered starting from 1. Each of the automata will be described using a recursive function whose argument is a proper tree labeled by g. For the sake of simplicity, each of the three functions will be referred to by the same name `run`. The functions assign a record value to each node, during a bottom-up traversal. For each function, we define below the set of record values it uses. The field values in the records have the following nice feature. If the transition of the corresponding automaton is symmetric relatively to the states that have been assigned to the left and right child, then `run` captures all the corresponding rows of the automaton by the result of a single operation. The operands of the latter operation are field values from the records that have been assigned to the children of the current node. The functions are entirely described in the appendix.

In our notation, the fields are separated by the dot symbol. If R is a record, we write $R_{\to 1}$ for the first field, $R_{\to 2}$ for the second one, etc. If B is a proper tree, we denote by B_l (resp. B_r) the left (resp. right) subtree of B.

In the proof of theorem 1,

(i) $\mathcal{S} = \{1, \ldots, 4, 7, \ldots, q+8\}$. The records have two fields. The first field "tells", for each node, whether it is or not in the first/second set variable. At each node corresponding to a vertex, the second field keeps track of the color. If the node is not a terminal, the second field of the assigned record reports the information it receives from its subtree about the colors. We use the value 0, if we want to specify that we do not care about the color of the node. There are $2q+3$ code record values for the terminals:

code record	meaning
0.0	Bits at positions $1, 2, 4$ are set
1.0	The label begins with 00
4.0	Bit $4, 6$ are set and the label does not begin with 11
2.i	Bits $1, 4, 5, i+6$ are set
3.i	Bits $2, 4, 5, i+6$ are set

A state is associated to each code record. The state associated to the code record value 1.0 is the initial state. The state associated to the code record value 0.0 is the unique accepting state.

(ii) $\mathcal{S}$ is the same as the one of the previous function. The records have 3 fields. The first one has the same role as above. The next fields either deal with the color(s) of the node itself if it is a terminal, or report what the node "learns" from its subtree about the colors. We use the code 0.0 if we want to specify that we do not care about the color of the node. There are $q + 2 + q(q+1)/2$ code record values for the terminals:

code record	meaning
1.0.0	Bit 4 is set and the label begins with 00
4.0.0	Bits $1, 2, 4$ are set
2.i.0	Bits $1, 4, 5, i + 6$ are set
3.i.j	Bits $2, 4, 6, i + 6, j + 6$ are set

In the proof of theorem 2, $\mathcal{S} = \{1, \ldots, 4, 7, \ldots, q+8+q(q+1)/2\}$. The records have 3 fields and have the same role as for the function corresponding to the incidence relation. There are $3q + 2$ code record values for the terminals:

code record	meaning
1.0.0	Bits 4 is set and the label begins with 00
4.0.0	Bits $4, 6$ are set and the label does not begin with 00
2.i.0	Bits $1, 4, 5, i + 6$ are set
3.i.0	Bits $2, 4, 5, i + 6$ are set
6.i.0	Bits $1, 2, 4, 5$ are set

References

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decompsable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[Bod96] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[Cou90] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990. Presented at the International Workshop on Graph Grammars and their application to computer Science, Warrenton, Virginia, 1986. LNCS Comp. Sci. vol-291, 133-146.

[Cou92] B. Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *Theoretical Informatics and Applications*, 26(3):257–286, 1992.

[Cou97a] B. Courcelle. *The expression of graph properties and graph transformations in monadic second-order logic.,* Handbook of graph grammars and computing by graph transformation, Vol. I: Foundations, pages 313–400. G. Rozenberg ed., World Scientific Publishing Co., Inc., 1997.

[Cou97b] B. Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 31:33–62, 1997.

[Don65] J. E. Doner. Decidability of the weak second-order theory of two successors. *Notices Amer. Math. Soc.*, 12:365–468, March 1965.

[FHL05] U. Feige, M. T. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum-weight vertex separator. In *proceedings of the* 37^{th} *Annual ACM symposium on Theory of Computing (STOC)*, pages 563–572, 2005.

[FV59] S. Feferman and R. Vaught. The first order properties of products of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

[Joh87] David S. Johnson. The NP-completeness column: An ongoing guide: The many faces of polynomial time. *Journal of Algorithms*, 8(2):285–303, June 1987.

[Mak04] J. A. Makowsky. Algorithmic uses of the feferman-vaught theorem. *Annals of Pure and Applied Logic*, 126:159–213, 2004.

[RS86] Robertson and Seymour. Graph minors II: Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[She75] S. Shelah. The monadic theory of order. *Annals of Maths*, 102:379–419, 1975.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, March 1968.

Appendix

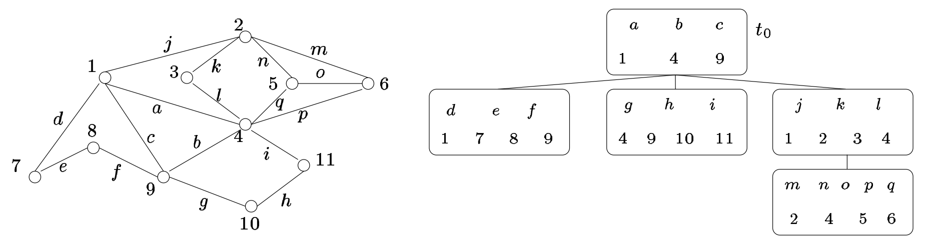
Fig. 1. A graph G and a tree-decomposition for G

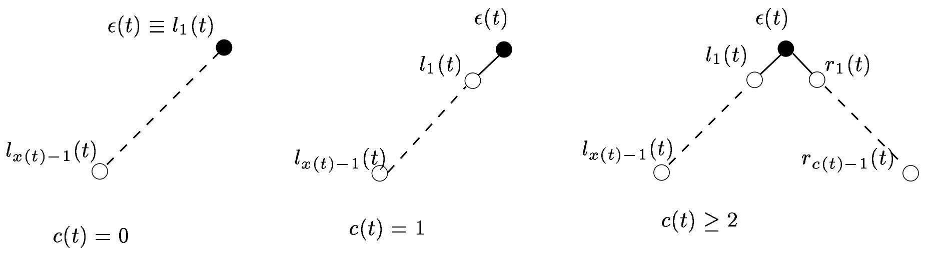

Fig. 2. The quarter with respect to a node t

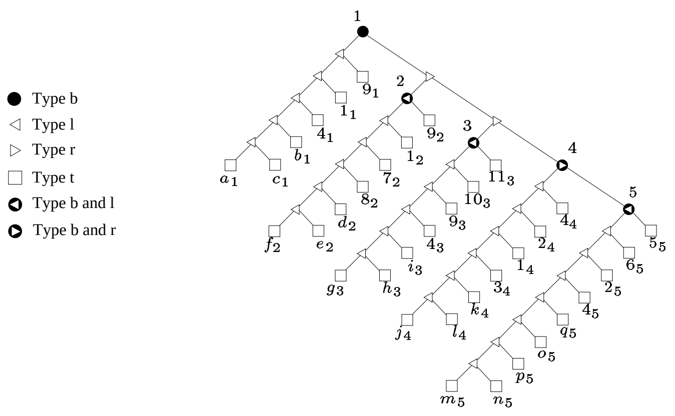

Fig. 3. The proper tree from the tree-decomposition in figure 1

run **for the equality**

```
run(B) {
begin
lab = g(root(B)) ;
if subwrd(lab, 4, 1) = 1 {This is a terminal} then
 begin
  w = subwrd(lab, 1, 2) ;
  if w = 00 then return(1.0);
  if w = 11 then return(0.0);
  if subwrd(lab, 5, 1) = 1 {This is a vertex} then
   begin
    for i = 1 ... q do
     begin
      if subwrd(lab, i + 6, 1) = 1 then
       begin
        if subwrd(lab,1,1)=1 then return(2.i);
        return(3.i);
       end
     end
   end
  return(4.0); {An edge that is not in both set variables }
 end
{If we are taken here, the current node is not a terminal}
if subwrd(lab, 1, 2) ≠ 00 then return(4.0);

{If we are taken here, the current label begins with 00}
l = run(B_l) ; r = run(B_r) ;
if ((l_→1) * (r_→1) = 1) then return(1.0);
if ((l_→1) * (r_→1) = 2) then
 begin
  i = (l_→2) + (r_→2) ;
  if (subwrd(lab, 3, 1) = 0) then return(2.i);
  {If we are taken here, the current node corresponds to a bag}
  if (subwrd(lab, i + 6, 1) = 1) {The parent bag contains color C_i} then
   return(2.i);
  {If we are taken here, the color vanishes or the last set variable
  is empty, hence not a singleton}
  return(4.0);
 end
if ((l_→1) * (r_→1) = 3) then
 begin
  i = (l_→2) + (r_→2) ;
  if (subwrd(lab, 3, 1) = 0) then return(3.i);
  if (subwrd(lab, i + 6, 1) = 1) then return(3.i);
  {If we are taken here, the color vanishes or the first set
  variable is empty, hence not a singleton}
  return(4.0);
 end
if ((l_→1) * (r_→1) = 6) and (l_→2 = r_→2) then return(0.0);
if ((l_→1) + (r_→1) = 1) then return(0.0);
{All other cases}
return(4.0);
end
}
```

`run` **for the incidence**

```
run(B) {
begin
lab = g(root(B)) ;
if subwrd(lab, 4, 1) = 1 {This is a terminal} then
 begin
  w = subwrd(lab, 1, 2) ;
  if w = 00 then return(1.0.0) ;
  if w = 11 then return(4.0.0);
  if subwrd(lab, 5, 1) = 1 {This is a vertex} then
   begin
    for i = 1...q do
     begin
      if subwrd(lab, i + 6, 1) = 1 then
       begin
        if subwrd(lab,1,1)=1 then return(2.i.0)
        return(4.0.0);
       end
     end
    end
  else {This is an edge}
   begin
    for i = 1...q do
     begin
      if subwrd(lab, i + 6, 1) = 1 then
       begin
        for j = (i + 1)...q do
         begin
          if subwrd(lab, j + 6, 1) = 1 then
           begin
            if subwrd(lab,2,1)=1 then return(3.i.j);
            return(4.0.0);
           end
         end
        if subwrd(lab,2,1)=1 then return(3.i.i);
        return(4.0.0);
       end
     end
   end
 end
{If we are taken here, the current node is not a terminal}
if subwrd(lab, 1, 2) ≠ 00 then return(4.0.0) ;
```

Incidence (1/2)

```
{If we are taken here, the current label begins with 00}
l = run(B_l) ; r = run(B_r) ;
if (l→1 * r→1 = 1) then return(1.0.0) ;
if (l→1 * r→1 = 2) then
 begin
  i = l→2 + r→2 ; {Only one of the two is not 0 }
  if (subwrd(lab, 3, 1) = 0) {Not a bag} then return(2.i.0) ;
  if (subwrd(lab, i + 6, 1) = 1) {The parent bag contains color C_i} then
   return(2.i.0) ;
  {If we are taken here, the color vanishes or the other set
  variable is empty, hence not a singleton}
  return(4.0.0) ;
 end
if (l→1 * r→1 = 3) then
 begin
  i = l→2 + r→2 ; j = l→3 + r→3 ;
  if (subwrd(lab, 3, 1) = 0) then return(3.i.j) ;
  β = subwrd(lab, i + 6, 1) ; γ = subwrd(lab, j + 6, 1) ;
  if (β = 0 and γ = 0) then
  {Both end colors vanish or the other set variable is empty,
  hence not a singleton}
   return(4.0.0) ;
  return(3.(i * β).(j * γ)) ;
 end
if (l→1 * r→1 = 6) then
{One child has been assigned 2. * .0,
the other one has been assigned 3. * .*}
 begin
  if (l→1 = 2) then
    c = l→2 ; i = r→2; j = r→3 ;
  else
    c = r→2 ; i = l→2; j = l→3 ;
  if (c = i or c = j) then return(0.0.0);
  return(4.0.0) ;
 end
if (l→1 + r→1 = 1) then return(0.0.0) ;
{All other cases}
return(4.0.0) ;
end
}
```

Incidence (2/2)

The state associated to the code record value 1.0.0 is the initial state. The state associated to the code record value 0.0.0 is the unique accepting state.

run for the Adjacency

```
run(B) {
begin
lab = g(root(B)) ;
if subwrd(lab, 1, 4) = 1 then {A terminal}
 begin
  w = subwrd(lab, 1, 2) ;
  if w = 00 then return(1.0.0);
  if subwrd(lab, 5, 1) = 1 {A vertex} then
   begin
    if w = 11 then
     begin
      for i = 1...q do
       begin
        if subwrd(lab, i + 6, 1) = 1 then return(6.i.0);
       end
     end
    for i = 1...q do
     begin
      if subwrd(lab, i + 6, 1) then
       begin
        if subwrd(lab, 1, 1) = 1 return(2.i.0);
        return(3.i.0);
       end
     end
   end
  return(4.0.0); {An edge in one of the set variables}
 end
{If we are taken here, the current node is not a terminal}
if subwrd(lab, 1, 2) ≠ 00 then return(4.0.0);

{If we are taken here, the current label begins with 00}
l = run(B_l) ; r = run(B_r) ;
if ((l_→1) * (r_→1) = 1) then return(1.0.0);
if ((l_→1) * (r_→1) = 2) then
 begin
  i = (l_→2) + (r_→2) ;
  if (subwrd(lab, 3, 1) = 0) {Not a bag} then return(2.i.0);
  if (subwrd(lab, i + 6, 1) = 1) {The parent bag contains color C_i} then
   return(2.i.0);
  {If we are taken here, the color vanishes or the last set variable
  is empty, hence not a singleton}
  return(4.0.0);
 end
```

Adjacency (1/2)

```
if $((l_{\to 1}) * (r_{\to 1}) = 3)$ then
 begin
  $i = (l_{\to 2}) + (r_{\to 2})$ ;
  if $(subwrd(lab, 3, 1) = 0)$ then return$(3.i.0)$;
  if $(subwrd(lab, i+6, 1) = 1)$ then return$(3.i.0)$;
  return$(4.0.0)$; {the color vanishes}
 end
if $((l_{\to 1}) * (r_{\to 1}) = 6)$ then
{One of the two cases: 1) One child has been assigned $2.\beta.0$,
the other one has been assigned $3.\gamma.0$, or 2) One child has been
assigned $6.\beta.\gamma$, the other one has been assigned $1.0.0$}
 begin
  $i = l_{\to 2}$ ; $j = r_{\to 2}$ ;
  if $i * j \neq 0$ { Case 1) } then
   begin
    if (i>j) then
     begin k=i ; i=j ; j=k; end
   end
  else {Case 2)} begin $i = (l_{\to 2}) + (r_{\to 2})$; $j = (l_{\to 3}) + (r_{\to 3})$; end
  if $(subwrd(lab, 3, 1) = 0)$ then
   begin
    if i=j then {Case 1)} return$(6.i.0)$;
    {If we are taken here then,
    it is either case 1) with $0 < i < j$
    or case 2) with $(i < j$ or $j = 0)$}
    return$(6.i.j)$;
   end
  $w = subwrd(lab, q+7, q(q+1)/2)$ ; {the adjacency encoding part }
  if at the position $(j-1)(w, (i-1))_q$ is 1 then return$(0.0.0)$;
 end
if $((l_{\to 1}) + (r_{\to 1}) = 1)$ then return$(0.0.0)$;
{All other cases}
return$(4.0.0)$;
end
}
```

Adjacency (2/2)

The state associated to the code record value 1.0.0 is the initial state. The state associated to the code record value 0.0.0 is the unique accepting state.