Journal of Graph Algorithms and Applications
http://jgaa.info/
vol. 7, no. 2, pp. 141–180 (2003)

Deciding Clique-Width for Graphs of Bounded Tree-Width

Wolfgang Espelage *Frank Gurski* *Egon Wanke*
Department of Computer Science
D-40225 Düsseldorf, Germany
http://www.cs.uni-duesseldorf.de/
{espelage,gurski,wanke}@cs.uni-dueseldorf.de

Abstract

We show that there exists a linear time algorithm for deciding whether a graph of bounded tree-width has clique-width k for some fixed integer k.

Communicated by Giuseppe Liotta and Ioannis G. Tollis: submitted October 2001; revised July 2002 and February 2003.

The work of the second author was supported by the German Research Association (DFG) grant WA 674/9-2.

1 Introduction

The clique-width of a graph is defined by a composition mechanism for vertex-labeled graphs, see [CO00]. The operations are the vertex disjoint union $G \oplus H$ of two graphs G and H, the addition of edges $\eta_{i,j}(G)$ between vertices labeled by i and vertices labeled by j, and the relabeling $\rho_{i \to j}(G)$ of vertices labeled by i into vertices labeled by j. The clique-width of a graph G is the minimum number of labels needed to define G.

Graphs of bounded clique-width are interesting from an algorithmic point of view. A lot of NP-complete graph problems can be solved in polynomial time for graphs of bounded clique-width if the composition of the graph is explicitly given. For example, all graph properties which are expressible in monadic second order logic with quantifications over vertices and vertex sets (MSO_1-logic) are decidable in linear time on graphs of bounded clique-width, see [CMR00]. The MSO_1-logic has been extended by counting mechanisms which allow the expressibility of optimization problems concerning maximal or minimal vertex sets, see [CMR00]. All these graph problems expressible in extended MSO_1-logic can be solved in polynomial time on graphs of bounded clique-width. Furthermore, a lot of NP-complete graph problems which are not expressible in MSO_1-logic or extended MSO_1-logic like Hamiltonicity and a lot of partitioning problems can also be solved in polynomial time on graphs of bounded clique-width, see [EGW01, KR01, Wan94].

The following facts are already known about graphs of bounded clique-width. If a graph G has clique-width at most k then the edge complement $\overline{G}$ has clique-width at most $2k$, see [CO00]. Distance hereditary graphs have clique-width at most 3, see [GR00]. The set of all graphs of clique-width at most 2 is the set of all cographs. The clique-width of permutation graphs, interval graphs, grids and planar graphs is not bounded by some fixed integer, see [GR00]. An arbitrary graph with n vertices has clique-width at most $n - r$, if $2^r < n - r$, see [Joh98].

One of the central open questions concerning clique-width is determining the complexity of recognizing and finding a decomposition with clique-width operations of graphs of clique-width at most k, for fixed $k \geq 4$. Clique-width of at most 2 is decidable in linear time, see [CPS85]. Clique-width of at most 3 is decidable in polynomial time, see [CHL$^+$00]. The recognition problem for graphs of clique-width at most k is still open for $k \geq 4$. The complexity of the minimization problem where k is additionally given to the input is also open, i.e., not known to be NP-complete nor known to be solvable in polynomial time.

A famous class of graphs for which a lot of NP-complete graph problems can be solved in polynomial time is the class of graphs of bounded tree-width, see Bodlaender [Bod98] for a survey. For every fixed integer l, it is decidable in linear time whether a given graph G has tree-width l, see [Bod96]. All graph properties expressible in monadic second order logic with quantifications over vertex sets and edge sets (MSO_2-logic) are decidable in linear time for graphs of bounded tree-width by dynamic programming, see [Cou90]. The MSO_2-logic has also been extended by counting mechanisms to express optimization problems which can then be solved in polynomial time for graphs of bounded tree-width,

see [ALS91].

Clique-width seems to be "more powerful" than tree-width. Every graph of tree-width at most l has clique-width at most $3 \cdot 2^{l-1}$, see [CR01]. Since the set of all cographs already contains all complete graphs, the set of all graphs of clique-width at most 2 does not have bounded tree-width. In [GW00], it is shown that every graph of clique-width at most k which does not contain the complete bipartite graph $K_{n,n}$ for some $n > 1$ as a subgraph has tree-width at most $3k(n-1)-1$.

An algorithm to decide a graph property on a graph of bounded tree-width can simply be obtained by partitioning the set of all so-called *l-terminal* graphs into a finite number of equivalence classes as follows. An l-terminal graph is a graph with a list of l distinct vertices called terminals. Two l-terminal graphs G and H can be combined to a graph $G \circ H$ by taking the disjoint union of G and H and then identifying the i-th terminal of G with the i-th terminal of H for $1 \leq i \leq l$. They are called *replaceable* with respect to a graph property Π if for all l-terminal graphs J the answer to Π is the same for $G \circ J$ and $H \circ J$. Replaceability is obviously an equivalence relation. A graph property Π is decidable in linear time on a graph of bounded tree-width if there is a finite number of equivalence classes with respect to Π for all l-terminal graphs and all $l \geq 0$. The linear time algorithm first computes a binary tree-decomposition T_D for G and then bottom-up the equivalence class for every l-terminal graph G' represented by a complete subtree T'_D of T_D. The equivalence class of G' defined by subtree T'_D with root u' is computable in time $O(1)$ from the classes of the two l-terminal graphs defined by the two subtrees in $T'_D - \{u'\}$, see also [Arn85, ALS91, AP89, Bod97, Cou90, LW88, LW93].

In this paper, we prove that the graph property "clique-width at most k" divides the set of all l-terminal graphs into a finite number of equivalence classes. This implies that there exists a linear time algorithm for deciding "clique-width at most k" for graphs of bounded tree-width. Since every graph of tree-width l has clique-width at most $3 \cdot 2^{l-1}$, there is also a linear time algorithm for computing the "exact clique-width" of a graph of bounded tree-width by testing "clique-width at most k" for $k = 1, \ldots, 3 \cdot 2^{l-1}$. Note that it remains still open whether the clique-width k property is expressible in MSO_2-logic and whether "clique-width at most k" is decidable in polynomial time for arbitrary graphs.

The paper is organized as follows. In Section 2 we define the clique-width of vertex labeled graphs. Every graph of clique-width at most k is defined by a k-expression X.

In Section 3, we define the k-expression tree of a k-expression. Every k-expression defines a unique k-expression tree and every k-expression tree defines a unique k-expression. We will mostly work with the expression tree instead of the expression, because many transformation steps are easier to explain for expression trees than for expressions.

In Section 4, we define a normal form for a k-expression. We show that for every k-expression there is an equivalent one in normal form.

In Section 5, we define l-terminal graphs, for some nonnegative integer l, and

an equivalence relation on the set of all l-terminal graphs, called replaceability. This equivalence relation is defined with respect to the graph property clique-width at most k. If the relation has a finite number of equivalence classes, then the graph property clique-width at most k is decidable in linear time on graphs of tree-width at most l by dynamic programming algorithms.

In Section 6, we give an overview about the proof of the main result.

In Section 7, we show that every combined graph $H \circ J$ of clique-width at most k can be defined by a k-expression in normal form whose expression tree satisfies further special properties concerning the composition of H and J to $H \circ J$. Every of these special expression trees defines a connection tree for H. The set of all connection trees for H is called the connection type of H. For fixed integers k and l, there is only a fixed number of mutually different connection trees and thus a fixed number of connection types.

In Section 8, we show that if two l-terminal graphs H_1 and H_2 define the same connection type then they are replaceable with respect to property clique-width at most k. This shows that the equivalence relation defined in Section 5 has a finite number of equivalence classes, which implies that graph property clique-width at most k is decidable in linear time for graphs of bounded tree-width.

2 Clique-width

We work with finite undirected *graphs* $G = (V_G, E_G)$, where V_G is a finite set of *vertices* and $E_G \subseteq \{\{u, v\} \mid u, v \in V_G,\ u \neq v\}$ is a finite set of *edges*. A graph $J = (V_J, E_J)$ is a *subgraph* of G if V_J is a subset of V_G and E_J is a subset of $E_G \cap \{\{u, v\} \mid u, v \in V_J,\ u \neq v\}$. J is an *induced subgraph* of G if additionally $E_J = \{\{u, v\} \in E_G \mid u, v \in V_J\}$. G and J are *isomorphic* if there is a bijection $b : V_G \to V_J$ such that for every pair of vertices $u, v \in V_G$, $\{u, v\}$ is an edge of G if and only if $\{b(u), b(v)\}$ is an edge of J. To distinguish between the vertices of (non-tree) graphs and trees, we simply call the vertices of the trees *nodes*.

The notion of clique-width for labeled graphs is first defined by Courcelle and Olariu in [CO00]. Let $[k] := \{1, \ldots, k\}$ be the set of all integers between 1 and k. A *k-labeled graph* $G = (V_G, E_G, \mathrm{lab}_G)$ is a graph (V_G, E_G) whose vertices are labeled by a mapping $\mathrm{lab}_G : V_G \to [k]$. The k-labeled graph consisting of a single vertex labeled by some label $t \in [k]$ is denoted by $\bullet_t$. A k-labeled graph $J = (V_J, E_J, \mathrm{lab}_J)$ is a *k-labeled subgraph* of G if $V_J \subseteq V_G$, $E_J \subseteq E_G \cap \{\{u, v\} \mid u, v \in V_J,\ u \neq v\}$ and $\mathrm{lab}_J(u) = \mathrm{lab}_G(u)$ for all $u \in V_J$. G and J are *isomorphic* if there is a bijection $b : V_G \to V_J$ such that $\{u, v\} \in E_G$ if and only if $\{b(u), b(v)\} \in E_J$, and for every vertex $u \in V_G$, $\mathrm{lab}_G(u) = \mathrm{lab}_J(b(u))$.

Definition 2.1 (Clique-width, [CO00]) *Let k be some positive integer. The class CW_k of k-labeled graphs is recursively defined as follows.*

1. *The k-labeled graphs $\bullet_t$ for $t \in [k]$ are in CW_k.*

2. *Let* $G = (V_G, E_G, lab_G) \in CW_k$ *and* $J = (V_J, E_J, lab_J) \in CW_k$ *be two vertex disjoint* k*-labeled graphs. Then the* k*-labeled graph*

$$G \oplus J := (V', E', lab')$$

defined by $V' := V_G \cup V_J$, $E' := E_G \cup E_J$, *and*

$$lab'(u) \ := \ \begin{cases} lab_G(u) & \text{if } u \in V_G \\ lab_J(u) & \text{if } u \in V_J \end{cases}, \ \forall u \in V'$$

is in CW_k.

3. *Let* $i, j \in [k]$, $i \neq j$, *be two distinct integers and* $G = (V_G, E_G, lab_G) \in CW_k$ *be a* k*-labeled graph then*

   (a) *the* k*-labeled graph* $\eta_{i,j}(G) \ := \ (V_G, E', lab_G)$ *defined by*

   $$E' := E_G \cup \{\{u, v\} \mid \ u, v \in V_G, \ u \neq v, \ lab(u) = i, \ lab(v) = j\}$$

   is in CW_k *and*

   (b) *the* k*-labeled graph* $\rho_{i \to j}(G) := (V_G, E_G, lab')$ *defined by*

   $$lab'(u) \ := \ \begin{cases} lab_G(u) & \text{if } lab_G(u) \neq i \\ j & \text{if } lab_G(u) = i \end{cases}, \ \forall u \in V_G$$

   is in CW_k.

An expression X built with the operations $\bullet_t, \oplus, \eta_{i,j}, \rho_{i \to j}$ for integers $t, i, j \in [k]$ is called a *k-expression*. To distinguish between the k-expression and the graph defined by the k-expression, we denote by $\mathrm{val}(X)$ the graph defined by expression X. That is, CW_k is the set of all graphs $\mathrm{val}(X)$, where X is a k-expression.

We say, a k-labeled graph G has *clique-width at most* k if G is contained in class CW_k, i.e., the set CW_k is the set of all k-labeled graphs of *clique-width at most* k. The *clique-width* of a k-labeled graph G is the smallest integer k such that G has clique-width at most k.

We sometimes use the simplified notions *labeled graph* and *expression* for a k-labeled graph and a k-expression, respectively. In these cases, however, either k is known from the context, or k is irrelevant for the discussion.

An *unlabeled* graph $G = (V_G, E_G)$ has *clique-width at most* k if there is some labeling $\mathrm{lab}_G : V_G \to [k]$ of the vertices of G such that the labeled graph $G' = (V_G, E_G, \mathrm{lab}_G)$ has clique-width at most k. The *clique-width* of an *unlabeled* graph $G = (V_G, E_G)$ is the smallest integer k such that there is some labeling $\mathrm{lab}_G : V_G \to [k]$ of the vertices of G such that the labeled graph $G' = (V_G, E_G, \mathrm{lab}_G)$ has clique-width at most k.

If X is a k-expression then obviously $\rho_{i \to 1}(X)$ is a k-expression for all $i \in [k], i > 1$. For the rest of this paper, we consider an unlabeled graph as a labeled graph in that all vertices are labeled by the same label, which is without loss of generality label 1. This allows us to use the notation "graph" without any confusion for labeled and unlabeled graphs.

3 Expression tree

Every k-expression X has by its recursive definition a tree structure that is called the *k-expression tree* T for X. It is an ordered rooted tree whose nodes are labeled by the operations of the k-expression and whose arcs are directed from the leaves towards the root of T. The root of T is labeled by the last operation of the k-expression.

Definition 3.1 (Expression tree) *The* k-expression tree T *for k-expression* $\bullet_t$ *consists of a single node r (the* root *of T) labeled by* $\bullet_t$*. The* k-expression tree T *for* $\eta_{i,j}(X)$ *and* $\rho_{i\to j}(X)$ *consists of a copy T' of the k-expression tree for X, an additional node r (the* root *of T) labeled by* $\eta_{i,j}$ *or* $\rho_{i\to j}$*, respectively, and an additional arc from the root of T' to node r. The* k-expression tree T *for* $X_1 \oplus X_2$ *consists of a copy T_1 of the k-expression tree for X_1, a copy T_2 of the k-expression tree for X_2, an additional node r (the* root *of T) labeled by* $\oplus$ *and two additional arcs from the roots of T_1 and T_2 to node r. The root of T_1 is the* left child *of r and the root of T_2 is the* right child *of r.*

A node of T labeled by $\bullet_t$*,* $\eta_{i,j}$*,* $\rho_{i\to j}$*, or* $\oplus$ *is called a* leaf*,* edge insertion node*,* relabeling node*, or* union node*, respectively.*

If integer k is known from the context or irrelevant for the discussion, then we sometimes use the simplified notion *expression tree* for the notion k-expression tree. The leaves of expression tree T for expression X correspond to the vertices of graph $\mathrm{val}(X)$. For some node u of expression tree T, let $T(u)$ be the subtree of T induced by node u and all nodes of T from which there is a directed path to u. Note that $T(u)$ is always an expression tree. The expression $X(u)$ defined by $T(u)$ can simply be determined by traversing the tree starting from the root, where the left children are visited first. The vertices of G' are the vertices of G corresponding to the leaves of $T(u)$. The edges of G' and the labels of the vertices of G' are defined by expression $X(u)$. For two vertices u, v of G', every edge $\{u, v\}$ of G' is also in G but not necessarily vice versa. Two equal labeled vertices in G' are also equal labeled in G but not necessarily vice versa. The labeled graph G' is denoted by $G(T, u)$ or simply $G(u)$, if tree T is unique from the context. Figure 1 illustrates these notations.

4 Normal form

We next define a so-called *normal form* for a k-expression. This normal form does not restrict the class of k-labeled graphs that can be defined by k-expressions, but is very useful for the proof of our main result.

To keep the definition of our normal form as simple as possible, we enumerate the vertices in a graph $G = \mathrm{val}(X)$ defined by some k-expression X as follows. The single vertex in $\mathrm{val}(\bullet_t)$ is the *first vertex* of $\mathrm{val}(\bullet_t)$. Let $G = \mathrm{val}(Y_1 \oplus Y_2)$. If $\mathrm{val}(Y_1)$ has n vertices and $\mathrm{val}(Y_2)$ has m vertices, then for $i = 1, \ldots, n$ the *i-th vertex* of G is the i-th vertex of $\mathrm{val}(Y_1)$ and for $i = n+1, \ldots, n+m$ the *i-th vertex* of G is the $(i-n)$-th vertex of $\mathrm{val}(Y_2)$. The *i-th vertex* of $\mathrm{val}(\eta_{i,j}(Y))$

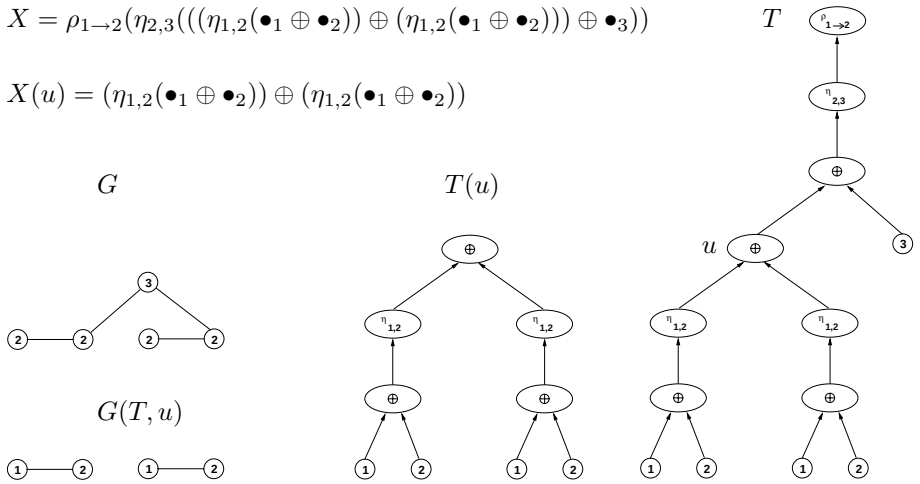


Figure 1: A 3-labeled graph G defined by a 3-expression X with 3-expression tree T, and a 3-labeled graph $G(T,u)$ defined by 3-expression $X(u)$ with 3-expression tree $T(u)$

and val$(\rho_{i\to j}(Y))$ is the i-th vertex of val(Y). We say two expressions X and Y are *equivalent*, denoted by $X \equiv Y$, if val(X) and val(Y) are isomorphic in consideration of the order of the vertices, that is,

1. val(X) and val(Y) have the same number n of vertices,
2. the i-th vertex in val(X), $1 \le i \le n$, has the same label as the i-th vertex in val(Y), and
3. there is an edge between the i-th and j-th vertex in val(X), $1 \le i, j \le n$, if and only if there is an edge between the i-th and j-th vertex in val(Y).

Otherwise X and Y are *not equivalent*, denoted by $X \not\equiv Y$.

If two k-expressions X and Y are equivalent then they do not need to be equal. If two k-labeled graphs val(X) and val(Y) defined by two k-expressions X and Y are isomorphic (see the second paragraph of Section 2) then X and Y do not need to be equivalent. Figure 2 shows an example.

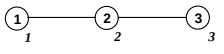
Definition 4.1 (Normal form) *Our* normal form *for k-expressions is defined as follows.*

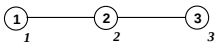
1. *The k-expression $\bullet_t$ for some $t \in [k]$ is in normal form.*
2. *If Y_1 and Y_2 are two k-expressions in normal form then the k-expression*

$$\rho_{i_n\to j_n}(\cdots \rho_{i_1\to j_1}(\eta_{i'_{n'},j'_{n'}}(\cdots \eta_{i'_1,j'_1}(Y_1 \oplus Y_2)\cdots))\cdots)$$

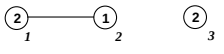
for $i_1, j_1, \ldots, i_n, j_n, i'_1, j'_1, \ldots, i'_{n'}, j'_{n'} \in [k]$ is in normal form if the following properties hold true.

$X_1 = \eta_{2,3}((\eta_{1,2}(\bullet_1 \oplus \bullet_2)) \oplus \bullet_3) \quad \equiv \quad X_2 = \eta_{1,2}(\bullet_1 \oplus (\eta_{2,3}(\bullet_2 \oplus \bullet_3)))$

$\mathrm{val}(X_1)$: 

$\mathrm{val}(X_2)$: 

$X_3 = (\eta_{1,2}(\bullet_2 \oplus \bullet_1)) \oplus \bullet_2 \quad \not\equiv \quad X_4 = \bullet_2 \oplus (\eta_{1,2}(\bullet_1 \oplus \bullet_2))$

$\mathrm{val}(X_3)$: 

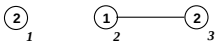
$\mathrm{val}(X_4)$: 

Figure 2: The small indices at the vertices represent their numbering with respect to the corresponding k-expression. The expressions X_1 and X_2 are equivalent but not equal. The labeled graphs $\mathrm{val}(X_3)$ and $\mathrm{val}(X_4)$ are isomorphic but the expressions X_3 and X_4 are not equivalent.

(a) For every edge insertion operation $\eta_{i'_{l'},j'_{l'}}$, $1 \le l' \le n'$,

$$\eta_{i'_{l'},j'_{l'}}(\eta_{i'_{l'-1},j'_{l'-1}}(\cdots \eta_{i'_1,j'_1}(Y_1 \oplus Y_2)\cdots)) \not\equiv \eta_{i'_{l'-1},j'_{l'-1}}(\cdots \eta_{i'_1,j'_1}(Y_1 \oplus Y_2)\cdots),$$

$$\eta_{i'_{l'},j'_{l'}}(Y_1) \equiv Y_1, \quad \textit{and} \quad \eta_{i'_{l'},j'_{l'}}(Y_2) \equiv Y_2.$$

(b) For every relabeling operation $\rho_{i_l \to j_l}$, $1 \le l \le n$, *graph*

$$val(\rho_{i_{l-1}\to j_{l-1}}(\cdots \rho_{i_1 \to j_1}(\eta_{i'_{n'},j'_{n'}}(\cdots \eta_{i'_1,j'_1}(Y_1 \oplus Y_2)\cdots))\cdots))$$

has a vertex labeled by i_l *and a vertex labeled by* j_l, *and* $i_l \notin \{j_1, \ldots, j_{l-1}\}$.

(c) For every pair of two distinct labels $i, j \in [k], i \ne j$,

i. if $val(Y_1)$ *has a vertex labeled by* i *and a vertex labeled by* j *then*

$$\begin{aligned} &\rho_{i_l\to j_l}(\cdots \rho_{i_1\to j_1}(\eta_{i'_{n'},j'_{n'}}(\cdots \eta_{i'_1,j'_1}(\quad Y_1 \quad \oplus \quad Y_2 \quad)\cdots))\cdots) \\ \not\equiv\ &\rho_{i_l\to j_l}(\cdots \rho_{i_1\to j_1}(\eta_{i'_{n'},j'_{n'}}(\cdots \eta_{i'_1,j'_1}(\quad \rho_{i\to j}(Y_1) \quad \oplus \quad Y_2 \quad)\cdots))\cdots) \end{aligned}$$

and

ii. if $val(Y_2)$ *has a vertex labeled by* i *and a vertex labeled by* j *then*

$$\begin{aligned} &\rho_{i_l\to j_l}(\cdots \rho_{i_1\to j_1}(\eta_{i'_{n'},j'_{n'}}(\cdots \eta_{i'_1,j'_1}(\quad Y_1 \quad \oplus \quad Y_2 \quad)\cdots))\cdots) \\ \not\equiv\ &\rho_{i_l\to j_l}(\cdots \rho_{i_1\to j_1}(\eta_{i'_{n'},j'_{n'}}(\cdots \eta_{i'_1,j'_1}(\quad Y_1 \quad \oplus \quad \rho_{i\to j}(Y_2) \quad)\cdots))\cdots). \end{aligned}$$

If X is a k-expression in normal form then the operations between two union operations are ordered such that there are first the edge insertion operations and after that the relabeling operations. Edges are inserted and vertices are relabeled as soon as possible in the following sense. By Definition 4.1(2.(a)),

every edge insertion operation $\eta_{i'_{l'},j'_{l'}}$ inserts at least one edge between a vertex of val(Y_1) and a vertex of val(Y_2) but no edge between two vertices of val(Y_1) or between two vertices of val(Y_2). By Definition 4.1(2.(b)), a relabeling operation $\rho_{i_l \to j_l}$ always relabels at least one vertex to some label already used by at least one other vertex. Thus every relabeling operation decreases the number of labels used by the vertices of the graph. Since none of the relabeling operations $\rho_{i_{l-1} \to j_{l-1}}, \ldots, \rho_{i_1 \to j_1}$ relabels anything to i_l, every vertex is relabeled at most once (between two union operations). The three properties, there is a vertex labeled by i_l, there is a vertex labeled by j_l, and $i_l \notin \{j_{l-1}, \ldots, j_1\}$ imply that $i_l \notin \{i_{l-1}, \ldots, i_1, j_{l-1}, \ldots, j_1\}$ and $j_l \notin \{i_{l-1}, \ldots, i_1\}$. Finally, by Definition 4.1(2.(c)), the number of labels used in the graphs defined by the subexpressions is always minimal.

The following observations are easy to verify. If k-expression $\rho_{i \to j}(Y)$ is in normal form then k-expression Y is in normal form, if k-expression $\eta_{i',j'}(Y)$ is in normal form then k-expression Y is in normal form, and if k-expression $Y_1 \oplus Y_2$ is in normal form then k-expression Y_1 and k-expression Y_2 are in normal form. That is, if an expression is in normal form, then every complete subexpression is in normal form.

Theorem 4.2 *For every k-expression X there is an equivalent k-expression in normal form.*

Proof: We show how to transform an arbitrary k-expression X into an equivalent k-expression in normal form.

The following transformation steps can be used to transform a k-expression X into an equivalent k-expression in that no edge insertion operation is applied directly after a relabeling operation.

Let $Z = \eta_{i',j'}(\rho_{i \to j}(Y))$ be a subexpression of X.

1. If $\{i, j\} \cap \{i', j'\} = \emptyset$, then Z can be replaced by $\rho_{i \to j}(\eta_{i',j'}(Y))$, because the two operations do not affect each other.

2. If $i \in \{i', j'\}$, then we can omit the edge insertion operation $\eta_{i',j'}$, because it does not create an edge.

3. If $i \notin \{i', j'\}$ and $j \in \{i', j'\}$, then we distinguish between two cases. If $j = i'$ then Z can be replaced by $\rho_{i \to j}(\eta_{i',j'}(\eta_{i,j'}(Y)))$, if $j = j'$ then Z can be replaced by $\rho_{i \to j}(\eta_{i',j'}(\eta_{i',i}(Y)))$.

These transformation steps can be used to transform a k-expression X into an equivalent k-expression such that all edge insertion and relabeling operations are in the right order with respect to Definition 4.1. The succeeding transformation steps will not change this *right* order.

Next we consider an induction on the number of union operations and the composition of X. The transformation steps do not change the number of union operations in the modified subexpressions.

Let

$$X = \rho_{i_n \to j_n}(\cdots \rho_{i_1 \to j_1}(\eta_{i'_{n'}, j'_{n'}}(\cdots \eta_{i'_1, j'_1}(\bullet_t)\cdots))\cdots)$$

be a k-expression without any union operation. Then X is equivalent to a k-expression $\bullet_j$ for some $j \in \{j_1, \ldots, j_n, t\}$.

Let

$$X = \eta_{i'_{l'}, j'_{l'}}(\eta_{i'_{l'-1}, j'_{l'-1}}(\cdots \eta_{i'_1, j'_1}(Y_1 \oplus Y_2)\cdots))$$

be a k-expression, where

$$X = \eta_{i'_{l'-1}, j'_{l'-1}}(\cdots \eta_{i'_1, j'_1}(Y_1 \oplus Y_2)\cdots)$$

is in normal form.

1. If $\mathrm{val}(Y_1)$ does not contain all edges between vertices labeled by $i'_{l'}$ and vertices labeled by $j'_{l'}$, then we transform the k-expression $\eta_{i'_{l'}, j'_{l'}}(Y_1)$ into an equivalent k-expression Y'_1 in normal form and replace in X the subexpression Y_1 by Y'_1. The transformation of $\eta_{i'_{l'}, j'_{l'}}(Y_1)$ into normal form is possible by the inductive hypothesis. The same replacement is possible for Y_2, if necessary.

2. If

   $$\eta_{i'_{l'}, j'_{l'}}(\eta_{i'_{l'-1}, j'_{l'-1}}(\cdots \eta_{i'_1, j'_1}(Y_1 \oplus Y_2)\cdots)) \equiv \eta_{i'_{l'-1}, j'_{l'-1}}(\cdots \eta_{i'_1, j'_1}(Y_1 \oplus Y_2)\cdots)$$

   then we omit operation $\eta_{i'_{l'}, j'_{l'}}$ from X, because it does not create any edge.

The result is an equivalent k-expression in normal form.

Let

$$X = \rho_{i_n \to j_n}(\cdots \rho_{i_1 \to j_1}(\eta_{i'_{n'}, j'_{n'}}(\cdots \eta_{i'_1, j'_1}(Y_1 \oplus Y_2)\cdots))\cdots)$$

be a k-expression, where subexpression

$$Z_0 := \eta_{i'_{n'}, j'_{n'}}(\cdots \eta_{i'_1, j'_1}(Y_1 \oplus Y_2)\cdots)$$

is in normal form.

If graph $\mathrm{val}(Y_1)$ has a vertex labeled by i and a vertex labeled by j for two distinct labels $i, j \in [k]$ such that

$$\begin{array}{rl} & \rho_{i_l \to j_l}(\cdots \rho_{i_1 \to j_1}(\eta_{i'_{n'}, j'_{n'}}(\cdots \eta_{i'_1, j'_1}(\quad Y_1 \quad \oplus \quad Y_2 \quad)\cdots))\cdots) \\ \equiv & \rho_{i_l \to j_l}(\cdots \rho_{i_1 \to j_1}(\eta_{i'_{n'}, j'_{n'}}(\cdots \eta_{i'_1, j'_1}(\quad \rho_{i \to j}(Y_1) \quad \oplus \quad Y_2 \quad)\cdots))\cdots), \end{array}$$

then we transform the k-expression $\rho_{i \to j}(Y_1)$ into an equivalent k-expressions Y'_1 in normal form and replace in k-expression X the subexpression Y_1 by Y'_1. The transformation of $\rho_{i_l \to j_l}(Y_1)$ into normal form is possible by the inductive

hypothesis. After that we transform Z_0 with the new subexpression Y_1 into normal form, if necessary. This is possible by the inductive hypothesis and the transformation steps already defined above. The same replacement is possible for Y_2, if necessary. This procedure can be repeated at most $k-1$ times for Y_1 and Y_2, because after every replacement the number of labels used by the vertices of $\mathrm{val}(Y_1)$ and $\mathrm{val}(Y_2)$ decreases by one.

We now compute for every label $l \in [k]$, the label $h(l)$ into which label l is relabeled by performing the n relabeling operations $\rho_{i_1 \to j_1}, \ldots, \rho_{i_n \to j_n}$ in this given order one after the other. Function $h : [k] \to [k]$ can be considered as a directed graph $H = (V_H, A_H)$ with vertex set $V_H = [k]$ and arc set $A_H = \{(l, h(l)) \mid l \in [k]\}$. Every vertex l of H has exactly one outgoing arc $(l, h(l))$.

We first remove all arcs (l_1, l_2) from H for which graph $\mathrm{val}(Z_0)$ has no vertex labeled by l_1 and all arcs (l_1, l_2) for which $l_1 = l_2$, because these arcs do not represent any relabeling of vertices of $\mathrm{val}(Z_0)$. Next we consider every pair of two arcs (l_1, l_2), (l_2, l_3) of H and simultaneously replace in expression Z_0 all labels l_1 by l_2 and all labels l_2 by l_1, and remove both arcs (l_1, l_2) and (l_2, l_3) from H. After that we insert a new arc (l_1, l_3) into H if $l_1 \neq l_3$. Note that k-expression Z_0 remains in normal form if two labels are exchanged in all operations of Z_0. Finally, we consider all arcs (l_1, l_2) of H for which graph $\mathrm{val}(Z_0)$ has no vertex labeled by l_2. We then simultaneously replace in expression Z_0 all labels l_1 by l_2 and all labels l_2 by l_1, and remove arc (l_1, l_2) from H.

Now we can define the new relabeling by the remaining arcs of H. We remove step by step an arc (l_1, l_2) from H and apply the relabeling operation $\rho_{l_1 \to l_2}$ to the current k-expression Z_i (which is initially Z_0) to get a new k-expression $Z_{i+1} = \rho_{l_1 \to l_2}(Z_i)$. This leads to a k-expression which is in normal form and equivalent to the original one. □

The proof of Theorem 4.2 uses a simple relabeling trick to omit a relabeling operation $\rho_{i_l \to j_l}(X)$ if graph $\mathrm{val}(X)$ has no vertex labeled by j_l. This relabeling simultaneously replaces in expression X all labels i_l by j_l and all labels j_l by i_l. Let $X_{i_l \leftrightarrow j_l}$ be the resulting expression. If X is in normal form then $X_{i_l \leftrightarrow j_l}$ is in normal form, and $X_{i_l \leftrightarrow j_l} \equiv \rho_{i_l \to j_l}(X)$.

5 Replaceability

Most of the bottom-up dynamic programming algorithms for deciding a graph property Π on a tree-structured graph G are based on the idea of substituting a subgraph of G by a small so-called *replaceable* subgraph. The substitution is defined by a composition mechanism which is different for the various graph models. However, the notion of *replaceability* can be defined for every composition mechanism. Thus, the bottom-up dynamic programming techniques work in principle for all tree-structured graphs, more or less successfully.

For the analysis of *tree-width bounded graphs*, we need so-called *l-terminal graphs* and an operation denoted by $\circ$ which combines two l-terminal graphs by identifying vertices. Since we are mainly interested in labeled graphs, we use

a labeled version of l-terminal graphs. Terminal graphs are also called *sourced graphs*, see [ALS91].

Definition 5.1 (k-labeled l-terminal graph) *A* k-labeled l-terminal graph *is a system*

$$G = (V_G, E_G, P_G, lab_G)$$

where (V_G, E_G, lab_G) *is a* k*-labeled graph and* $P_G = (x_1, \ldots, x_l)$ *is a sequence of* $l \geq 0$ *distinct vertices of* V_G*. The vertices in sequence* P_G *are called* terminal vertices *or* terminals *for short. Vertex* x_i*,* $1 \leq i \leq l$*, is the* i*-th* terminal *of* G*. The other vertices in* $V_G - P_G$ *are called the* inner vertices *of* G*.*

Let $H = (V_H, E_H, P_H, lab_H)$ *and* $J = (V_J, E_J, P_J, lab_J)$ *be two vertex disjoint* k*-labeled* l*-terminal graphs such that the* i*-th terminal of* P_H *has the same label as the* i*-th terminal of* P_J *for* $i = 1, \ldots, l$*. Then the composition* $H \circ J$ *is the* k*-labeled graph obtained by taking the disjoint union of* (V_H, E_H, lab_H) *and* (V_J, E_J, lab_J)*, and then identifying corresponding terminals, i.e., for* $i = 1, \ldots, l$*, identifying the* i*-th terminal of* H *with the* i*-th terminal of* J*, and removing multiple edges.*

Definition 5.2 (Replaceability of k-labeled l-terminal graphs) *Let* Π *be a graph property, i.e.,* $\Pi : \mathcal{G}_k \to \{true, false\}$*, where* $\mathcal{G}_k$ *is the set of all* k*-labeled graphs. Two* k*-labeled* l*-terminal graphs* H_1 *and* H_2 *are called* replaceable *with respect to* Π*, denoted by* $H_1 \sim_{\Pi,l} H_2$*, if for every* k*-labeled* l*-terminal graph* J*,*

$$\Pi(H_1 \circ J) \;=\; \Pi(H_2 \circ J).$$

Figure 3 and 4 show three 4-labeled 3-terminal graphs H_1, H_2, J. The two 4-labeled 3-terminal graphs H_1 and H_2 are not replaceable, for example, with respect to Hamiltonicity, because $H_1 \circ J$ has a Hamilton cycle but $H_2 \circ J$ does not.

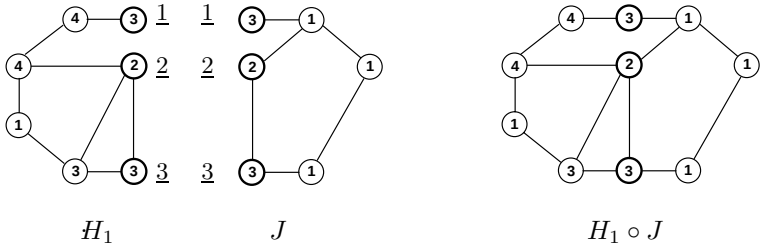


Figure 3: Two 4-labeled 3-terminal graphs H_1,J and the composed graph $H_1 \circ J$. The underlined integers represent the numbering of the terminals.

It is well known that if $\sim_{\Pi,l}$ divides the set of all k-labeled l-terminal graphs into a finite number of equivalence classes then Π is decidable in linear time for all k-labeled graphs of tree-width at most l. Note that linear time means under the assumption that integer l is fixed and not part of the input.

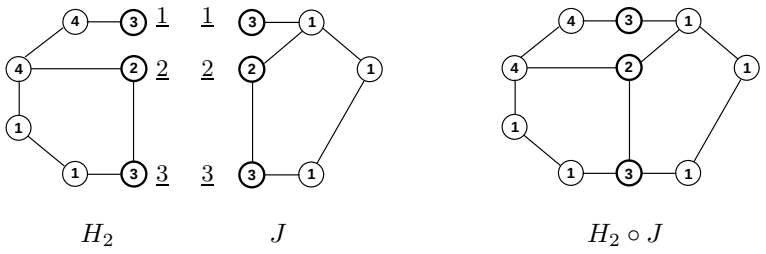


Figure 4: Two 4-labeled 3-terminal graphs H_2,J and the composed graph $H_2 \circ J$

The same schema can be used to solve graph properties on graphs of bounded clique-width. In this case the composition of two vertex disjoint k-labeled graphs H and J is done by an operation $\times_S$, where $S \subseteq [k] \times [k]$. The composed k-labeled graph $H \times_S J$ is the disjoint union of H and J with all additional edges between vertices $u \in V_H$ and $v \in V_J$ for which $(\mathrm{lab}_H(u), \mathrm{lab}_J(v)) \in S$. Two k-labeled graphs H_1 and H_2 are replaceable with respect to some graph property Π, denoted by $\sim_{\Pi,k}$, if for all k-labeled graphs J and all $S \subseteq [k] \times [k]$,

$$\Pi(H_1 \times_S J) \;=\; \Pi(H_2 \times_S J).$$

If this equivalence relation $\sim_{\Pi,k}$ has a finite number of equivalence classes then property Π is decidable in linear time for all k-labeled graphs $\mathrm{val}(X)$ of clique-width at most k if the k-expression X is given to the input (integer k is assumed to be fixed and not part of the input).

For all who are interested in the details how to solve a graph property on a tree-width or clique-width bounded graph with the bottom-up techniques mentioned above, we refer to [Arn85, AP89, ALS91, Bod97, Bod98, CMR00, Cou90, EGW01, KR01, LW93, Wan94]. These details are not necessary for this paper.

6 Overview

In this section, we intuitively explain how the proof of our main result is running. Let Π_k be the graph property clique-width at most k. Let $\sim_{\Pi_k,l}$ be the equivalence relation defined for k-labeled l-terminal graphs as in Definition 5.2. Our aim is to show that $\sim_{\Pi_k,l}$ divides the set of all k-labeled l-terminal graphs into a finite number of equivalence classes, for every fixed $k \geq 1$ and every fixed $l \geq 0$. This would imply that the graph property clique-width at most k is decidable in linear time for graphs of tree-width at most l.

For every k-labeled l-terminal graph G we will define a so-called *connection type* consisting of a set of so-called *connection trees*. We will show that two k-labeled l-terminal graphs are replaceable with respect to the graph property clique-width at most k if they are of the same connection type, but not necessarily vice versa. Thus, the number of equivalence classes of $\sim_{\Pi_k,l}$ can be

bounded by the number of mutually different connection types for all k-labeled l-terminal graphs. If there is a finite number of mutually different connection types for every fixed $k \geq 1$ and every fixed $l \geq 0$, then $\sim_{\Pi_k,l}$ has a finite number of equivalence classes, and our main result follows.

The outline of the proof can easily be explained more precisely, but still intuitively, with a simplified version of the connection tree. To distinguish between the real connection tree and the simplified one, we will call the simplified version the *strong connection tree*. Let H and J be two k-labeled l-terminal graphs such that the k-labeled graph $H \circ J$ has clique-width at most k, see also Figure 5. Let X be a k-expression for $H \circ J$ and let T be the k-expression tree of X. The k-expression tree T can be decomposed into two subtrees, say T_H and T_J, as follows. Subtree T_H describes the k-labeled subgraph of $H \circ J$ induced by the vertices of H. That is, T_H consists of the leaves of T representing the vertices of H and of all nodes of T on the paths from these leaves to the root of T. Subtree T_J is defined in the same way with respect to the vertices of J. Note that T_H and T_J are not necessarily expression tress. Every node of T is in at least one of these two subtrees T_H and T_J. Some of the nodes of T are contained in both subtrees. More precisely, the root of T is in both subtrees and at least the leaves of T representing the identified terminals of H and J, and all nodes of T on the paths of these leaves to the root of T are in both subtrees.

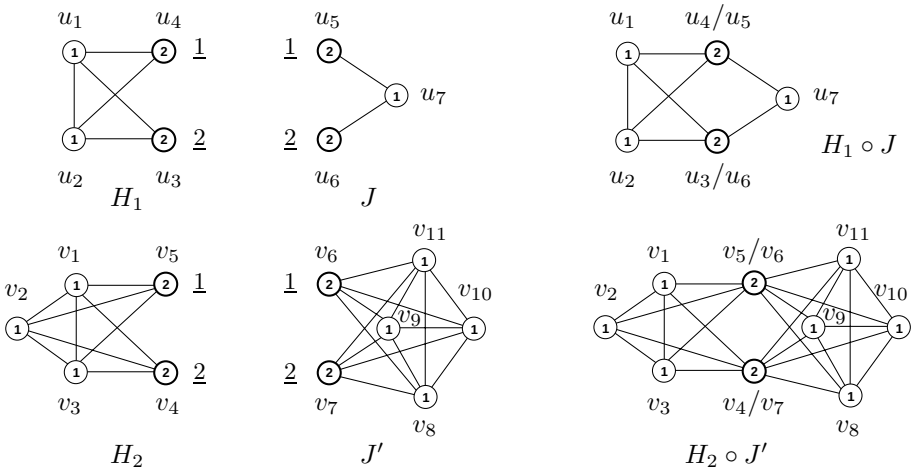


Figure 5: Four 2-labeled 2-terminal graphs H_1, H_2, J, and J', and the two 2-labeled graphs $H_1 \circ J$ and $H_2 \circ J'$

The common part of both subtrees T_H and T_J, denoted by C, defines a *strong connection tree* for H. The leaves in the common part C are either leaves of T or union nodes. If a leaf u represents a vertex of $H \circ J$ obtained by identifying the i-th terminal of H with the i-th terminal of J, then u will additionally be labeled by index i. Let u be a union node of the k-expression

tree T and let u_l and u_r be the left and right child of u in T. If u is in the common part C but u_l or u_r is not, then we add a left child v_l to C or a right child v_r to C, respectively, such that we get an ordered tree in that every union node has a left child and a right child. If u_l (u_r) is a node of T_H but not a node of T_J, then the inserted leaf v_l (v_r, respectively) is labeled by the set L of all labels of the vertices in the k-labeled graph defined by the k-expression subtree $T(u_l)$ (k-expression subtree $T(u_r)$, respectively) of T. Figure 6 illustrates such a labeling of the inserted leaves by an example. The existence of the non-empty labeling L indicates that the leaf represents a subtree of T_H and not a subtree of T_J. These leaves are called *internal* leaves, the other leaves are called *external* leaves. The notions *internal* and *external* refer to the association that the left argument H is the internal graph for which we compute the connection tree, and the right argument J is the external graph, i.e., the environment to which H is attached. The resulting structure C is called a *strong connection tree* for H. To get all strong connection trees for H, we have to consider all k-labeled l-terminal graphs J such that $H \circ J$ has clique-width at most k, and all possible k-expressions for $H \circ J$. The set of all strong connection trees for H is the *strong connection type* of H.

Let us next explain why two k-labeled l-terminal graphs H_1 and H_2 of the same strong connection type are replaceable with respect to clique-width at most k. After that we consider the size of the strong connection tees. Assume $H_1 \circ J$ has clique-width at most k for some k-labeled l-terminal graph J. Let X be a k-expression for $H_1 \circ J$. Let T be the k-expression tree of X and let T_{H_1} and T_J be the two subtrees for H_1 and J, respectively. The common part of T_{H_1} and T_J defines a strong connection tree C for H_1 which is, by our assumption, also a strong connection tree for H_2. That is, there has to be at least one k-labeled l-terminal graph J' such that $H_2 \circ J'$ has clique-width at most k. Furthermore, there has to be a k-expression X' with a k-expression tree T' for $H_2 \circ J'$ such that the common part of the two subtrees T'_{H_2} and $T'_{J'}$ defines the same strong connection tree C for H_2. Now we can replace in k-expression tree T subtree T_{H_1} by subtree T'_{H_2}, see Figure 7. This can easily be done by substituting the corresponding subtrees represented by the internal leaves. Let T'' be the resulting k-expression tree we get after this replacement.

It is easy to verify that the k-expression tree T'' defines the k-labeled graph $H_2 \circ J$. The vertices from H_2 and J are labeled in the k-labeled graph defined by T'' as in the k-labeled graphs $H_2 \circ J'$ and $H_1 \circ J$, respectively. Two vertices from H_2 or two vertices from J are connected by an edge if and only if they are connected by an edge in $H_2 \circ J'$ or $H_1 \circ J$, respectively. This is, because the subtrees defined by the paths from the involved leaves to the roots are equal in both k-expression trees T'' and T or in both k-expression trees T'' and T', respectively.

The additional L-labeling of the internal leaves in the strong connection tree C is necessary to ensure that T'' defines no forbidden edge between an inner vertex u_1 of H_2 and an inner vertex u_2 of J. If the graph defined by T'' has such a forbidden edge then $H_1 \circ J$ would also have at least one such forbidden edge, because the corresponding subgraph of H_1 would have at least one vertex

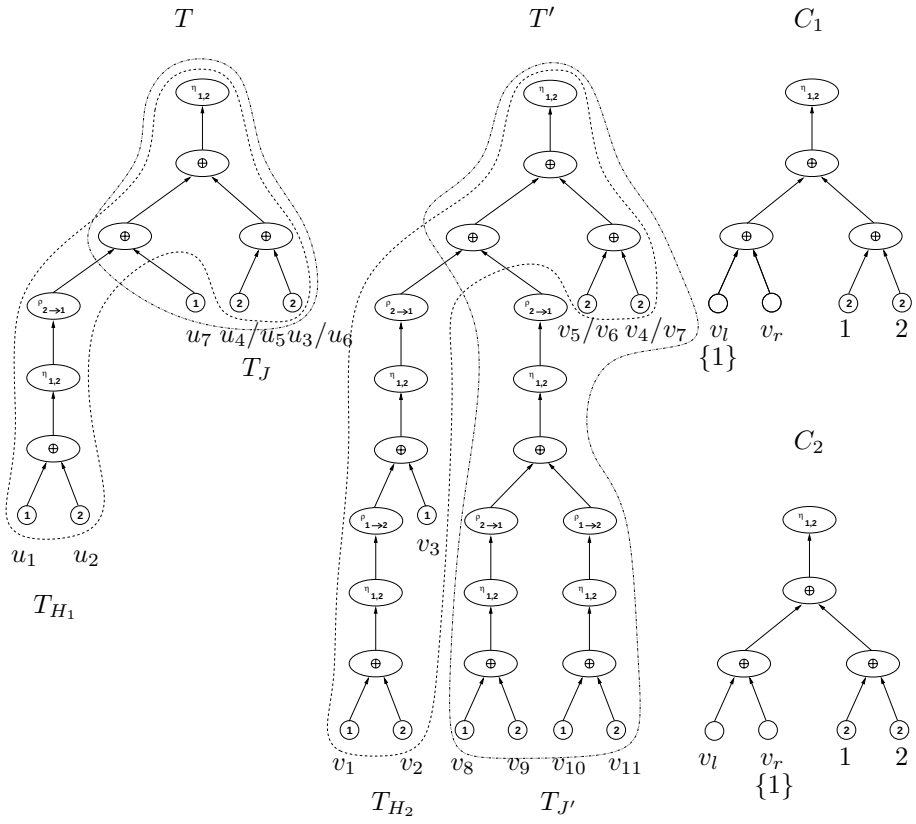


Figure 6: A 2-expression tree T for the 2-labeled graph $H_1 \circ J$ and a 2-expression tree T' for the 2-labeled graph $H_2 \circ J'$. C_1 is a strong connection tree for H_1 and H_2. C_2 is a strong connection tree for J and J'.

labeled as u_1 of H_2. If for every k-labeled l-terminal graph J, graph $H_1 \circ J$ has clique-width at most k if and only if $H_2 \circ J$ has clique-width at most k, then obviously H_1 and H_2 are replaceable with respect to clique-width at most k.

Finally, we have to consider the size of the strong connection trees. The size of the common part of the two subtrees T_H and T_J can, unfortunately, not be bounded by some constant depending only on k and l. However, the main part of the next section is the proof that for every k-labeled graph $H \circ J$ of clique-width at most k there is at least one k-expression tree T such that the information we really need from the common part of the two subtrees T_H and T_J can be bounded. This information is still tree-structured and will be defined in the next section as the real connection tree.

We will show step by step that there is a k-expression tree for $H \circ J$ in that the paths in the common part of T_H and T_J have the following structure. We divide the operations of the nodes of T into H-operations and J-operations. An

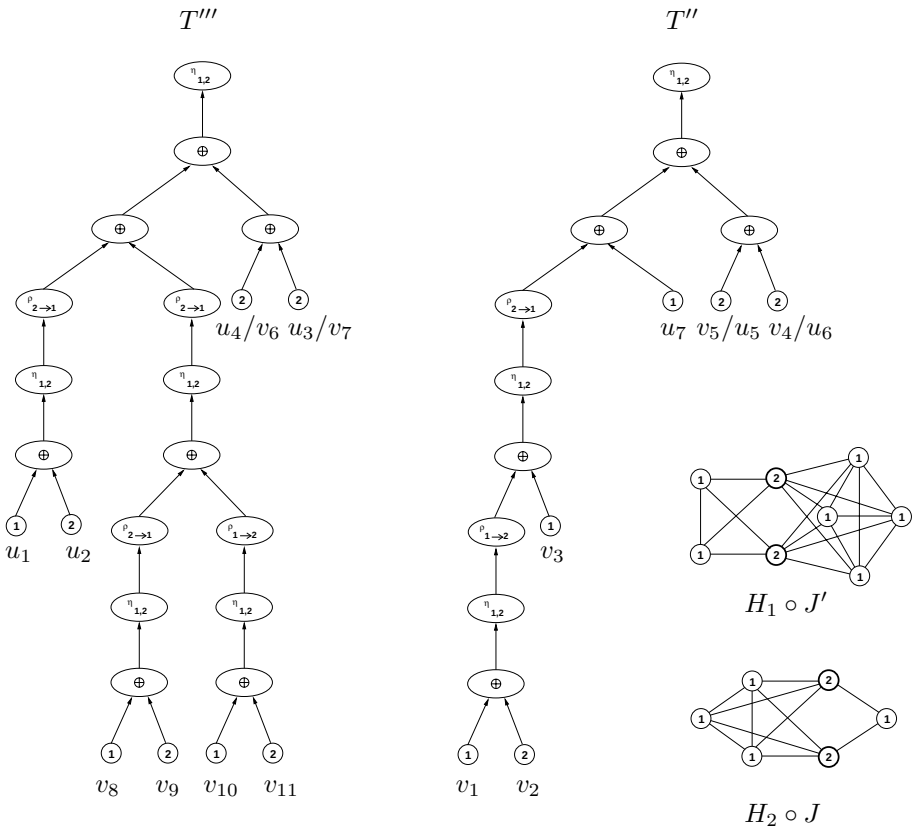


Figure 7: A 2-expression tree T''' for the 2-labeled graph $H_1 \circ J'$ and a 2-expression tree T'' for the 2-labeled graph $H_2 \circ J$.

H-operation changes a label of a vertex from H or inserts an edge incident to a vertex from H. A J-operation does anything concerning the vertices from J. Some of the operations could even be H-operations and an J-operations. In the next section, we will prove that there is always a k-expression tree T for $H \circ J$ such that in the common part of T_H and T_J the number of times the classification into H- and J-operations changes along a path from a leaf to the root can be bounded by some constant depending only on k and l. This property finally allows us to define a connection structure of bounded size, which we call the *connection tree* for H. The main idea is to replace the unbounded subpaths with certain operations of the same type by single so-called *bridge* nodes.

7 Determining the connection type

We consider the case where we have a k-labeled l-terminal graph

$$H = (V_H, E_H, P_H, \mathrm{lab}_H)$$

and a k-labeled l-terminal graph $J = (V_J, E_J, P_J, \mathrm{lab}_J)$ such that H and J are vertex disjoint and the combined graph

$$G = (V_G, E_G, \mathrm{lab}_G) = H \circ J$$

has clique-width at most k.

We partition the vertex set V_G of G into three disjoint sets U_H, U_J, U_P such that $U_H \cup U_J \cup U_P = V_G$. Vertex set $U_H = V_H - P_H$ contains the inner vertices from H, vertex set $U_J = V_J - P_J$ contains the inner vertices from J, and vertex set U_P contains the joined terminals from H and J. Vertex set U_P has exactly l vertices, because the l terminals of H are identified with the l terminals of J. Note that graph G does not have any edge between a vertex of U_H and a vertex of U_J.

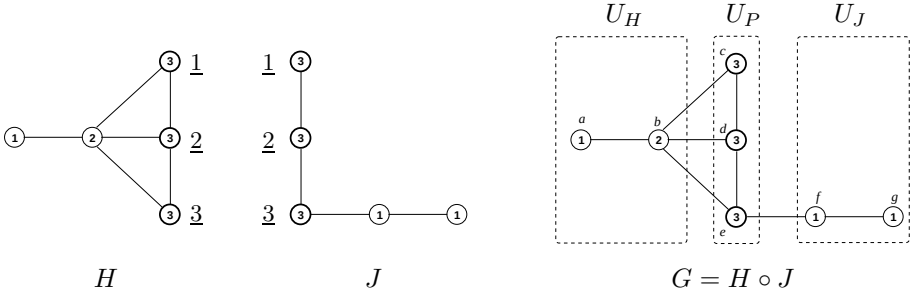


Figure 8: Two 3-labeled 3-terminal graphs H and J, the 3-labeled graph $G = H \circ J$, and the partition of its vertices into U_H, U_P, and U_J

Let T be a k-expression tree for $G = H \circ J$. The subtree T_P of T is defined by the l leaves of T that correspond to the l vertices of U_P and by all nodes of T on the paths from these leaves to the root of T, see Figure 9. Thus the root of T_P is the root of T. Tree T_P is in general not an expression tree. It is only an expression tree if neither H nor J has inner vertices. In this case, T_P and T are equal.

Our intention is to show that for each such pair H, J as above there is always at least one k-expression tree T for G such that T_P has a very special form. This special form represents the necessary information how H and J are combined. We will see that the size of this connection information will depend only on k and l but not on the size of H or J.

The following four subsections start with a lemma that allows us to consider a more restricted k-expression tree T than before. The restrictions are expressed

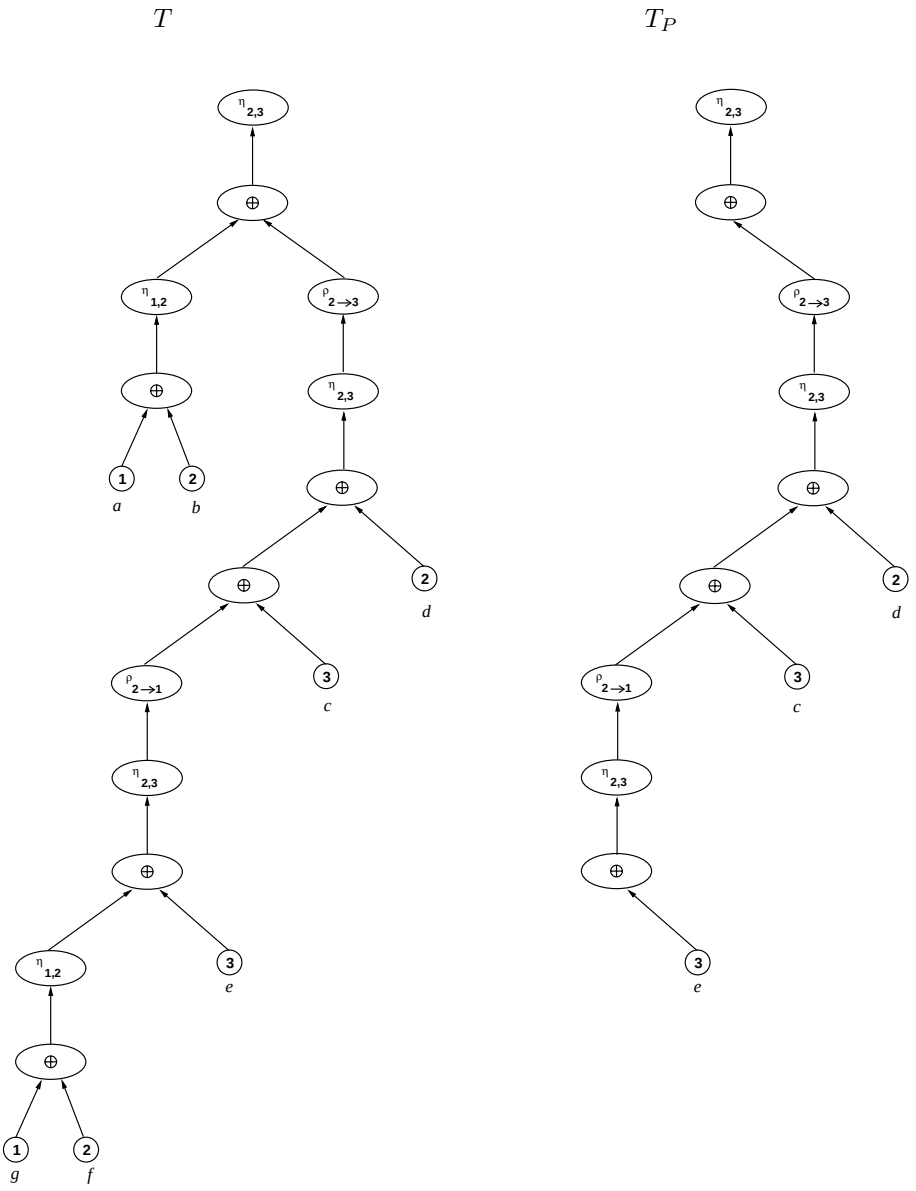


Figure 9: A 3-expression tree T for the 3-labeled graph G of Figure 8 and the subtree T_P of T

by certain properties that have to be satisfied. The lemmas show that this is always possible without loss of generality.

Partition into paths of type 1, 1.*a*, and 1.*b*

Lemma 7.1 *There is always a k-expression tree T for G that satisfies the following property.*

Property 7.2 *Let u_1 be a union node of T such that one of its children u_0 is in T_P and the other child u_0' is not in T_P. Then the vertices of $G(u_0')$ are either all from U_H or all from U_J.*

Proof: Since $G(u_0')$ does not contain vertices from U_P, we know that the vertices of $G(u_0')$ are all from $U_H \cup U_J$. If the vertices of $G(u_0')$ are not all from U_H or not all from U_J then let T_H and T_J be the two k-expression trees that define the subgraphs of $G(u_0')$ induced by the vertices of U_H and U_J, respectively. T_H and T_J can easily be constructed from $T(u_0')$ by removing subtrees whose leaves represent only vertices from U_J or U_H, respectively. A union node that loses one of its children can be omitted by making the remaining child to the child of its parent node. Then we replace subtree $T(u_0')$ by T_H and T_J as follows. We insert a new union node v_0 between u_1 and u_0, and make the roots of T_H and T_J to the second child of u_1 and v_0, respectively. The expression of the resulting tree obviously defines the same graph as before but u_1 now satisfies Property 7.2. This can be done for all union nodes which do not satisfy Property 7.2. See also Figure 10. □

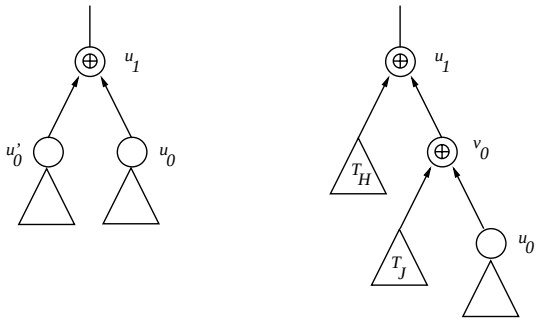


Figure 10: A transformation step used in the proof of Lemma 7.1

Let X be the k-expression of k-expression tree T which satisfies Property 7.2. Then we can apply the transformation steps of the proof of Theorem 4.2 to get a k-expression in normal form equivalent to X. This is possible because the transformation steps of the transformation into normal form only rearrange some relabeling and edge insertion operations. They do not change Property 7.2 of T. ¿From now on we will assume that T satisfies Property 7.2 and that X is in normal form.

Let u_1 be a union node of T such that one of its children u_0 is in T_P and the other child u_0' is not in T_P. We define $\xi(u_1) := 0$ or $\xi(u_1) := 1$ if the vertices of $G(u_0')$ are all from U_H or all from U_J, respectively. In all other cases and in

the case where u_1 is not a union node, we say $\xi(u_1)$ is undefined. For better readability we write $\xi(u_1) = H$ instead of $\xi(u_1) = 0$ and $\xi(u_1) = J$ instead of $\xi(u_1) = 1$. This does not mean that $\xi(u_1)$ is the graph H or J, but only that all vertices of $G(u'_0)$ are from U_H or H_J, respectively. By Lemma 7.1, we can now assume that $\xi(u_1)$ is well defined for all union nodes u_1 of T for which exactly one of their children is not in T_P.

The tree T_P with l leaves now consists of at most $2l - 1$ maximal paths $p = (u_1, \ldots, u_{s'})$, $s' \geq 1$, such that u_1 is a union node with two children in T_P or u_1 has only one child in T_P which is a leaf. The last node $u_{s'}$ of such a path p is either the root of T_P or a child of some union node whose children are both in T_P. All the graphs $G(u_s)$ for $s = 1, \ldots, s'$ contain the same vertices of U_P. Such a path of T_P is called a 1-*path* or *path of type* 1. Every non-leaf node of T_P is in exactly one of these paths of type 1.

A maximal subpath $(u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of T_P such that u_1 is a union node, $u_2, \ldots, u_{r'}$ are edge insertion nodes, and $u_{r'+1}, \ldots, u_{s'}$ are relabeling nodes, is called a *frame* of T_P. Every frame has at most $\binom{k}{2} + k$ nodes, because there is exactly one union node u_1, there are at most $\binom{k}{2}$ edge insertion nodes $u_2, \ldots, u_{r'}$, and at most $k - 1$ relabeling nodes $u_{r'+1}, \ldots, u_{s'}$. Figure 11 shows the general structure of a frame.

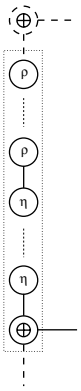


Figure 11: A frame always starts with a union node followed by edge insertion nodes and relabeling nodes.

The first frame of every 1-path is called a *path of type* 1.*a*. The remaining part, if not empty, is called a *path of type* 1.*b*. There are at most $2l - 1$ paths of type 1.*a* and at most $2l - 1$ paths of type 1.*b*. Every 1.*a* path has at most $\binom{k}{2} + k$ nodes, because it is a frame. For every union node u_1 of a 1.*b*-path there is either $\xi(u_1) = H$ or $\xi(u_1) = J$.

Partition into paths of type 2.*a* and 2.*b*

For some node u_s of the k-expression tree T, let $L_H(u_s)$, $L_J(u_s)$, and $L_P(u_s)$ be the label sets of the vertices of $G(u_s)$ which are from U_H, U_J, and U_P,

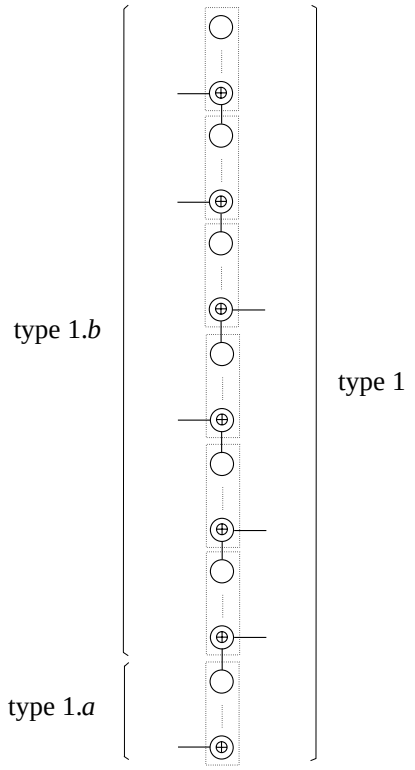


Figure 12: Every 1-path p is divided into a path of type 1.a and a path of type 1.b. The path of type 1.a is the first frame of p. The path of type 1.b is the remaining part of p, which can also be empty.

respectively. The intersection sets

$$L_H(u_s) \cap L_J(u_s), \quad L_P(u_s) \cap L_H(u_s), \quad L_P(u_s) \cap L_J(u_s),$$

and

$$L_P(u_s) \cap L_H(u_s) \cap L_J(u_s)$$

are abbreviated by

$$L_{H\cap J}(u_s), \quad L_{P\cap H}(u_s), \quad L_{P\cap J}(u_s)$$

and

$$L_{P\cap H\cap J}(u_s),$$

respectively.

Lemma 7.3 *There is always a k-expression tree T for G such that the k-expression X of T is in normal form and T satisfies Property 7.2 and additionally Property 7.4.*

Property 7.4 *Let u_s be a relabeling node of T_P labeled by $\rho_{i\to j}$ and let u_{s-1} be the child of u_s in T_P. If $i \in L_P(u_{s-1})$ then $j \in L_P(u_{s-1})$.*

Proof: By induction on the height of T_P. Assume X is in normal form and T_P satisfies Property 7.2. Let $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of T_P and u_s, $r' < s \leq s'$, be a relabeling node labeled by $\rho_{i\to j}$. Let $i \in L_P(u_{s-1})$. By the inductive hypothesis, we assume that $T(u_{s-1})$ already satisfies Property 7.4.

If $j \notin L_P(u_{s-1})$ then we simultaneously replace in the expression of subtree $T(u_{r'})$ every label i by label j and every label j by label i. The expression of the resulting subtree $T(u_{r'})$ is still in normal form and $T(u_{r'})$ satisfies Property 7.2 and 7.4. Since i is not involved in the relabeling operations of the nodes $u_{r'+1}, \ldots, u_{s-1}$, the resulting expression X is obviously in normal form and defines the same graph as before, and $T(u_s)$ satisfies Property 7.2 and Property 7.4. □

Let u_{s-1} be the child of some relabeling node u_s of T_P. By Lemma 7.3, we can now assume that

$$L_P(u_{s-1}) \quad \supseteq \quad L_P(u_s).$$

If $L_P(u_{s-1}) = L_P(u_s)$, then the reverse inclusion holds true for the sets $L_{P\cap H}(u_s)$ and $L_{P\cap J}(u_s)$, i.e.,

$$L_{P\cap H}(u_{s-1}) \quad \subseteq \quad L_{P\cap H}(u_s) \quad \text{and} \quad L_{P\cap J}(u_{s-1}) \quad \subseteq \quad L_{P\cap J}(u_s),$$

because a relabeling of a label from $L_{P\cap H}(u_{s-1})$ or $L_{P\cap J}(u_{s-1})$ is always a relabeling of a label from $L_P(u_{s-1})$.

This allows us to divide every 1.*b*-path p into *paths of type* 2.*a* and *paths of type* 2.*b* as follows. The 2.*a*-paths are the frames $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of p for which at least one of the following two properties holds true.

1. There is some relabeling node u_s, $r' < s \leq s'$, such that

   $$L_P(u_{s-1}) \supsetneq L_P(u_s), \quad L_{P\cap H}(u_{s-1}) \subsetneq L_{P\cap H}(u_s),$$

   or

   $$L_{P\cap J}(u_{s-1}) \subsetneq L_{P\cap J}(u_s).$$

2. $$L_{P\cap H}(u_0) \subsetneq L_{P\cap H}(u_1) \quad \text{or} \quad L_{P\cap J}(u_0) \subsetneq L_{P\cap J}(u_1),$$

   where u_0 is the child of union node u_1 which is in T_P.

It is easy to verify that this is equivalent to property

$$L_P(u_0) \supsetneq L_P(u_{s'}), \quad L_{P\cap H}(u_0) \subsetneq L_{P\cap H}(u_{s'}), \quad \text{or} \quad L_{P\cap J}(u_0) \subsetneq L_{P\cap J}(u_{s'}).$$

where u_0 is the child of union node u_1 which is in T_P.

The 2.*a*-paths are the frames q of the 1.*b*-paths for which either the number of labels in L_P decreases or the number of labels in $L_{P\cap H}$ or $L_{P\cap J}$ increases.

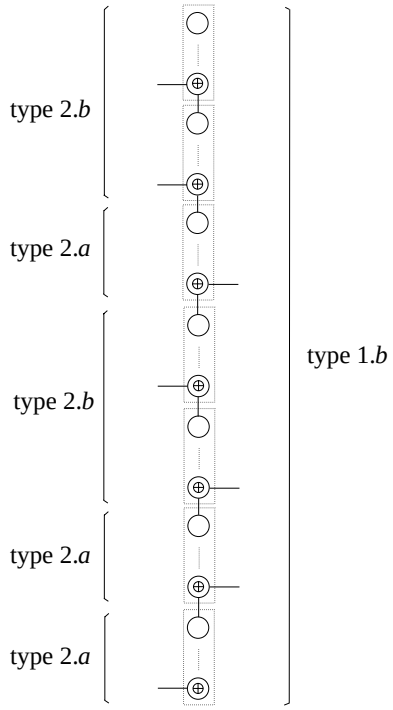


Figure 13: Every 1.b-path is divided into paths of type 2.a and paths of type 2.b. The 2.a-paths are the frames q of the 1.b-paths for which the number of labels in L_P decreases or the number of labels in $L_{P\cap H}$ or $L_{P\cap J}$ increases.

The 2.b-paths are the remaining parts of the 1.b-paths. In a 2.b-path p all the sets $L_P(u_s)$ are equal, all the sets $L_{P\cap H}(u_s)$ are equal, all the sets $L_{P\cap J}(u_s)$ are equal, and thus also all the sets $L_{P\cap H\cap J}(u_s)$ are equal, for all nodes u_s of p including the child u_0 of the first node u_1 which is in T_P. See also Figure 13.

For a frame $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of a 2.b-path let

$$L_P(q) = L_P(u_{s'}), \quad L_{P\cap H}(q) = L_{P\cap H}(u_{s'}),$$

$$L_{P\cap J}(q) = L_{P\cap J}(u_{s'}), \quad \text{and} \quad L_{P\cap H\cap J}(q) = L_{P\cap H\cap J}(u_{s'}).$$

We use q as the argument instead of some node of q to emphasize that the sets above are equal for all nodes of q including the child u_0 of the first node of q which is in T_P. It is easy to count that for every 1.b-path p there are at most $3k-1$ paths of type 2.a and thus at most $3k$ paths of type 2.b. A worst case example for $k = 3$ is shown in the following table. The j-th row shows the labeling for the last node u_i of the j-th 2.a-frame.

j	$L_P(u_i)$	$L_{P\cap H}(u_i)$	$L_{P\cap J}(u_i)$
1	$\{1,\ 2,\ 3\}$	$\{1\}$	$\emptyset$
2	$\{1,\ 2,\ 3\}$	$\{1,\ 2\}$	$\emptyset$
3	$\{1,\ 2,\ 3\}$	$\{1,\ 2,\ 3\}$	$\emptyset$
4	$\{1,\ 2,\ 3\}$	$\{1,\ 2,\ 3\}$	$\{1\}$
5	$\{1,\ 2,\ 3\}$	$\{1,\ 2,\ 3\}$	$\{1,\ 2\}$
6	$\{1,\ 2,\ 3\}$	$\{1,\ 2,\ 3\}$	$\{1,\ 2,\ 3\}$
7	$\{1,\ 2\}$	$\{1,\ 2\}$	$\{1,\ 2\}$
8	$\{1\}$	$\{1\}$	$\{1\}$

Partition into paths of type 3.*a* and 3.*b*

Lemma 7.5 *There is always a k-expression tree T for G such that the k-expression X of T is in normal form and T satisfies Property 7.2, Property 7.4, and additionally Property 7.6.*

Property 7.6 *Let p be a 2.b-path of T_P, and $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of p such that node u_s, $r' < s \leq s'$, is a relabeling node labeled by $\rho_{i\to j}$. If $i \in L_{H\cap J}(u_{s-1})$ then $j \in L_{H\cap J}(u_1)$.*

Before we prove Lemma 7.5 let us emphasize that label j will even be from $L_{H\cap J}(u_1)$ and not only from $L_{H\cap J}(u_{s-1})$.

Proof: Assume X is in normal form, T satisfies Property 7.2 and Property 7.4, and $T(u_{s-1})$ satisfies additionally Property 7.6 for some s, $r' < s \leq s'$. Let $i \in L_{H\cap J}(u_{s-1})$.

If $j \in L_P(q)$ then the assumption $i \in L_{H\cap J}(u_{s-1})$ and the relabeling $\rho_{i\to j}$ at node u_s imply $j \in L_{P\cap H\cap J}(u_s) = L_{P\cap H\cap J}(q)$ and thus $j \in L_{H\cap J}(u_1)$.

If $j \notin L_P(q)$ and $j \notin L_{H\cap J}(u_1)$ then we simultaneously replace in the expression of subtree $T(u_{r'})$ every label i by label j and every label j by label i. The new expression of the resulting subtree $T(u_{r'})$ is still in normal form and subtree $T(u_{r'})$ still satisfies the Properties 7.2, 7.4, and 7.6. Let $\rho_{i_1\to j_1}, \ldots, \rho_{i_{l-1}\to j_{l-1}}$ be the relabeling operations of the nodes $u_{r'+1}, \ldots, u_{s-1}$. Label i is not involved in these relabeling operations, i.e., $i \notin \{i_1, \ldots, i_{l-1}, j_1, \ldots, j_{l-1}\}$. Label j is not relabeled by these relabeling operations, i.e., $j \notin \{i_1, \ldots, i_{l-1}\}$, and none of these relabeling operations relabels some label of $L_{H\cap J}(u_{r'})$ to j in the original expression, because the original tree $T(u_{r'})$ satisfies Property 7.6 and $j \notin L_{H\cap J}(u_1)$. Thus the new expression of the resulting tree $T(u_s)$ is in normal form and defines the same graph as before, and tree $T(u_s)$ now satisfies the Properties 7.2, 7.4, and 7.6. □

For some node u_s of T_P and some label $j \in [k]$ let $\mathrm{forb}_P(u_s, j)$ be the set of all labels $i \in L_P(u_s)$ such that graph $G(u_s)$ has two non adjacent vertices, one labeled by i and one labeled by j. If the set $\mathrm{forb}_P(u_s, j)$ is empty then either graph $G(u_s)$ has no vertex labeled by j or every vertex of $G(u_s)$ labeled by j is adjacent to every vertex of $G(u_s)$ labeled by some label of $L_P(u_s)$. (Remember that $L_P(u_s)$ is always non-empty for the nodes u_s of T_P).

Let $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of T_P such that u_1 is a union node. Let u_0 be the child of u_1 which is in T_P. If one of the edge insertion nodes u_r, $1 < r \leq r'$, is labeled by $\eta_{i,j}$ then $i \notin \text{forb}_P(u_0, j)$ and $j \notin \text{forb}_P(u_0, i)$, because otherwise $\eta_{i,j}$ would create a forbidden edge between two vertices from $G(u_0)$.

Let u_s be a relabeling node of T_P labeled by $\rho_{i \to j}$. If $i \notin L_P(u_{s-1})$ then obviously

$$\text{forb}_P(u_s, j) = \text{forb}_P(u_{s-1}, j) \cup \text{forb}_P(u_{s-1}, i).$$

Intuitively speaking, a vertex labeled by some label of $L_P(u_{s-1})$ is not adjacent in $G(u_s)$ to some vertex labeled by j if and only if it is not adjacent in $G(u_{s-1})$ to some vertex labeled by j or i.

Lemma 7.7 *Assume expression tree T satisfies Property 7.2, Property 7.4, and Property 7.6 and the k-expression X of T is in normal form. Let p be a 2.b-path of T_P, let $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of p, and let u_0 be the child of u_1 which is in T_P. If a node u_s, $r' < s \leq s'$, is a relabeling node labeled by $\rho_{i \to j}$ and if $i \in L_{H \cap J}(u_{s-1})$ then*

$$\mathit{forb}_P(u_0, i) \subsetneq \mathit{forb}_P(u_{s'}, j) \text{ and } \mathit{forb}_P(u_0, j) \subsetneq \mathit{forb}_P(u_{s'}, j).$$

Proof: Since i is not involved in the relabeling operations of the nodes $u_{r'+1}$, $\ldots, u_{s-1}$, label i is also in $L_{H \cap J}(u_1)$. By Property 7.6, we know that $j \in L_{H \cap J}(u_1)$ and thus $i, j \in L_{H \cap J}(u_1)$. Let u_0' be the other child of u_1 which is not in T_P. Without loss of generality, let $\xi(u_1) = H$. Since i and j are both in $L_J(u_1)$ and since the vertices of $G(u_0')$ are all from U_H, graph $G(u_0)$ has at least one vertex labeled by i and at least one vertex labeled by j.

If label i or label j is involved in an edge insertion operation $\eta_{i',j'}$ of the nodes $u_2, \ldots, u_{r'}$ then the other label of $\{i', j'\}$ has to be in $L_P(q) - L_{H \cup J}(u_1)$, i.e., is not in $L_{H \cup J}(u_1)$, where $L_{H \cup J}(u_1)$ is defined by $L_H(u_1) \cup L_J(u_1)$. Otherwise, a forbidden edge between a vertex from U_H and a vertex from U_J is created, because i and j are both in $L_H(u_1)$ and both in $L_J(u_1)$. By our normal form Property 2.(a), we know that all these edge insertion operations do not create a new edge between two vertices from $G(u_0')$ or two vertices from $G(u_0)$. Thus every of these edge insertion operations in that label i or j is involved creates an edge between a vertex from $G(u_0')$ labeled by i or j, respectively, and a vertex from $G(u_0)$ labeled by some label of $L_P(q) - L_{H \cup J}(u_1)$.

If every label of $\text{forb}_P(u_0, i)$ is also in $\text{forb}_P(u_0, j)$ then an additional relabeling $\rho_{i \to j}$ applied to the expression represented by $T(u_0)$ does not change the graph $G(u_{s'})$. This contradicts normal form Property 2.(c). On the other hand, if every label of $\text{forb}_P(u_0, j)$ is also in $\text{forb}_P(u_0, i)$ then an additional relabeling $\rho_{j \to i}$ applied to the expression represented by $T(u_0)$ does not change the graph $G(u_{s'})$. This also contradicts normal form Property 2.(c).

So there has to be at least one label in $\text{forb}_P(u_0, i)$ which is not in $\text{forb}_P(u_0, j)$ and one label in $\text{forb}_P(u_0, j)$ which is not in $\text{forb}_P(u_0, i)$. Since $\text{forb}_P(u_0, i) \subseteq \text{forb}_P(u_{s-1}, i)$ and $\text{forb}_P(u_0, j) \subseteq \text{forb}_P(u_{s-1}, j)$, the result follows. □

Next we divide every 2.*b*-path p into *paths of type* 3.*a* and *paths of type* 3.*b* as follows. The 3.*a*-paths are the frames $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of p for which

$$L_{H\cap J}(u_0) \subsetneq L_{H\cap J}(u_{s'}) \quad \text{or} \quad \text{forb}_P(u_0, j) \subsetneq \text{forb}_P(u_{s'}, j)$$

for some $j \in L_P(u_0) \cup L_{H\cap J}(u_0)$, where u_0 is the child of u_1 in T_P.

The sets above can change their size in a frame of a 2.*b*-path as follows.

1. $L_{H\cap J}(u_0) \subseteq L_{H\cap J}(u_1)$ and $\text{forb}_P(u_0, j) \subseteq \text{forb}_P(u_1, j)$, because a union operation can not remove labels from $L_{H\cap J}(u_0)$ or $\text{forb}_P(u_0, j)$, respectively.

2. $L_{H\cap J}(u_{r-1}) = L_{H\cap J}(u_r)$ and $\text{forb}_P(u_0, j) \subseteq \text{forb}_P(u_r, j)$ for $r = 2, \ldots, r'$, because the edge insertion operations do not change the labels, and do not create edges between two vertices from $G(u_0)$, respectively. Note that they can not remove labels from $\text{forb}_P(u_0, j)$, although they can remove labels from $\text{forb}_P(u_1, j)$.

3. Let u_s, $r' < s \leq s'$, be a relabeling node labeled by $\rho_{j\to j'}$.

   (a) If $j \notin L_P(u_{s-1}) \cup L_{H\cap J}(u_{s-1})$, then $L_{H\cap J}(u_{s-1}) \subseteq L_{H\cap J}(u_s)$.

   (b) If $j \in L_P(u_{s-1}) \cup L_{H\cap J}(u_{s-1})$ then $j \in L_{H\cap J}(u_{s-1})$, because we consider a 2.*b*-path. By Lemma 7.5, $j' \in L_{H\cap J}(u_{s-1})$ and thus $L_{H\cap J}(u_{s-1}) \supsetneq L_{H\cap J}(u_s)$. By Lemma 7.7, $\text{forb}_P(u_s, j) = \emptyset$, $\text{forb}_P(u_0, j) \subsetneq \text{forb}_P(u_s, j')$ and $\text{forb}_P(u_0, j') \subsetneq \text{forb}_P(u_s, j')$.

The size of $\text{forb}_P(u_{s-1}, j)$ for some $j \in L_P(u_{s-1}) \cup L_{H\cap J}(u_{s-1})$ can only become smaller in case 3.(*b*), where $j \in L_{H\cap J}(u_{s-1})$ is relabeled into another label $j' \in L_{H\cap J}(u_{s-1})$. In this case $\text{forb}_P(u_s, j) = \emptyset$, because $G(u_s)$ has no vertex labeled by j.

A simple idea shows that the number of 3.*a*-paths (3.*a*-frames) can be bounded by $(k+1)^{k+1}$. For a node u_s let $\alpha(u_s) = (z_0, \ldots, z_{k'})$ be the vector, where $k' = |L_P(u_s)|$ and z_t, $0 \leq t \leq k'$, is the number of sets $\text{forb}_P(u_s, j)$, $j \in L_P(u_s) \cup L_{H\cap J}(u_s)$, of size t. We say vector $(z'_0, \ldots, z'_{k'})$ is larger than vector $(z_0, \ldots, z_{k'})$, denoted by

$$(z'_0, \ldots, z'_{k'}) \;>\; (z_0, \ldots, z_{k'}),$$

if there is some t, $0 \leq t \leq k'$, such that $z'_t > z_t$ and $z'_{t'} = z_{t'}$ for $t' = t+1, \ldots, k'$. For every 3.*a*-path $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$, we have $\alpha(u_{s'}) > \alpha(u_0)$, where u_0 is the child of u_1 which is in T_P. This bounds the number of 3.*a*-paths by $(k+1)^{k+1}$. Note that this bound is not really tight.

The remaining parts of p are the 3.*b*-paths. In a 3.*b*-path p all the sets $L_{H\cap J}(u_s)$ are equal for all nodes u_s of p including the child of the first node which is in T_P. To emphasizes this we define $L_{H\cap J}(q) = L_{H\cap J}(u_{s'})$ for the frames $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of a 3.*b*-path p. The sets $\text{forb}_P(u_s, j)$ for $j \in L_P(u_0) \cup L_{H\cap J}(u_0)$ do not need to be equal for all nodes u_s of p. In a frame $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of a 3.*b*-path p, it could be that there is some r, $1 \leq r < r'$, such that set $\text{forb}_P(u_r, j)$ has a label which is not in $\text{forb}_P(u_0, j)$. However, we know that for $s = r', \ldots, s'$, $\text{forb}_P(u_0, j) = \text{forb}_P(u_s, j)$.

Partition into paths of type 4

To partition the paths of type 3.*b* into paths of type 4, we need three more lemmas. The first lemma already holds for paths of type 1.*b*, but we use it only for paths of type 3.*b*.

Lemma 7.8 *Let $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of a 1.b-path p such that u_r, $1 < r \leq r'$, is an edge insertion node labeled by $\eta_{i',j'}$. Let u_0 and u'_0 be the two children of u_1, where u_0 is in T_P. If $\xi(u_1) = H$ (if $\xi(u_1) = J$) then operation $\eta_{i',j'}$ only inserts edges between vertices from graph $G(u'_0)$ labeled by labels of $L_H(u_1)$ (of $L_J(u_1)$) and vertices from $G(u_0)$ not labeled by labels of $L_J(u_1)$ (of $L_H(u_1)$, respectively).*

Proof: If $\xi(u_1) = H$ (if $\xi(u_1) = J$) then all vertices of $G(u'_0)$ are from U_H (from U_J, respectively). Since $\eta_{i',j'}$ creates at least one edge between a vertex from $G(u'_0)$ and a vertex from $G(u_0)$ and since there is no edge between a vertex from U_H and a vertex from U_J, one label of $\{i', j'\}$ has to be in $L_H(u_1)$ (in $L_J(u_1)$) and the other label of $\{i', j'\}$ can not be in $L_J(u_1)$ (in $L_H(u_1)$, respectively). □

The next lemma shows that the relabeling operations of a frame

$$q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$$

from a 3.*b*-path with $\xi(u_1) = H$ relabels only inner vertices from H.

Lemma 7.9 *Let $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of a 3.b-path p such that node u_s, $r' < s \leq s'$, is a relabeling node labeled by $\rho_{i \to j}$.*

1. *If $\xi(u_1) = H$ then $i \in L_H(u_{s-1}) - L_P(q) - L_J(u_{s-1})$ and $j \in L_H(u_{s-1})$.*
2. *If $\xi(u_1) = J$ then $i \in L_J(u_{s-1}) - L_P(q) - L_H(u_{s-1})$ and $j \in L_J(u_{s-1})$.*

Proof: Since in a 3.*b*-path p, the labels of $L_P(q)$ and $L_{H \cap J}(q)$ are not relabeled, label i can only be in $L_H(u_{s-1}) - L_P(q) - L_J(u_{s-1})$ or $L_J(u_{s-1}) - L_P(q) - L_H(u_{s-1})$.

If $i \in L_H(u_{s-1}) - L_P(q) - L_J(u_{s-1})$ then we get $j \in L_H(u_{s-1})$, otherwise $L_{P \cap H}(u_{s-1}) \subsetneq L_{P \cap H}(u_s)$ or $L_{H \cap J}(u_{s-1}) \subsetneq L_{H \cap J}(u_s)$. Both are not possible in a 3.*b*-path. On the other hand, if $i \in L_J(u_{s-1}) - L_P(q) - L_H(u_{s-1})$ then we get $j \in L_J(u_{s-1})$, otherwise $L_{P \cap J}(u_{s-1}) \subsetneq L_{P \cap J}(u_s)$ or $L_{H \cap J}(u_{s-1}) \subsetneq L_{H \cap J}(u_s)$.

Let u_0 be the child of u_1 which is in T_P. Assume first that $\xi(u_1) = H$, $i \in L_J(u_{s-1}) - L_P(q) - L_H(u_{s-1})$, and $j \in L_J(u_{s-1})$. Then $G(u_0)$ has a vertex labeled by i and a vertex labeled by j.

1. If j is not involved in an edge insertion operation of the nodes $u_2, \ldots, u_r$ then in $G(u_0)$ label i can be relabeled into j, without changing $G(u_{s'})$. This contradicts our normal form Property 2.(c).

2. If j is involved in some edge insertion operation of the nodes $u_2, \ldots, u_r$ then j is contained in $L_{H\cap J}(u_1) = L_{H\cap J}(q) = L_{H\cap J}(u_0)$ and thus $\mathrm{forb}_P(u_{s'}, j) = \mathrm{forb}_P(u_0, j)$, and we can also relabel i into j in graph $G(u_0)$ without changing the resulting graph $G(u_{s'})$. This also contradicts our normal form Property 2.(c).

Thus, we get $i \in L_H(u_{s-1}) - L_P(q) - L_J(u_{s-1})$ and $j \in L_H(u_{s-1})$. For $\xi(u_1) = J$, we get $i \in L_J(u_{s-1}) - L_P(q) - L_H(u_{s-1})$ and $j \in L_J(u_{s-1})$. □

In the proof of the next lemma, we will frequently rearrange frames in a path of type 3.*b*. Assume a path p consists of two consecutive frames, i.e.,

$$p = (u_1, \ldots, u_{r'}, \ldots, u_{s'},\ u_{s'+1}, \ldots, u_{r''}, \ldots, u_{s''}),$$

where u_1 and $u_{s'+1}$ are union nodes, $u_2, \ldots, u_{r'}$ and $u_{s'+2}, \ldots, u_{r''}$ are edge insertion nodes, and $u_{r'+1}, \ldots, u_{s'}$ and $u_{r''+1}, \ldots, u_{s''}$ are relabeling nodes. Let u_0' and u_0 be the two children of u_1, where u_0 is in T_P, and let u_0'' be the child of $u_{s'+1}$ which is not in T_P.

If we exchange the two frames of p then we get the new path

$$p' = (u_{s'+1}, \ldots, u_{r''}, \ldots, u_{s''}, u_1, \ldots, u_{r'}, \ldots, u_{s'}).$$

In the resulting expression tree, union node $u_{s'+1}$ has the two children u_0'' and u_0, and union node u_1 has the two children u_0' and $u_{s''}$. The left-right order of the children of u_1 and $u_{s'+1}$ is not changed. That is, if u_0' is the left child (right child) of u_1 in the original expression tree then u_0' is the left child (right child, respectively) of u_1 in the new expression tree, and if u_0'' is the left child (right child) of $u_{s'+1}$ in the original expression tree then u_0'' is the left child (right child, respectively) of $u_{s'+1}$ in the new expression tree.

This rearrangement changes the expression defined by the original expression tree $T(u_{s''})$ as follows, see also Figure 14. Let X_1, X_2, X_3 be the expressions defined by the expression trees $T(u_0')$, $T(u_0)$, and $T(u_0'')$, respectively. Without loss of generality, let u_0' be the left child of u_1 and u_0'' be the right child of $u_{s'+1}$. Let $\eta_{i_2,j_2}, \ldots, \eta_{i_{r'},j_{r'}}$ be the edge insertion operations of the nodes $u_2, \ldots, u_{r'}$, let $\rho_{i_{r'+1}\to j_{r'+1}}, \ldots, \rho_{i_{s'}\to j_{s'}}$ be the relabeling operations of the nodes $u_{r'+1}, \ldots, u_{s'}$, let $\eta_{i_{s'+2},j_{s'+2}}, \ldots, \eta_{i_{r''},j_{r''}}$ be the edge insertion operations of the nodes $u_{s'+2}, \ldots, u_{r''}$, and let $\rho_{i_{r''+1}\to j_{r''+1}}, \ldots, \rho_{i_{s''}\to j_{s''}}$ be the relabeling operations of the nodes $u_{r''+1}, \ldots, u_{s''}$. Then the original expression defined by $T(u_{s''})$ is

$$\rho_{i_{s''}\to j_{s''}}(\cdots \rho_{i_{r''+1}\to j_{r''+1}}(\eta_{i_{r''},j_{r''}}(\cdots \eta_{i_{s'+2},j_{s'+2}}(Y \oplus X_3)\cdots))\cdots),$$

where

$$Y = \rho_{i_{s'}\to j_{s'}}(\cdots \rho_{i_{r'+1}\to j_{r'+1}}(\eta_{i_{r'},j_{r'}}(\cdots \eta_{i_2,j_2}(X_1 \oplus X_2)\cdots))\cdots).$$

The expression of the new expression tree $T(u_{s'})$ which we get after exchanging the two frames is

$$\rho_{i_{s'}\to j_{s'}}(\cdots \rho_{i_{r'+1}\to j_{r'+1}}(\eta_{i_{r'},j_{r'}}(\cdots \eta_{i_2,j_2}(X_1 \oplus Y')\cdots))\cdots),$$

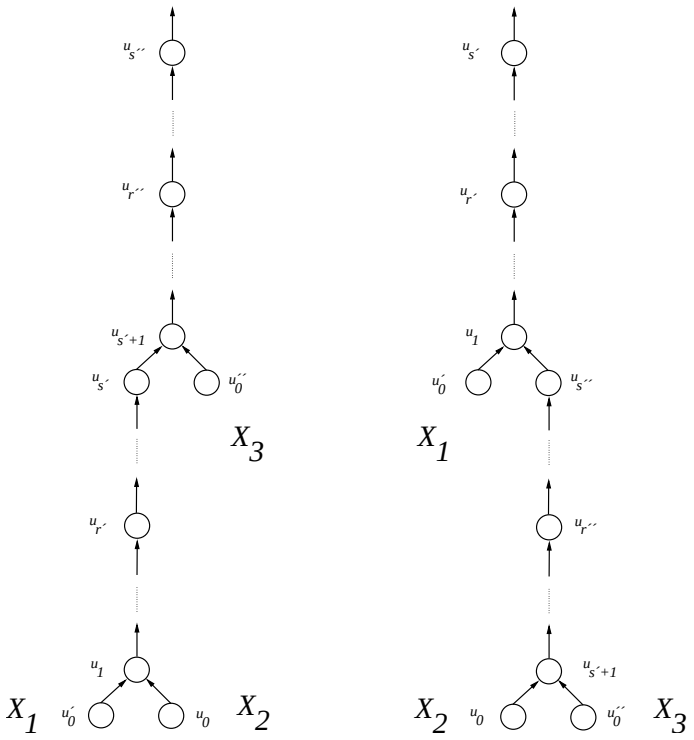


Figure 14: The rearrangement of two frames.

where

$$Y' = \rho_{i_{s''} \to j_{s''}}(\cdots \rho_{i_{r''+1} \to j_{r''+1}}(\eta_{i_{r''}, j_{r''}}(\cdots \eta_{i_{s'+2}, j_{s'+2}}(X_2 \oplus X_3)\cdots))\cdots).$$

Note that the new expression and the original expression do not need to be equivalent, but the order of the leaves in the tree is not changed.

We need the following additional notation. Let u_s be a node of T. The labels of $L_H(u_s) - L_P(u_s) - L_J(u_s)$ and $L_J(u_s) - L_P(u_s) - L_H(u_s)$ are called the *unfixed labels* of $G(u_s)$. Within a path of type 3.*b*, only unfixed labels are relabeled, see also Lemma 7.9. The vertices labeled by unfixed labels of $G(u_s)$ are called *unfixed vertices* of $G(u_s)$.

Lemma 7.10 *There is always a k-expression X in normal form such that tree T satisfies Property 7.2, Property 7.4, Property 7.6, and additionally Property 7.11.*

Property 7.11 *Every 3.b-path p is divided into at most $3 \cdot 2^{2(k+1)}$ paths p' such that either for all frames $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of p' $\xi(u_1) = H$ or for all frames $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ of p' $\xi(u_1) = J$.*

Proof: Let $q_1, \ldots, q_t$ be the frames of a 3.b-path p, i.e., $p = q_1 \odot \cdots \odot q_t$, where operation $\odot$ is the concatenation of paths.

For a frame $q = (u_1, \ldots, u_{s'})$ of p, let $\xi(q) := \xi(u_1)$, $\mathrm{unfix}_H(q) = |L_H(u_{s'}) - L_P(u_{s'}) - L_J(u_{s'})|$, $\mathrm{unfix}_J(q) = |L_J(u_{s'}) - L_P(u_{s'}) - L_H(u_{s'})|$ and

$$\begin{array}{lcll} \min_H(p) & = & \min\{\mathrm{unfix}_H(q_1), \ldots, \mathrm{unfix}_H(q_t)\} & \text{and} \\ \min_J(p) & = & \min\{\mathrm{unfix}_J(q_1), \ldots, \mathrm{unfix}_J(q_t)\}. & \end{array}$$

Let r_1, r_2, $1 \le r_1 \le r_2 \le t$, such that $r_2 - r_1$ is maximal and either

$$\begin{array}{lcllcllcl} \mathrm{unfix}_H(q_{r_1}) & = & \min_H(p) & \text{and} & \mathrm{unfix}_J(q_{r_2}) & = & \min_J(p) & \text{or} \\ \mathrm{unfix}_J(q_{r_1}) & = & \min_J(p) & \text{and} & \mathrm{unfix}_H(q_{r_2}) & = & \min_H(p). & \end{array}$$

If $q_{i_1}, \ldots, q_{i_n}$, $1 \le i_1 < i_2 < \cdots < i_n \le t$, are the frames for which

$$\mathrm{unfix}_H(q_{i_1}) = \mathrm{unfix}_H(q_{i_2}) = \cdots = \mathrm{unfix}_H(q_{i_n}) = \min_H(p)$$

and if $q_{j_1}, \ldots, q_{j_m}$, $1 \le j_1 < j_2 < \cdots < i_m \le t$, are the frames for which

$$\mathrm{unfix}_J(q_{j_1}) = \mathrm{unfix}_J(q_{j_2}) = \cdots = \mathrm{unfix}_J(q_{j_m}) = \min_J(p),$$

then either $r_1 = i_1$ and $r_2 = j_m$ or $r_1 = j_1$ and $r_2 = i_n$.

We divide the path p into three parts $p_{\mathrm{first}}, p_{\mathrm{middle}}, p_{\mathrm{last}}$ such that $p = p_{\mathrm{first}} \odot p_{\mathrm{middle}} \odot p_{\mathrm{last}}$. Subpath p_{middle} starts with frame q_{r_1} and ends with frame q_{r_2}, see also Figure 15.

If the first part p_{first} or the last part p_{last} of p are not empty then they will be partitioned in the same way as p. Since

$$\begin{array}{lcllcllcl} \min_J(p_{\mathrm{first}}) & > & \min_J(p) & \text{and} & \min_H(p_{\mathrm{last}}) & > & \min_H(p) & \text{or} \\ \min_H(p_{\mathrm{first}}) & > & \min_H(p) & \text{and} & \min_J(p_{\mathrm{last}}) & > & \min_J(p), & \end{array}$$

the partition procedure yields at most $2^{2(k+1)}$ such paths p_{middle}.

Assume $\mathrm{unfix}_H(q_{r_1}) = \min_H(p)$ and $\mathrm{unfix}_J(q_{r_2}) = \min_J(p)$. The second case where $\mathrm{unfix}_J(q_{r_1}) = \min_J(p)$ and $\mathrm{unfix}_H(q_{r_2}) = \min_H(p)$ runs analogously. Then $\xi(q_{r_1}) = H$ and $\xi(q_{r_2}) = J$ and we rearrange the frames in path p_{middle} such that in the new path there is first frame q_{r_1}, then all frames q with $\xi(q) = J$ and then all remaining frames q with $\xi(q) = H$. If we move all frames q of p_{middle} where $\xi(q) = J$ to the front, then the remaining frames q of p_{middle} where $\xi(q) = H$ (except frame q_{r_1}) will automatically move to the end. This rearrangement will yield at most $3 \cdot 2^{2(k+1)}$ paths such that for all frames q of every path all $\xi(q)$ are equal.

The order of the frames q with $\xi(q) = J$ or $\xi(q) = H$ is not changed, i.e, it is the same order as in the original path p_{middle}. The order of the nodes in the frames is also not changed when moving frames. To ensure that the resulting expression is really equivalent to the original one, we perform a relabeling of the unfixed labels as follows.

For every node u_s of the new path p_{middle}, we define step by step a bijection $b_{u_s} : [k] \to [k]$. The idea is to use for the operations on subgraph $G(u_s)$ label

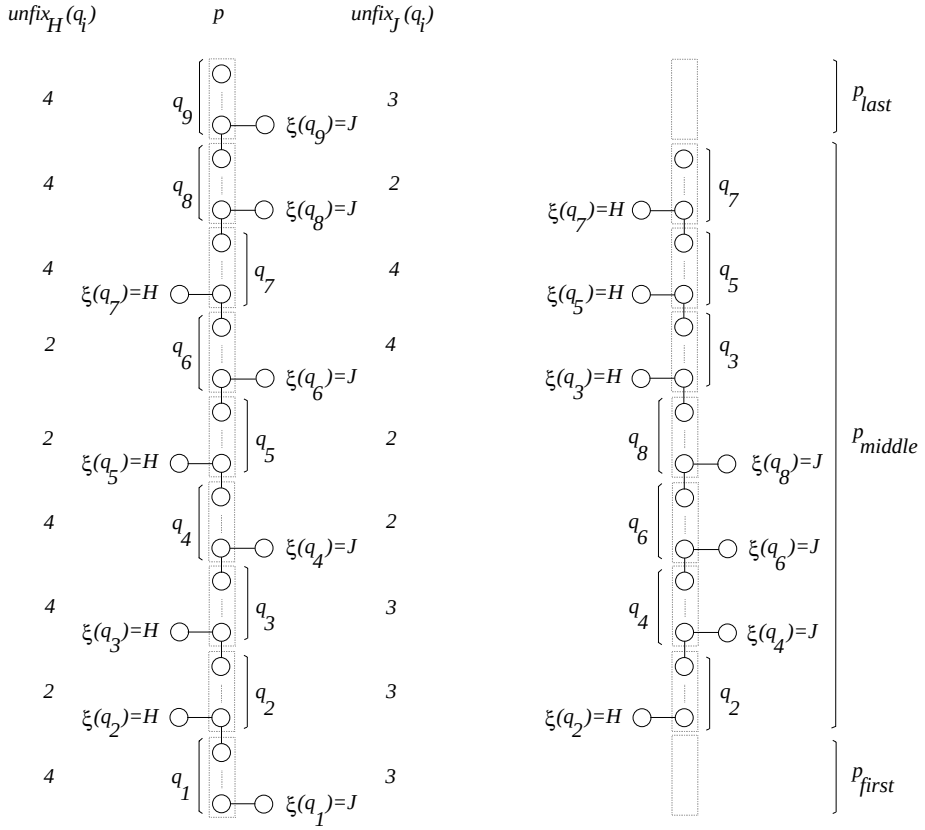


Figure 15: The partition of path p into three parts $p_{\text{first}}, p_{\text{middle}}, p_{\text{last}}$, where $\min_H(p) = 2$, $\min_J(p) = 2$, $r_1 = 2$, and $r_2 = 8$

$b_{u_s}(i)$ instead of label i. For all labels $i \in L_P(u_s) \cup L_{H\cap J}(u_s)$, we will have $b_{u_s}(i) = i$, because these labels are not relabeled along a path of type 3.*b*.

The bijections for the nodes of q_{r_1} are identities. The other bijections are defined step by step depending on the operations of the nodes along the new path p_{middle}.

We consider the nodes u_s of the new path p_{middle} in the given order starting with the parent node of the last node of frame q_{r_1}.

1. If u_s is a union node then let u'_{s-1} and u_{s-1} be the two children of u_s, where u'_{s-1} is not in T_P. The bijection b_{u_s} of u_s is initially the bijection $b_{u_{s-1}}$ of the child u_{s-1}.

   We then simultaneously replace in the expression of subtree $T(u'_{s-1})$ every label i by $b_{u_s}(i)$. After that, we verify whether there is a vertex w of $G(u'_{s-1})$ which is an unfixed vertex in the original tree $T(u_s)$ but not an unfixed vertex in the new tree $T(u_s)$.

If there is such a vertex w originally labeled by i and now labeled by $b_{u_s}(i)$, then we choose an arbitrary label j such that $b_{u_s}(j)$ is not used up to now, i.e., $b_{u_s}(j) \notin L_P(u_s) \cup L_H(u_s) \cup L_J(u_s)$ with respect to the expression defined up to now by the new tree $T(u_s)$. We define $b_{u_s}(i) := b_{u_s}(j)$ and $b_{u_s}(j) := b_{u_s}(i)$, and simultaneously exchange in the expression of subtree $T(u'_{s-1})$ every label $b_{u_s}(i)$ by $b_{u_s}(j)$ and every label $b_{u_s}(j)$ by $b_{u_s}(i)$.

2. If u_s is an edge insertion node labeled by $\eta_{i,j}$, then bijection b_{u_s} is the bijection $b_{u_{s-1}}$ and node u_s will be labeled by operation $\eta_{b_{u_s}(i),b_{u_s}(j)}$.

3. If u_s is a relabeling node labeled by $\rho_{i\to j}$, then bijection b_{u_s} is the bijection $b_{u_{s-1}}$ and node u_s will be labeled by operation $\rho_{b_{u_s}(i)\to b_{u_s}(j)}$.

For the final node u_s of the new path p_{middle} we perform one additional relabeling of the resulting expression defined by $T(u_s)$ such that the unfixed vertices of $G(u_s)$ are labeled as in the graph defined by the final node of the original path p_{middle}.

All these relabeling steps are possible, because for all nodes u_s in the first part of p_{middle} $\text{unfix}_H(u_s) = \min_H(p)$, and for all nodes u_s of the last part of p_{middle} $\text{unfix}_J(u_s) = \min_J(p)$. Thus, there are always enough unused labels to relabel the unfixed vertices. Note that the labels of the sets $L_P(u_s)$ and $L_{H\cap J}(u_s)$ are unchanged along the nodes of p_{middle}.

It remains to show that the new expression is equivalent to the original expression. Let $q = (u_1, \ldots, u_{r'}, \ldots, u_{s'})$ be a frame of the original path p_{middle} where $\xi(u_1) = H$. The other case where $\xi(u_1) = J$ runs analogously and is even less complicated. Frame q can be moved by the rearrangement toward the end of p_{middle}. Let u'_0 and u_0 be the two children of u_1, where u_0 is in T_P. Node u'_0 is also a child of u_1 in the new expression tree, because the children of the union nodes which are not in T_P are not changed by the rearrangement of the frames.

Consider now an edge insertion operation $\eta_{i,j}$ of some node u_r, $1 < r \leq r'$, of frame q in the original expression. By Lemma 7.8, we know that the edge insertion operation $\eta_{i,j}$ of node u_r creates only edges between vertices from $G(u'_0)$ and vertices from $G(u_0)$. We also know that one of the two labels i, j is from $L_H(u_1)$ and the other is not from $L_J(u_1)$. Without loss of generality, let $j \in L_H(u_1)$ and $i \notin L_J(u_1)$.

The rearrangement of the frames does not change the order of the leaves in the expression tree, see Figure 14. It also does not change the order of the frames q with the same $\xi(q)$ on path p_{middle}. Since $\eta_{i,j}$ creates only edges between vertices from U_H and vertices from $U_H \cup U_P$, all these edges are also created by the corresponding edge insertion operation $\eta_{b_{u_r}(i),b_{u_r}(j)}$ in the new expression.

Assume edge insertion operation $\eta_{b_{u_r}(i),b_{u_r}(j)}$ in the new expression creates an edge which is not in the graph defined by the original expression. Then one of the nodes of this edge has to be in U_J. This node can only be labeled by $b_{u_r}(j)$, because $i \notin L_J(u_1)$ for u_1 from the original expression tree, and by our

relabeling procedure $b_{u_r}(i) \notin L_J(u_1)$ for u_1 from the new expression tree. Now we get $b_{u_r}(j) \in L_{H\cap J}(u_1)$, $b_{u_r}(i) \in L_P(u_1)$, and $b_{u_r}(i) \notin \text{forb}_P(u_{s'}, b_{u_r}(j))$ for u_1 from the new expression tree. Since all sets $L_P(u_s)$ are equal for all nodes u_s of p_{middle}, all sets $L_{H\cap J}(u_s)$ are equal for all nodes u_s of p_{middle}, and all sets $\text{forb}_P(u_s, j)$ are equal for all the last nodes u_s of all frames of p_{middle}, and since these labels are not relabeled by our relabeling procedure, we get that all edges created by $\eta_{b_{u_r}(i),b_{u_r}(j)}$ have to be in the graph defined by the original expression. This contradicts our assumption.

Thus the original expression and the new expression are equivalent. Note that the normal form property and the Properties 7.2, 7.4, and 7.6 are also not changed by the rearrangement of the frames. □

Lemma 7.10 allows us to divide every 3.*b*-path into at most $3 \cdot 2^{2(k+1)}$ *paths of type* 4. The paths of type 4 are the those parts of the paths p_{middle} in that for all frames q all $\xi(q)$ are equal.

The connection type of H

Let us summarize how the paths of tree T_P are partitioned now. Tree T_P consists of

1. at most $2l - 1$ paths of type 1.*a*,
2. at most $(2l - 1) \cdot (3k - 1)$ paths of type 2.*a*,
3. at most $(2l - 1) \cdot 3k \cdot (k + 1)^{k+1}$ paths of type 3.*a*, and
4. at most $(2l - 1) \cdot 3k \cdot ((k + 1)^{k+1} + 1) \cdot 3 \cdot 2^{2(k+1)}$ paths of type 4.

Every non-leaf node of T_P is in exactly one of these paths of type 1.*a*, 2.*a*, 3.*a*, or 4. Every path of type 1.*a*, 2.*a*, or 3.*a* has at most $\binom{k}{2} + k$ nodes, because these paths are frames. For all frames $q = (u_1, \ldots, u_{s'})$ in a path of type 4 all $\xi(u_1)$ are equal, all sets $L_P(q)$ and $L_{H\cap J}(q)$ are equal, and all sets $\text{forb}_P(u_{s'}, j)$ are equal for all $j \in L_P(q) \cup L_{H\cap J}(q)$. For a node u_s of T_P, let $L_P(u_s)$ be the set of all *terminal labels*, $L_H(u_s)$ be the set of all *internal labels*, and $L_J(u_s)$ be the set of all *external labels* for node u_s.

We now replace every 4-path $p = (u_1, \ldots, u_{s'})$ of T_P which consists of more than one frame by some so-called *bridge node* node v. Let u_0 be the child of u_1 which is in T_P and $u_{s'+1}$ be the parent node of $u_{s'}$ in T_P. Then the path $(u_0, u_1, \ldots, u_{s'}, u_{s'+1})$ is replaced by path $(u_0, v, u_{s'+1})$. Node v is called an *internal* bridge node if $\xi(u_1) = H$ and an *external bridge node* if $\xi(u_1) = J$. Every bridge node represents a 4-path with more than one frame. Note that in a succeeding replacement the nodes u_0 and $u_{s'+1}$ can also be bridge nodes. At every bridge node we store the information whether it is internal or external, and the set of all terminal labels $L_P(u_{s'})$, the set of all internal labels $L_H(u_{s'})$, the set of all external labels $L_J(u_{s'})$, and the pairs $(\text{forb}_P(u_{s'}, j), j)$ for all $j \in L_P(u_{s'}) \cup L_{H\cap J}(u_{s'})$.

A union node u_1 is called an *internal union node* if $\xi(u_1) = H$ and an *external union node* if $\xi(u_1) = J$. At every union node u_s of T_P for which $\xi(u_s)$ is defined, we store the information whether u_s is internal or external.

At every non-bridge node u_s of T_P we store additionally to the clique-width operation the set of all terminal labels $L_P(u_s)$, the set of all internal labels $L_H(u_s)$, the set of all external labels $L_J(u_s)$, and all pairs $(\text{forb}_P(u_s, j), j)$ for all $j \in L_P(u_s) \cup L_{H\cap J}(u_s)$. If a leaf u_s of T_P represents a vertex of G obtained by joining the i-th terminal vertex from H with the i-th terminal vertex from J, then leaf u_s is additionally labeled by index i. The result C is called a *connection tree* for the k-labeled l-terminal graph H.

The set of all mutually different connection trees of H with respect to all k-labeled l-terminal graphs J is called the *connection type* of H. Two connection trees C_1, C_2 for H are *equivalent* if there is a bijection b between the nodes of C_1 and C_2 such that

1. node u_{s-1} is a child (left child, right child) of node u_s in C_1 if and only if node $b(u_{s-1})$ is a child (left child, right child, respectively) of node $b(u_s)$ in C_2,
2. node u_s of C_1 is an external or internal union node if and only if node $b(u_s)$ of C_2 is an external or internal union node, respectively,
3. node u_s of C_1 is an external or internal bridge node if and only if node $b(u_s)$ of C_2 is an external or internal bridge node, respectively,
4. node u_s of C_1 and node $b(u_s)$ of C_2 store the same terminal label sets, internal label sets, external label sets, the same pairs $(\text{forb}_P(u_s, j), j)$, and the same clique-width operation,
5. node u_s of C_1 and node $b(u_s)$ of C_2 store the same index if u_s and $b(u_s)$ are leaves representing a vertex obtained by joining to terminal vertices.

Note that the two notions *connection tree* and *connection type* are always defined with respect to graph property clique-width at most k. For better readability, we will sometimes omit this extension.

8 Main result

The following theorem implies the main result of this paper.

Theorem 8.1 *If two k-labeled l-terminal graphs are of the same connection type with respect to graph property clique-width at most k, then they are replaceable with respect to graph property clique-width at most k.*

Proof: Let H_1 and H_2 be two k-labeled l-terminal graphs such that H_1 and H_2 are of the same connection type. Let J be any k-labeled l-terminal graph such that $H_1 \circ J$ has clique-width at most k. We will show that $H_2 \circ J$ has

also clique-width at most k. This implies that H_1 and H_2 are replaceable with respect to graph property clique-width at most k.

Let T_1 be a k-expression tree for $H_1 \circ J$ which defines connection tree C. Let $T_{1,P}$ be the subtree of T_1 defined by the leaves of T_1 which represent the joined terminal vertices of H_1 and J, and by the nodes on the paths from these leaves to the root of T_1.

Since H_1 and H_2 are of the same connection type, C is also a connection tree for H_2 with respect to some k-labeled l-terminal graph J'. Let T' be a k-expression tree for $H_2 \circ J'$ which defines connection tree C. Let T'_P be the subtree of T' defined by the leaves of T' which represent the joined terminal vertices of H_2 and J', and by the nodes on the paths from these leaves to the root of T'.

Since T_1 and T' define the same connection tree C, there is a one-to-one correspondence between some nodes of T_1, T', and C. For a node u of C, we write u^C to indicate that u is a node of C. If v is the corresponding node of T_1, then we write u^{T_1} for v. The corresponding node in T' is denoted by $u^{T'}$. We use this notation also for frames and paths.

Our aim is to define a new k-expression tree T_2 from T_1 and T' such that T_2 defines $H_2 \circ J$. We start with a copy T'_1 of the k-expression tree T_1. Let $T'_{1,P}$ be defined for T'_1 in the same way as $T_{1,P}$ is defined for T_1. Let u_1^C be an internal union node, let $u_0^{T'_1}$ be the child of $u_1^{T'_1}$ which is not in $T'_{1,P}$, and let $u_0^{T'}$ be the child of $u_1^{T'}$ which is not in T'_P. By our notation, it is clear from which trees these nodes are. For every such node u_0^C we replace in the copy T'_1 of T_1 the subtrees $T'_1(u_0^{T'_1})$ by the subtrees $T'(u_0^{T'})$. Let u^C be an internal bridge node, let $p^{T'_1}$ be the corresponding 4-path in T'_1 and $p^{T'}$ be the corresponding 4-path in T'. For every such node we substitute in the current tree T'_1 the 4-path $p^{T'_1}$ by the 4-path $p^{T'}$. This substitution includes all the subtrees at the children of the union nodes of $p^{T'_1}$ and $p^{T'}$ which are not in $T'_{1,P}$ and T'_P, respectively. The resulting tree is denoted by T_2. It is clear that T_2 is a k-expression tree.

It remains to show that k-expression tree T_2 defines $H_2 \circ J$. Let $T_{2,P}$ be the subtree of T_2 defined by the leaves of T_2 which represent the joined terminal vertices of H_2 and J, and by the nodes on the paths from these leaves to the root of T_2.

We first show that the vertices in the k-labeled graph $H_2 \circ J$ are labeled as in the k-labeled graph defined by k-expression tree T_2. There is obviously a one-to-one correspondence between the vertices of $H_2 \circ J$ and the vertices of the graph defined by T_2, because T_2 is constructed from T_1 and T' which define J and H_2. Let T_{2,H_2} and $T_{2,J}$ be the subtrees of T_2 defined by the leaves which represent vertices of H_2 and J, respectively, and by the nodes on the paths from these leaves to the roots. The vertices of H_2 are labeled in $H_2 \circ J$ as in the graph defined by k-expression tree T_2, because these vertices are only relabeled by relabeling operations of T_{2,H_2} which do not belong to the external 4-paths of $T_{2,P}$. (Here external and internal 4-path means that the path is copied from T_1 and T', respectively.) The same holds for the vertices of J, because these vertices are only relabeled by relabeling operations of $T_{2,J}$ which do not belong

to the internal 4-paths of $T_{2,P}$.

Next we show that all edges of $H_2 \circ J$ are also in the graph defined by T_2. Let T'_{H_2} and $T_{1,J}$ be the subtrees of T' and T_1, respectively, defined by the leaves which represent vertices of H_2 and J, respectively, and by the nodes on the paths from these leaves to the roots. Let e be an edge of $H_2 \circ J$. If the end vertices of e are both from H_2 or both from J then e is created by an edge insertion node $u_s^{T'}$ or $u_s^{T_1}$ which is also in T'_{H_2} or $T_{1,J}$, respectively. The composition of T_2 now implies that node $u_s^{T_2}$ exists in T_2 and the corresponding edge is also contained in the graph defined by T_2. Thus, all edges of $H_2 \circ J$ are in the graph defined by T_2.

The most interesting part is to show that the edge insertion operations of T_2 do not create any edge which is not in $H_2 \circ J$. An edge insertion node $u_s^{T_2}$ of T_2 which does not belong to $T_{2,P}$ creates only edges which are also in $H_2 \circ J$, because the corresponding subtree defined by $T_2(u_s^{T_2})$ is either completely copied from T' or completely copied from T_1.

Assume now the edge insertion node $u_s^{T_2}$ belongs also to $T_{2,P}$. Let $\eta_{i,j}$ be the edge insertion operation of $u_s^{T_2}$. If $u_s^{T_2}$ is not from a 4-path of $T_{2,P}$ which consists of more than one frame, then $u_s^{T_2}$ is also in C. Then there is an equivalent edge insertion node $u_s^{T_1}$ or $u_s^{T'}$ in T_1 or T' which is labeled as $u_s^{T_2}$ in T_2. This equal labeling ensures that the edge insertion operation $\eta_{i,j}$ defines a new edge between a vertex labeled by i and a vertex labeled by j if it defines at least one such edge in T_1 or T'.

If $u_s^{T_2}$ is from a 4-path p^{T_2} which is also in $T_{2,P}$ and which consists of more than one frame, then without loss of generality, let p^{T_2} be copied from T', i.e., let p^{T_2} be an internal 4-path for which $\xi(q^{T_2}) = H_2$ for all frames q^{T_2} of p^{T_2}. Let u^C be the corresponding internal bridge node for p^{T_2} and let u'^C be the child of u^C in C. The child u'^C can be a bridge node or a usual node. If it is a usual node then the equal labeling of u'^{T_2} and $u'^{T'}$ ensures that $\eta_{i,j}$ defines a new edge between a vertex labeled by i and a vertex labeled by j if it defines at least one such edge in T'.

If child u'^C is a bridge node then let v^{T_2} be the last node of the path of T_2 which is represented by u'^C in C. If u'^C is an internal (external) bridge node then the equal labeling of v^{T_2} and $v^{T'}$ (and v^{T_1}, respectively) ensures that $\eta_{i,j}$ defines a new edge between a vertex labeled by i and a vertex labeled by j if it defines such an edge in T'. Thus, every edge in the graph defined by T_2 is also in $H_2 \circ J$, and vice versa. □

By Theorem 8.1 and the fact that there is only a finite number of connection types for fixed integers l and k it follows that the equivalence relation $\sim_{\Pi_k,l}$ has a finite number of equivalence classes, where Π_k is the graph property clique-width at most k. This implies the following corollary.

Corollary 8.2 *For every integer k, there exists a linear time algorithm for deciding clique-width at most k of a graph of bounded tree-width.*

Since the clique-width of a graph of tree-width l is bounded by $3 \cdot 2^{l-1}$, see [CR01], there is also an algorithm which minimizes the clique-width of a graph

of bounded tree-width in linear time.

Corollary 8.3 *There exists a linear time algorithm for computing the clique-width of a graph of bounded tree-width.*

The corollary above only states that such a linear-time algorithm for deciding clique-width k for graphs of bounded tree-width exists. Although the proof is constructive, the resulting algorithm seems to be only interesting from a theoretical point of view.

Note that our result does not imply that the clique-width k property is expressible in counting MSO_2-logic. The equivalence between a finite number of equivalence classes of $\sim_{\Pi_k,l}$ and monadic second-order definability is only given for special graph classes as for example for graphs of bounded tree-width [Lap98], but not for the class of all graphs.

Acknowledgments

The authors wish to thank the anonymous referees for several useful suggestions.

References

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

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems on graphs embedded in k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989.

[Arn85] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985.

[Bod96] H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[Bod97] H.L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of Mathematical Foundations of Computer Science*, volume 1295 of *LNCS*, pages 29–36. Springer-Verlag, 1997.

[Bod98] H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[CHL+00] D.G. Corneil, M. Habib, J.M. Lanlignel, B. Reed, and U. Rotics. Polynomial time recognition of clique-width at most three graphs. In *Proceedings of Latin American Symposium on Theoretical Informatics (LATIN '2000)*, volume 1776 of *LNCS*. Springer-Verlag, 2000.

[CMR00] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

[CO00] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77–114, 2000.

[Cou90] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[CPS85] D.G. Corneil, Y. Perl, and L.K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

[CR01] D.G. Corneil and U. Rotics. On the relationship between clique-width and treewidth. In *Proceedings of Graph-Theoretical Concepts in Computer Science*, volume 2204 of *LNCS*, pages 78–90. Springer-Verlag, 2001.

[EGW01] W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *Proceedings of Graph-Theoretical Concepts in Computer Science*, volume 2204 of *LNCS*, pages 117–128. Springer-Verlag, 2001.

[GR00] M.C. Golumbic and U. Rotics. On the clique-width of some perfect graph classes. *IJFCS: International Journal of Foundations of Computer Science*, 11(3):423–443, 2000.

[GW00] F. Gurski and E. Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In *Proceedings of Graph-Theoretical Concepts in Computer Science*, volume 1938 of *LNCS*, pages 196–205. Springer-Verlag, 2000.

[Joh98] Ö. Johansson. Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. *Congressus Numerantium*, 132:39–60, 1998.

[KR01] D. Kobler and U. Rotics. Polynomial algorithms for partitioning problems on graphs with fixed clique-width. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 468–476. ACM-SIAM, 2001.

[Lap98] D. Lapoire. Recognizability equals definability, for every set of graphs of bounded tree-width. In *Proceedings 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *LNCS*, pages 618–628. Springer-Verlag, 1998.

[LW88] T. Lengauer and E. Wanke. Efficient solution of connectivity problems on hierarchically defined graphs. *SIAM Journal on Computing*, 17(6):1063–1080, 1988.

[LW93] T. Lengauer and E. Wanke. Efficient analysis of graph properties on context-free graph languages. *Journal of the ACM*, 40(2):368–393, 1993.

[Wan94] E. Wanke. k-NLC graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54:251–266, 1994,
revised version: `http://www.cs.uni-duesseldorf.de/~wanke`.