Grammars and clique-width bounds from split decompositions

Bruno Courcelle

Grammars and clique-width bounds from split decompositions

Bruno Courcelle
Labri, CNRS and Bordeaux University*
33405 Talence, France
email: courcell@labri.fr

July 1, 2019

Abstract

Graph decompositions are important for algorithmic purposes and for graph structure theory. We relate the *split decomposition* introduced by Cunnigham to *vertex substitution*, *graph grammars* and *clique-width*.

For this purpose, we extend the usual notion of substitution, upon which *modular decomposition* is based, by considering graphs with *dead* (or *non-boundary*) vertices. We obtain a graph grammar for distance-hereditary graphs consisting of four rules collected in a single equation. We also bound the clique-width of a graph in terms of those of the components of a split decomposition that need not be canonical.

For extending these results to directed graphs and their split decompositions (that we handle formally as *graph-labelled trees*), we need another extension of substitution : instead of two types of vertices, dead or *alive* as for undirected graphs, we need four types, in order to encode edge directions. We bound linearly the clique-width of a directed graph G in terms of the maximal clique-width of a component arising in a graph-labelled tree that defines G. This result concerns all directed graphs, not only the strongly connected ones considered by Cunningham.

Introduction

Hierarchical graph decompositions and the associated graph complexity measures such as tree-width and clique-width are important for algorithmic purposes and graph structure theory. Tree-width and clique-width occur as parameters

*This work has been supported by the French National Research Agency (ANR) within the IdEx Bordeaux program "Investments for the future", CPU, ANR-10-IDEX-03-02, and also within the project GraphEn started in October 2015.

in many *fixed-parameter tractable* (FPT) algorithms [14, 17, 20, 24], in particular for the verification of monadic second-order properties of graphs. They are actually linearly related in many interesting cases [10, 11, 26].

These algorithms are based on the construction of finite automata that run on algebraic terms representing tree-decompositions, or on *clique-width terms* in the case where the parameter is clique-width (see Definition 1.2). However, these automata cannot be implemented in the classical way because their sets of states are much to large. The notion of *fly-automaton* [12, 13] can overcome this difficulty in many cases[1]: these automata compute their transitions instead of looking into transition tables.

Even for checking properties of graphs of bounded tree-width, it is convenient to input graphs by clique-width terms, and we develop the theory and practice of fly-automata for graphs defined in this way in [12, 13]. As for tree-width, computing the exact clique-width of a graph is an NP-complete problem [23]. However, clique-width terms witnessing "good" approximations of clique-width can be used with fly-automata. Such terms can be constructed by different algorithms, and with help of *modular* or *split decomposition*[2], in preliminary steps. Let us point a difference between the two : modular decomposition is based on a rooted tree, and is clearly hierachical. Split decomposition is based on trees without root ; by choosing a root for such a tree, one can turn the decomposition into a hierachical one, but one needs an appropriate notion of graph substitution : we define one in this article. Rank-width, in a similar way, is based on unrooted trees. However, by choosing a root and using appropriate graph operations, derived from those upon which clique-width is defined, one obtains also a hierachical decomposition [16].

Modular decomposition is related to clique-width as follows. Each undirected graph has a canonical (i.e., unique up to isomorphism) modular decomposition. Its clique-width is the maximal clique-width of a prime module of the modular decomposition (Proposition 2.112 of [14]). It has also a canonical split decomposition. Theorem 4.14 establishes that its clique-width is linearly bounded in terms of the maximal clique-width of a *prime component* of this decomposition[3], the other components being stars and cliques. Theorem 5.14 shows the same for directed graphs. These theorems improve boundings based on logic or rank-width (cf. Sections 4.4 and 5.3).

Our initial motivating example was the class of *distance-hereditary graphs* (*DH graphs* in short). They are the undirected graphs G in which the distance in any connected induced subgraph is the same as in G. They are known to have clique-width at most 3 [29, 36, 37]. However, recognizing the clique-width terms that define them is not easy. For the purpose of testing fly-automata, one may wish to generate large "random" DH graphs together with the algebraic terms of clique-width 3 that denote them. A good tool consists in using a *context-*

[1] A software is developped by Irène Durand [21]. Some parts of it are accessible online [22].

[2] See [31] and the references in that article for modular decomposition. Split decomposition defined by Cunnigham [18] is studied in [7, 27, 28]. We give definitions in Sections 4 and 5. Modular and split decomposition can both be computed in linear time for undirected graphs.

[3] Prime components cannot be *split*, cf. Section 4.1.

free graph grammar, built from clique-width operations that use three labels[4]. The characterization of DH graphs from [1], based on the addition of pendant edges and twins (see Definition 1.1), uses rewriting rules that are not those of a context-free graph grammar appropriate for using fly-automata intended to run on their derivation trees or on the equivalent clique-width terms. However, from this characterization, we can construct such a context-free grammar based on *vertex-replacement*, a notion developped in [14] and in [6], where an axiomatic definition of context-free graph grammars is given.

This construction uses a generalization of the standard notion of substitution of a graph for a vertex, that underlies the theory of modular decomposition. We distinguish in a graph H some vertices as *alive* and the others as *dead*. Dead vertices will not be linked to any others in case H is substituted into another graph[5]. We use this notion of substitution to define an associated notion of context-free *vertex-replacement grammar* (see [6, 14] for these graph grammars defined in a more general setting). We obtain a very simple grammar for DH graphs. Distance-hereditary graphs are also easily characterized in terms of their canonical split decompositions, and our grammar is based on these decompositions.

Generalizing this observation, we use grammars to generate the graphs whose components relative to split decomposition belong (up to isomorphism) to a fixed finite set $\mathcal{M}$. We bound their clique-width in terms of the maximal clique-width of a graph in $\mathcal{M}$, in a better way than what is known from [7].

Split decomposition[6] for strongly connected (directed) graphs has also been studied in [7, 18]. We extend to directed graphs our results for undirected graphs. For expressing split decompositions in terms of graph substitution, we need a more involved notion of substitution. Whereas for undirected graphs, we distinguish dead vertices from *live* ones (the others), for directed graphs, we need three types of live vertices, in order to encode three types of connections between two vertices u and v : from u to v (only), from v to u (only) or in both directions.

We consider split decompositions of directed graphs that need not be strongly connected, and we handle them formally as *graph-labelled trees*, a notion used in [7, 27]. We prove that the clique-width of a directed graph G is bounded by $8k+1$, where k is the maximum clique-width of a component of a graph-labelled tree that defines G.

Some properties of undirected graphs do not extend immediately to directed ones. Each component of a graph-labelled tree that defines an undirected graph G is isomorphic to an induced subgraphs of G, hence has no larger clique-width.

[4] But we do not get an equal probability for two DH graphs of same size. See Section 4.5 for an unambigous grammar.

[5] In [36], dead vertices are defined in clique-width terms by *inactive* labels. We will use $\perp$ as an inactive label. Dead vertices may be called *non-boundary* by analogy with the case of graphs of bounded tree-width, built by gluing small graphs at *boundary vertices* [20], also called *sources* in [14].

[6] A different notion of split decomposition for directed graphs, that generalizes also the one for undirected graphs, has been defined in [34]. We will say a few words about it in Section 5.4.

For strongly connected directed graphs, each such component H is isomorphic to a *minor* of the considered graph G. We obtain a bounding of the form[7] $cwd(H) \leq f(cwd(G))$ where f is a fixed exponential function. Improving this bound is an open problem.

To summarize, the purpose of this article is to clarify the close relationships between split decomposition, clique-width and vertex-replacement graph grammars based on vertex substitutions. In particular, we translate split decompositions of undirected graphs into graph grammars and we bound linearly the clique-width of a decomposed graph, either directed or not, in terms of those of the components. Our graph grammars can be used for counting graphs of special types and for random generation, along the lines of, e.g., [4, 25, 38].

Section 1 is devoted to basic definitions. In particular, we present our view of context-free graph grammars in terms of equation systems by using the example of cographs. Section 2 introduces vertex substitutions for undirected graphs with dead vertices, and the corresponding grammars. Section 3 relates clique-width and substitutions. Section 4 studies split decomposition of undirected graphs in this perspective, with the help of graph-labelled trees. Section 5 develops the case of directed graphs.

Acknowledgement: I thank M. Bousquet-Mélou, I. Durand, E. Gioan, M. Kanté, S. Oum, C. Paul and the referees for their useful comments and suggestions (in particular the reference to [4]). I also thank the editors of the special issue of *Discrete Applied Mathematics* relative to the GROW meeting in Toronto, in 2017 for wellcoming this submission, and the *Fields Institute* for supporting this excellent workshop.

1 Graphs and clique-width

Most definitions are well-known, we mainly review notation. We state a few facts that are either well-known or easy to prove.

The union of two sets is denoted by $\uplus$ in cases where we stress they are disjoint. The cardinality of a set X is denoted by $|X|$ and its powerset by $\mathcal{P}(X)$. The set of integers $\{1, ..., n\}$ is denoted by $[n]$.

All trees and graphs are finite.

Trees

The set of *nodes* of a tree T is denoted by N_T, and its set of *leaves*, *i.e.*, of nodes of degree 1, by L_T. A node that is not a leaf is *internal.*

If T has a root, then $<_T$ denotes the corresponding *ancestor relation*, a strict partial order on N_T (a node is not an ancestor of itself). The root, denoted by $root_T$, is the unique maximal element and the leaves different from the root are

[7] Although in general, edge contraction does not preserve bounded clique-width , [9].

the minimal ones. A *star* S_n is a tree with $n-1$ leaves linked to a single node called its *center*, where $n \geq 3$.

Graphs

We consider finite *simple graphs*, *i.e.*, that are loop-free and without parallel edges. Graphs[8] are directed or not. Directed edges are called *arcs.* A graph G has a vertex set V_G included in a fixed countably infinite set $\mathcal{V}$. Its set of edges or arcs is denoted by E_G. The corresponding binary adjacency relation is denoted by edg_G (even if G is directed). If G is undirected, we denote by uv, equvalently by vu, an edge linking vertices u and v. If G is directed, we denote by uv an arc from u to v. We also write $u - v$ or $u \to v$ to indicate that uv is, respectively, an edge or an arc.

We denote by $H \subseteq_i G$ that H is an induced subgraph of G, by $G[X]$ the induced subgraph of G with vertex set $X \subseteq V_G$, by $G - X$ the graph $G[V_G - X]$ and by $G - x$ the graph $G[V_G - \{x\}]$ where $x \in V_G$.

Definition 1.1 : *Distance-hereditary graph*

An undirected graph G is *distance-hereditary* (DH in short) if the distance of two vertices in every connected induced subgraph is the same as in G. For an example, the cycle C_4 (with 4 vertices) is DH whereas C_5 is not. The DH graphs are characterized as follows[9] : a DH graph is an isolated vertex, or the disjoint union of two DH graphs or is obtained from a DH graph by the addition of a pendant edge to a vertex x, or of a *true* or *false twin* to x. Adding to x a true twin is adding a new vertex y linked to x and to the neighbours of x. Adding a false twin is similar with y not linked to x.

Clique-width

Clique-width is based on operations that modify or combine vertex-labelled graphs. There will be some restrictions regarding the special label $\perp$ to be used in Section 3.

Definition 1.2 : *Labelled graphs and clique-width*

(a) Let C be a finite set of labels. A *C-labelled graph*, or simply, a *C-graph,* is a triple $G = (V_G, E_G, \pi_G)$ where π_G is a mapping : $V_G \to C$. Its *type*, denoted by $\tau(G)$, is $\pi_G(V_G)$, *i.e.*, the finite set of labels from C that label some vertex of G.

We denote by $\simeq$ the isomorphism of C-graphs up to vertex labels, *i.e.*, the isomorphism of the underlying unlabelled graphs, and by $\equiv$ the existence of an isomorphism that respects labels. An *abstract C-graph* (resp. *abstract graph*) is an equivalence class of $\equiv$ (resp. of $\simeq$).

(b) We define operations on C-graphs :

[8] Undefined notions are as in [19].

[9] This characterization is from [1]. The DH graphs are also the graphs of rank-width 1 [37]. They have clique-width at most 3, as we will prove in detail.

- the union of two disjoint C-graphs ; it is denoted by the binary infix function symbol[10] $\oplus$,

- the unary operation $add_{a,b}$ for $a, b \in C$, $a \neq b$ adds an undirected edge between each a-labelled vertex x and each b-labelled vertex y (unless there is already an edge xy),

- for building directed graphs, we use similarly $\overrightarrow{add}_{a,b}$ to add arcs from a-labelled to b-labelled vertices,

- the unary operation[11] $relab_h$ changes every vertex label a into $h(a)$ where h is a partial mapping : $C \to C$ (a label a is not modified if $h(a)$ is undefined).

- for each $a \in C$, the nullary symbol $\mathbf{a}(x)$ denotes the isolated vertex x ($x \in \mathcal{V}$) labelled by a.

Hence, $\tau(G \oplus H) = \tau(G) \cup \tau(H), \tau(add_{a,b}(G)) = \tau(\overrightarrow{add}_{a,b}(G)) = \tau(G)$, $\tau(relab_h(G)) = h'(\tau(G))$ where h' is the total mapping such that : $h'(a) :=$ `if` $h(a)$ *is defined* `then` $h(a)$ `else` a. We have $\tau(\mathbf{a}(x)) = \{a\}$.

(c) We denote by F_C the countable set of these operations. A term over F_C is *well-formed* if no two occurrences of nullary symbols denote the same vertex; in particular, the graphs defined by the two arguments of $\oplus$ are disjoint. We denote by $T(F_C)$ the set of well-formed terms, that we will call the *clique-width terms*. Each such term t denotes a C-graph $\boldsymbol{val}(t)$ whose vertices are those specified by the nullary symbols of t. Its *width* is the number of labels that occur in t.

Using a standard convention, we will denote in the same way a function symbol and the graph operation it defines. Hence, $relab_h(t)$ *is* a term if t is a term in $T(F_C)$, and $relab_h(G)$ *denotes* a C-graph if G denotes a C-graph.

(d) The *clique-width* of a C-graph[12] G, denoted by $cwd(G)$, is the least width of a term t such that $G \simeq \boldsymbol{val}(t)$. We denote by $cwd^*(G)$ the least width of a term t such that $G \equiv \boldsymbol{val}(t)$. Hence, $cwd(G) \leq cwd^*(G)$. Clearly, $cwd(G) = cwd^*(G')$ where G' is obtained from G by relabelling all its vertices in the same way.

(e) If G and H are not disjoint, we define $G \oplus H$ as the union of two disjoint isomorphic copies of G and H. The resulting C-graph is well-defined up to isomorphism, hence as an abstract C-graph. $\square$

Here are some examples (cf. [14]). The clique-width of a tree is at most 3, that of the clique K_n is 2 for $n \geq 2$. The undirected cycles C_3, C_4 have clique-width 2, C_5, C_6 have clique-width 3, and C_n has clique-width 4 for $n \geq 7$. For directed cycles $\overrightarrow{C}_n$, we have $cwd(\overrightarrow{C}_3) = 3$ and $cwd(\overrightarrow{C}_n) = 4$ if $n \geq 4$.

[10] As $\oplus$ is associative, we will write $t = t_1 \oplus t_2 \oplus ... \oplus t_n$ instead of $t_1 \oplus (t_2 \oplus (... \oplus t_n)...)$ or any equivalent writing. It is also comutative.

[11] If h modifies only one label, we call $relab_h$ an *elementary relabelling*. By using only elementary relabellings, one obtains the same notion of clique-width ([14], Proposition 2.118).

[12] In [14], we denote cwd^* by cwd.

Lemma 1.3 : For every C-graph G, we have :

$$\max\{|\tau(G)|, cwd(G)\} \leq cwd^*(G) \leq |\tau(G)| \cdot cwd(G).$$

Proof: The first inequality is clear from definitions. To prove the second one, we assume without loss of generality, that the type of G is $[p]$. Let H be G with all vertices labelled in the same way. Let C be the set of k labels of a term t that defines H. For each a in C and $i \in [p]$, we define a new label (a,i) that will only label the vertices x such that $\pi_G(x) = i$.

Consider in t a nullary symbol $\boldsymbol{a}(x)$. If $\pi_G(x) = i$, we replace it by $(\boldsymbol{a}, \boldsymbol{i})(x)$.

Each relabelling $relab_h$ is replaced by $relab_{h'}$ where h' maps (a,i) to (b,i) whenever h maps a to b. Similarly, we replace $add_{a,b}$ by the composition of the operations $add_{(a,i),(b,j)}$ for $i,j \in [p]$. We obtain in this way a term[13] t' over the set of labels $[p] \uplus (C \times [p])$. We let $h'' : C \times [p] \to [p]$ map (a,i) to i for $a \in C$ and $i \in [p]$. Then $G = relab_{h''}(\boldsymbol{val}(t')) = \boldsymbol{val}(relab_{h''}(t'))$. The term $relab_{h''}(t')$ uses at most $p(1+k)$ labels. However, we can fix some $a \in C$ and replace everywhere (a,i) by i, for each i. We obtain a term of width at most pk that defines G. $\square$

Hence if the type of G consists of p labels and k is the clique-width of the corresponding unlabelled graph, then one can define G, with its labelling, by a term with at most pk labels. This lemma implies that $cwd(G) \leq 2cwd(G-x)$ if x is a vertex of G. This bound is proved in [30].

Questions 1.4 : Can one improve the bounds[14] $cwd(G) \leq 2cwd(G-x)$ and $|\tau(G)| \cdot cwd(G)$ (of Lemma 1.3) ?

Definition 1.5 : *Abstract graphs*

We will also use nullary "generic" symbols $\mathbf{a}$ that do not denote any particular vertex. The vertex defined by an occurrence[15] u of $\mathbf{a}$ in a term t is u itself. We will also consider that a term written with such nullary symbols denotes an abstract C-graph (cf. Definition 1.2(a,e)). See [14], Section 2.52.

We will denote by $\overline{F}_C$ the signature F_C where each symbol $\mathbf{a}(x)$ is replaced by $\mathbf{a}$, and by $\overline{t}$ the term in $T(\overline{F}_C)$ obtained from a term $t \in T(F_C)$ by replacing each $\mathbf{a}(x)$ by $\mathbf{a}$. Then $\boldsymbol{val}(\overline{t}) \equiv \boldsymbol{val}(t)$.

Example 1.6 : *The grammar for cographs.*

We present *context-free graph grammars*, defined as *equation systems* whose unknowns are sets of *abstract graphs*, by using the example of cographs.

(1) One characterization of cographs is the following recursive one. Graphs are simple and undirected. The *join* of two disjoint undirected graphs G and

[13] The construction is similar for directed graphs and it needs no more labels.

[14] It not hard to prove that $lcwd(G) \leq lcwd(G-x)+2$ where $lcwd$ denotes the *linear clique-width*. This variant is defined by requiring that at least one of the two arguments of $\oplus$ is a nullary symbol. See *e.g.*, [14, 32].

[15] Occurrences in terms can be designated by Dewey words or by integers, cf. Definition 2.3 in [14],

H, denoted by $G \otimes H$, is defined as their union augmented with edges between any vertex of G and any vertex of H.

A *cograph* is either an isolated vertex, $G \oplus H$ or $G \otimes H$ for disjoint cographs G and H. Hence, the set $\mathcal{C}$ of abstract cographs is the least set (least for inclusion) that satisfies the recursive equation :

$$\mathcal{C} = \{*\} \cup (\mathcal{C} \oplus \mathcal{C}) \cup (\mathcal{C} \otimes \mathcal{C})$$

where $*$ denotes an isolated vertex (up to isomorphism), $\mathcal{D} \oplus \mathcal{E} := \{G \oplus H \mid G \in \mathcal{D}, H \in \mathcal{E}\}$ and similarly for $\otimes$.

We call such a description a *(context-free) graph grammar*.

Each cograph has thus a hierchical description, in terms of smaller cographs and the two operations $\oplus$ and $\otimes$. Hence, it is *defined by*, or more formally, is the *value of a term* in $T(\{\oplus, \otimes, *\})$, *i.e.,* a term written with $\oplus, \otimes$ and $*$. For example the term $t := ((* \oplus *) \oplus *) \otimes (* \oplus *)$ defines the complete bipartite graph $K_{3,2}$.

A fundamental property ([14], Proposition 3.23) states that the *same* recursive equation in sets of terms $X \subseteq T(\{\oplus, \otimes, *\})$, hence :

$$X = \{*\} \cup (X \oplus X) \cup (X \otimes X)$$

defines (by taking the least solution) *the terms representing cographs.* (If $Y, Z \subseteq T(\{\oplus, \otimes, *\})$, then $Y \oplus Z$ denotes the set of terms $t \oplus t'$ such that $t \in Y$ and $t' \in Z$.)

Actually, this equation defines the full set $T(\{\oplus, \otimes, *\})$. Cographs are the graphs defined by all terms in $T(\{\oplus, \otimes, *\})$. A cograph can be defined by several different terms, hence, this grammar is ambigous, which makes difficult its use for counting. However, an unambigous grammar can be used as we will see in Section 4.5.

The general notion of a grammar allows systems of mutually recursive equations that define sets of graphs or sets of terms. An example is :

$$\mathcal{D} = \{* \otimes *\} \cup (\mathcal{D} \oplus \mathcal{E}) \cup (\mathcal{E} \otimes \mathcal{E}),$$
$$\mathcal{E} = \{*\} \cup (\mathcal{D} \oplus *) \cup (\mathcal{E} \otimes \mathcal{E}).$$

The least sets $\mathcal{D}$ and $\mathcal{E}$ satisfying these equations are particular sets of cographs. The two sets of terms that form the least solution in $T(\{\oplus, \otimes, *\})$ of the (identical) system :

$$Y = \{* \otimes *\} \cup (Y \oplus Z) \cup (Z \otimes Z),$$
$$Z = \{*\} \cup (Y \oplus *) \cup (Z \otimes Z).$$

define the sets $\mathcal{D}$ and $\mathcal{E}$.

To simplify notation, we will write $*$ instead of $\{*\}$ and $* \otimes *$ instead of $\{* \otimes *\}$ in such equations, and similarly for terms without unknowns. The same letters X, Y, Z... will be used for sets of terms and the sets of graphs they denote.

(2) Systems of recursive set equations, written with set union and the extensions of functions to sets, make sense in any *F-algebra* $\mathbb{M} = (M, (f_{\mathbb{M}})_{f\in F})$ where M is a set equipped with functions $f_{\mathbb{M}}$ indexed by a functional signature F. Take for example F consisting of a, b, f, g, h of respective arities 0,0,1,2,3. Then a system of equations like :

$$\mathcal{D} = a \cup f(b) \cup f(\mathcal{D}) \cup g(\mathcal{E}, \mathcal{E}),$$
$$\mathcal{E} = b \cup h(\mathcal{D}, \mathcal{E}, \mathcal{E}) \cup g(\mathcal{E}, \mathcal{D}),$$

where $\mathcal{D}, \mathcal{E} \subseteq M$ has a least solution. Its least solution in subsets of $T(a, b, f, g, h)$ consists of two sets whose sets of *values* in $\mathbb{M}$ are $\mathcal{D}$ and $\mathcal{E}$.

Such sets are the *equational sets* of the algebra $\mathbb{M}$. This notion is relative to the *algebraic structure* specified by the operations $(f_{\mathbb{M}})_{f\in F}$, cf. [14], Chapter 3.

2 Substitution to vertices

In this section, we consider undirected graphs. We will adapt the definitions to directed graphs in Section 5.

Let C be a set of labels containing $\perp$. The vertices of a graph G labelled by $\perp$ will be said to be *dead* ; they form the set V_G^{dead}. The others, said to be *alive* form the set V_G^{live}. The unary operation κ, read *kill*, relabels all vertices by $\perp$, hence makes them dead.

Definition 2.1 : *Substitution.*

Let K be a C-graph and $x_1, \dots, x_p$ be pairwise distinct vertices. Let $H_1, \dots, H_p$ be pairwise disjoint C-graphs, that are disjoint[16], $n \geq 3$, from K. We define a C-graph $G := K[x_1 \leftarrow H_1, \dots, x_p \leftarrow H_p]$ as follows[17] :

$$V_G := (V_K - \{x_1, \dots, x_p\}) \uplus V_{H_1} \uplus \dots \uplus V_{H_p},$$
$$\pi_G(v) := \pi_K(v) \text{ if } v \in V_K - \{x_1, \dots, x_p\},$$
$$\pi_G(v) := \pi_K(x_i) \text{ if } v \in V_{H_i}^{live},$$
$$\pi_G(v) := \perp \text{ if } v \in V_{H_i}^{dead}.$$

Its edges are as follows, for u, v in V_G :

$uv \in E_G$ if and only if :

either $uv \in E_K$ and neither u nor v is in $\{x_1, \dots, x_p\}$,

or $uv \in E_{H_i}$ for some i,

or $u \in V_K$, $ux_i \in E_K$ and $v \in V_{H_i}^{live}$ (or vice-versa by exchanging u and v since we define undirected graphs, so that uv and vu designate the same edge),

or $u \in V_{H_i}^{live}$, $v \in V_{H_j}^{live}$ and $x_ix_j \in E_K$ (so that $i \neq j$).

[16] It is actually enough to assume that $(V_K - \{x_1, \dots, x_p\}) \cap (V_{H_1} \uplus \dots \uplus V_{H_p}) = \emptyset$.

[17] Read "H_i is substituted to x_i in K".

The type of G is thus that of K, possibly augmented with $\perp$ if some H_i has dead vertices : these vertices are dead in G. The labels of K have no influence on the definition of the edges of G, they only specify, together with the labels of the graphs H_i, those of the resulting graph G. The labels of the H_i's other than $\perp$ do not contribute to the labelling of G : if for each i, a mapping $h_i : C \to C$ satisfies $h_i(a) = \perp$ if and only if $a = \perp$, then :

$$K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p] = K[x_1 \leftarrow relab_{h_1}(H_1), \ldots, x_p \leftarrow relab_{h_p}(H_p)].$$

If all vertices of H_p are dead, then

$$K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p] = (K - x_p)[x_1 \leftarrow H_1, \ldots, x_{p-1} \leftarrow H_{p-1}]$$
$$\oplus H_p.$$

Because of dead vertices, this notion of substitution differs from the classical one, used in particular in the theory of modular decomposition (see the survey [31]). If $K, H_1, \ldots, H_p$ have no dead vertices, then $K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p]$ is the usual substitution, as in [14], Section 2.5.

Proposition 2.2 : Let K, H_1, H_2 be pairwise disjoint C-graphs and $x_1 \in V_K$.

(1) If x_2 is another vertex of K, then $K[x_1 \leftarrow H_1][x_2 \leftarrow H_2] = K[x_1 \leftarrow H_1, x_2 \leftarrow H_2]$.

(2) If $x_2 \in V_{H_1}$, then $K[x_1 \leftarrow H_1][x_2 \leftarrow H_2] = K[x_1 \leftarrow H_1[x_2 \leftarrow H_2]]$.

Proof : (1) Straightforward verification from the definitions.

(2) Let $G := K[x_1 \leftarrow H_1][x_2 \leftarrow H_2]$ and $G' := K[x_1 \leftarrow H_1[x_2 \leftarrow H_2]]$.Clearly, $V_G = V_{G'}$.

(2.1) We now compare edges. Let u, v belong to V_G. If u and v are both, either in V_K, or in V_{H_1} or in V_{H_2}, then $uv \in E_G$ if and only if $uv \in E_{G'}$. Otherwise we distinguish three cases.

(i) $u \in V_K$ and $v \in V_{H_1}$; then, $uv \in E_G$ if and only if $ux_1 \in E_K$ and v is live in H_1, if and only if $uv \in E_{G'}$.

(ii) $u \in V_K$ and $v \in V_{H_2}$; then, $uv \in E_G$ if and only if $ux_2 \in E_{K[x_1 \leftarrow H_1]}$ and v is live in H_2. The condition $ux_2 \in E_{K[x_1 \leftarrow H_1]}$ is equivalent to : $ux_1 \in E_K$ and x_2 is live in H_1.

Now $uv \in E_{G'}$ if and only if $ux_1 \in E_K$ and v is live in $H_1[x_2 \leftarrow H_2]$ which is true if and only if v is live in H_2 and x_2 is live in H_1. Hence, $uv \in E_G$ if and only if $uv \in E_{G'}$.

(iii) $u \in V_{H_1} - \{x_2\}$ and $v \in V_{H_2}$; then $uv \in E_G$ if and only if $ux_2 \in E_{H_1}$ and v is live in H_2, if and only if $uv \in E_{H_1[x_2 \leftarrow H_2]}$, if and only if $uv \in E_{G'}$.

Hence, G and G' have the same edges.

(2.2) It remains to verify that $\pi_G = \pi_{G'}$.

If $u \in (V_K - \{x_1\}) \uplus (V_{H_1} - \{x_2\})$, then $\pi_G(u) = \pi_{G'}(u)$ because u is not affected by the substitutions to x_2.

If $u \in V_{H_2}$, then $\pi_G(u) = \perp$ if u is dead in H_2; it is $\pi_{K[x_1 \leftarrow H_1]}(x_2)$ otherwise. We have $\pi_{K[x_1 \leftarrow H_1]}(x_2) = \perp$ if x_2 is dead in H_1, and otherwise, it is $\pi_K(x_1)$.

Now, $\pi_{G'}(u) = \perp$ if u is dead in $H_1[x_2 \leftarrow H_2]$ and it is $\pi_K(x_1)$ otherwise; observe that u is dead in $H_1[x_2 \leftarrow H_2]$ if and only if it is dead in H_2 or x_2 is dead in H_1. We obtain $\pi_G(u) = \pi_{G'}(u)$ in this case; this value is either $\pi_K(x_1)$ or $\perp$ (if u is dead in H_2 or x_2 is dead in H_1).

This completes the proof. □

Properties (1) and (2) are called respectively *commutativity* and *associativity of substitution* in [6]. They are axioms for the definition of *context-free graph grammars* based on an abstract notion of substitution and on equation systems, as explained in Example 1.6. These grammars are particular *vertex replacement grammars* [14].

Definition 2.3 : *Graph operations based on substitution.*

(a) For each C-graph K with vertex set enumerated as $\{x_1, \ldots, x_p\}$, we define as follows a p-ary graph operation[18] on C-graphs denoted by $\sigma[K, x_1, \ldots, x_p]$:

$$\sigma[K, x_1, \ldots, x_p](H_1, \ldots, H_p) := K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p] \qquad (1)$$

where $H_1, \ldots, H_p$ are pairwise disjoint C-graphs that are disjoint from K. Note that the vertex set of $\sigma[K, x_1, \ldots, x_p](H_1, \ldots, H_p)$ is $V_{H_1} \uplus \ldots \uplus V_{H_p}$.

If $H_1, \ldots, H_p$ are not pairwise disjoint, we replace them by isomorphic copies in a standard way (cf. Definitions 1.2(e) and 1.5, and Chapter 2 of [14]), so that $\sigma[K, x_1, \ldots, x_p]$ becomes a p-ary operation on abstract C-graphs.

(b) We denote by Σ_C the countable set of these operations together with the nullary symbols $\boldsymbol{a}(x)$ (this symbol denote the a-labelled vertex $x \in \mathcal{V}$, as in Definition 1.2(b)). A term $t \in T(\Sigma_C)$ is *well-formed* if each vertex x occurs at most once in some symbol $\boldsymbol{a}(x)$. It defines a C-graph $\boldsymbol{val}(t)$ where (1) is used to evaluate $\sigma[K, x_1, \ldots, x_p](H_1, \ldots, H_p)$.

(c) The signature $\overline{\Sigma}_C$ is obtained from Σ_C by replacing, for each a, each symbol $\boldsymbol{a}(x)$ by $\boldsymbol{a}$. As for clique-width terms in $T(\overline{F}_C)$, each term in $T(\overline{\Sigma}_C)$ denotes an abstract C-graph.

We denote by $relab_a$ the relabelling $relab_h$ such that $h(\perp) := \perp$ and $h(b) := a$ if $b \neq \perp$.

Proposition 2.4 : Let $t, t' \in T(\Sigma_C)$ and x be a vertex in the C-graph $\boldsymbol{val}(t)$ defined in t by $\boldsymbol{a}(x)$. Let t' be a term such that $V_{\boldsymbol{val}(t')} \cap (V_{\boldsymbol{val}(t)} - \{x\}) = \emptyset$. We have : $\boldsymbol{val}(t)[x \leftarrow \boldsymbol{val}(t')] = \boldsymbol{val}(t[relab_a(t')/\boldsymbol{a}(x)])$.

The C-graph $\boldsymbol{val}(t)[x \leftarrow \boldsymbol{val}(t')]$ is obtained by substituting in $\boldsymbol{val}(t)$ the C-graph $\boldsymbol{val}(t')$ to the vertex x. The term $t[relab_a(t')/\boldsymbol{a}(x)]$ is obtained by substituting in t the term $relab_a(t')$ to the unique occurrence of $\boldsymbol{a}(x)$. It is well-defined because $V_{\boldsymbol{val}(t')} \cap (V_{\boldsymbol{val}(t)} - \{x\}) = \emptyset$ (cf. the first footnote in Definition 2.1).

[18] We use the *same* notation $\sigma[K, v_1, \ldots, v_p]$ for the p-ary function symbol *and* the corresponding operation. Cf. Definition 1.2(c).

Proof : By induction on the structure of t.
If $t = \boldsymbol{a}(x)$, then $\boldsymbol{val}(t)[x \longleftarrow \boldsymbol{val}(t')] = \boldsymbol{a}(x)[x \longleftarrow \boldsymbol{val}(t')] = relab_a(\boldsymbol{val}(t'))$ $= \boldsymbol{val}(relab_a(t')) = \boldsymbol{val}(t[relab_a(t')/\boldsymbol{a}(x)])$.

Let now $t = \sigma[K, v_1, \dots, v_p](t_1, \dots, t_p)$. Without loss of generality and to simplify notation, we assume that $\boldsymbol{a}(x)$ occurs in t_1. Then, for every term s :

$t[s/\boldsymbol{a}(x)] = \sigma[K, v_1, \dots, v_p](t_1[s/\boldsymbol{a}(x)], t_2, \dots, t_p)$ and so
$\boldsymbol{val}(t[s/\boldsymbol{a}(x)]) =$
$$K[v_1 \leftarrow \boldsymbol{val}(t_1[s/\boldsymbol{a}(x)]), v_2 \leftarrow \boldsymbol{val}(t_2), \dots, v_p \leftarrow \boldsymbol{val}(t_p)]. \qquad (2)$$

By induction :

$$\boldsymbol{val}(t_1[relab_a(t')/\boldsymbol{a}(x)]) = \boldsymbol{val}(t_1)[x \leftarrow \boldsymbol{val}(t')],$$

hence, Equality (2) where $s = relab_a(t')$ yields

$\boldsymbol{val}(t[relab_a(t')/\boldsymbol{a}(x)]) = K[v_1 \leftarrow \boldsymbol{val}(t_1)[x \leftarrow \boldsymbol{val}(t')], \dots, v_p \leftarrow \boldsymbol{val}(t_p)]$
$= K[v_1 \leftarrow \boldsymbol{val}(t_1), \dots, v_p \leftarrow \boldsymbol{val}(t_p)][x \leftarrow \boldsymbol{val}(t')]$ by Propoposition 2.2(2),
$= \boldsymbol{val}(t)[x \leftarrow \boldsymbol{val}(t')]$. $\square$

By using these operations, one can define *graph grammars*, formalized by *systems of recursive equations* in sets of abstract C-graphs, of which one takes least solutions (cf. Example 1.6 and [14], Chapters 3 and 4).

Definitions 2.5 : *Some useful operations.*
Here are some operations on D-graphs, where $D := \{\bot, \top\}$ and $\top$ labels the live vertices. The first two equalities are mere observations.

$\kappa(H) = \sigma[K, x_1](H)$ where K consists of the dead vertex x_1.
$H_1 \oplus H_2 = \sigma[K, x_1, x_2](H_1, H_2)$ where K consists of two isolated live vertices x_1 and x_2.

We define :

$H_1 \otimes H_2 := \sigma[K, x_1, x_2](H_1, H_2)$ where K is the edge x_1x_2, and x_1 and x_2 are alive.
$\Lambda(H_1, H_2) := \sigma[K, x_1, x_2](H_1, H_2)$ where K is the edge x_1x_2, x_1 is alive and x_2 is dead.

The operations $\oplus$ and $\otimes$ are associative and commutative. Here are some other algebraic properties[19] :

[19] We leave as an open problem to find a complete set of equational axioms for $\oplus, \otimes, \Lambda, \kappa, \top$.

$$\kappa(G{\otimes}H) = \kappa(\Lambda(G,H)), \tag{3}$$
$$\kappa(G{\oplus}H) = \kappa(G){\oplus}\kappa(H) = \Lambda(\kappa(G),H) = \Lambda(\kappa(G),\kappa(H)), \tag{4}$$
$$\Lambda(G,H_1{\oplus}H_2) = \Lambda(\Lambda(G,H_1),H_2). \tag{5}$$

We let $\overline{\Sigma}_{dh}$ be the signature $\{\oplus,\otimes,\Lambda,\kappa,\top\} \subseteq \overline{\Sigma}_D$. For every vertex x, we have $\bot(x) = \kappa(\top(x))$. Hence, we need not put the nullary symbol $\bot$ in $\overline{\Sigma}_{dh}$. Actually, a vertex introduced by $\bot(x)$ is dead from the very beginning and is isolated in the defined graph.

Examples 2.6 : *Some grammars over* $\overline{\Sigma}_{dh}$.
Grammars are defined as equation systems that define sets of abstract D-graphs.

(1) The following equation for cographs that we have already seen in Example 1.6, can be solved in sets of abstract D-graphs :

$$X = \top \cup (X \oplus X) \cup (X \otimes X).$$

All vertices of the generated D-graphs are labelled by $\top$ because $\oplus$ and $\otimes$ do not introduce dead vertices. We recall that $X \oplus X := \{G \oplus H \mid G, H \in X\}$ if X is a set of labelled graphs and similarly for $\otimes$ and the other operations considered below.

As observed in Example 1.6, this equation can also be solved in $T(\{\oplus,\otimes,\top\})$. Its solution is a set of terms[20] that we will denote by $L(X)$. More generally, for a system of set equations over a functional signature F that has unknowns $X, Y, Z...$, we will denote by $L(X), L(Y), L(Z)...$ the associated sets of terms in $T(F)$. If the system is solved in an F-algebra $\mathbb{M}$, the corresponding sets of objects $X, Y, Z...$ are the sets of *values in* $\mathbb{M}$ of the terms in $L(X), L(Y), L(Z)....$

(2) We turn a *rooted tree* (cf. Section 1) into a $\{\top,\bot\}$-graph such that the root is the only live node.

These trees are defined recursively as follows : a unique node a is a tree with root a. If A and B are disjoint rooted trees with respective roots a and b, than one obtains a rooted tree C by taking the union of A and B, linked by an edge ab, and a is taken as root of C. Then, $C = \Lambda(A,B)$ if A, B, C are as above. The equation that defines the set R of rooted trees is thus :

$$R = \top \cup \Lambda(R,R).$$

Another grammar for trees, consisting of two equations is :

$$Y = \top \cup \Lambda(\top, Z),$$
$$Z = Y \cup (Z \oplus Z).$$

[20]We call *language* a set of terms, whence the notation $L(X)$. A set of graphs is *not called* a language, in order to have a coherent terminology.

Here, Z defines the nonempty disjoint unions of rooted trees.

(3) The set T of *(unrooted) trees* is defined by the equation $T = \kappa(R)$ or $T = \kappa(Y)$, with R, Y, Z defined as in (2). For an example, the tree with nodes u, v, w, x, y, z, root x and edges xy, xz, xv, zu and vw is defined by the term :

$$\Lambda(\Lambda(\Lambda(\top(x), \top(y)), \Lambda(\top(z), \top(u))), \Lambda(\top(v), \top(w)))$$

belonging to $L(R)$ or by the term :

$$\Lambda(\top(x), \top(y) \oplus \Lambda(\top(z), \top(u)) \oplus \Lambda(\top(v), \top(w)))$$

that belongs to $L(Y)$.

The paths with one live vertex at one end are defined by the equation

$$P = \top \cup \Lambda(\top, P).$$

(4) We will prove in the next proposition that the equation

$$W = \top \cup (W \oplus W) \cup (W \otimes W) \cup \Lambda(W, W)$$

defines, up to vertex labels, the distance-hereditary graphs (cf. Definition 1.1).

From these equations, we obtain that the rooted trees are defined by *all terms* in $T(\{\Lambda, \top\})$ or by *certain terms* in $T(\{\oplus, \Lambda, \top\})$, and that the distance-hereditary graphs are defined by terms in $T(\{\oplus, \otimes, \Lambda, \top\})$. In these equations and the generated terms, the label $\perp$ for dead vertices does not appear explicily, but it is introduced by the operations Λ and κ. $\square$

In the following description of distance-hereditary (DH) graphs, all vertices are defined as dead, equivalently, unlabelled. The following recursive definition of DH graphs has been established in [3], but we think interesting to prove it by using the concepts of the present article. We recall that equation systems always define abstract graphs.

Proposition 2.7 : (1) The distance-hereditary graphs form the set X defined by the two equations :

$$X = \kappa(W) \text{ and } W = \top \cup (W \oplus W) \cup (W \otimes W) \cup \Lambda(W, W).$$

(2) The connected distance-hereditary graphs form the set Y defined by the equation :

$$Y = \kappa(\top) \cup \kappa(W \otimes W)$$

and the equation of (1) that defines W.

Proof : (1) Note that[21] $L(W) = T(\{\oplus, \otimes, \Lambda, \top\})$. For both directions we will use the characterization of DH graphs recalled in Definition 1.1.

Claim 1 : Every DH graph G is in the set $\boldsymbol{val}(X) = \boldsymbol{val}(\kappa(W))$.

Proof : We use induction on the number n of vertices of G.

If $n = 1$, then G is a single dead vertex, hence $G \equiv \kappa(\top) \in \kappa(W)$.

Otherwise there are four cases.

(i) G is the disjoint union of two DH graphs H, H'. Then, $H \equiv \kappa(t), H' \equiv \kappa(t')$ for some $t, t' \in W = T(\{\oplus, \otimes, \Lambda, \top\})$. Hence, $G \equiv \kappa(t \oplus t')$ where $t \oplus t' \in L(W)$ because $\kappa(t \oplus t') \equiv \kappa(t) \oplus \kappa(t')$.

(ii) If G is obtained from a DH graph G' by adding a pendant vertex y to a vertex x of G', we have $G = G'[x \leftarrow H]$ where H is the edge xy, with x alive and y dead[22]; hence $H = \boldsymbol{val}(\Lambda(\top(x), \top(y)))$.

We have $G' = \kappa(\boldsymbol{val}(t'))$ where t' is a well-formed term over $\oplus, \otimes, \Lambda$ and the nullaries that define vertices. It has one occurrence of $\top(x)$.

We let $t := t'[\Lambda(\top(x), \top(y))/\top(x)]$. By Proposition 2.4, we have $\boldsymbol{val}(t) = \boldsymbol{val}(t'[\Lambda(\top(x), \top(y))/\top(x)]) = \boldsymbol{val}(t'[relab_\top(\Lambda(\top(x), \top(y)))/\top(x)])$

$\boldsymbol{val}(t'[x \leftarrow \boldsymbol{val}(relab_\top(\Lambda(\top(x), \top(y))))] = \boldsymbol{val}(t'[x \leftarrow H])$.

Hence $G = \kappa(t)$, so that $G \equiv \kappa(\overline{t})$ where $\overline{t} \in L(W)$ is obtained from t by replacing by $\top$ the symbols $\top(z)$ that define vertices.

(iii) Let G be obtained from a DH graph G' by adding a false twin y to a vertex x. We have $G = G'[x \leftarrow H]$ where H consists of two isolated live vertices x and y. Hence $H = \boldsymbol{val}(\top(x) \oplus \top(y))$. The proof continues as in (ii) with $\top(x) \oplus \top(y)$ instead of $\Lambda(\top(x), \top(y))$.

(iv) Let G be obtained from a DH graph G' by adding a true twin y to a vertex x. Here $G = G'[x \leftarrow H]$ where H consists of two live vertices x and y linked by an edge, hence $H = \boldsymbol{val}(\top(x) \otimes \top(y))$. The proof continues as in (iii) with $\top(x) \otimes \top(y)$ instead of $\top(x) \oplus \top(y)$.$\square$

Claim 2 : If $G = \boldsymbol{val}(t)$ for some well-formed term t over $\oplus, \otimes, \Lambda$ and the nullaries that define vertices, then $\kappa(G)$ is DH.

Proof : By induction on the size of t.

If $t = \top(x)$, the result holds because an isolated vertex is DH. Otherwise, we can find a position u in t such that t/u, the subterm of t issued from position u, is either $\Lambda(\top(x), \top(y))$, $\top(x) \oplus \top(y)$, or $\top(x) \otimes \top(y)$. Then $t = t'[(t/u)/\top(x)]$ for some well-formed term t' (t' is obtained by replacing in t the subterm t/u by $\top(x)$). By induction, $\kappa(\boldsymbol{val}(t'))$ is a DH graph G' and $G = G'[x \leftarrow H]$ by Proposition 2.4, where H is respectively as in cases (ii), (iii) or (iv).$\square$

(2) It is clear that a term t in $L(X)$ defines a connected graph if and only if it is not of the form $\kappa(t_1 \oplus t_2)$. Hence, the connected DH graphs can be defined by the equation :

[21] See Example 1.6 for the solution of equations in sets of terms. If t is a term, we will write $G \equiv\ t$ to indicate that $G\ \equiv \boldsymbol{val}(t)$.

[22] We use here the footnote in Definition 2.1.

$$Y = \kappa(\top) \cup \kappa(W \otimes W) \cup \kappa(\Lambda(W, W))$$

where W is as in (1). However, we observed in Definition 2.5 that $\kappa(\Lambda(G, H)) = \kappa(G \otimes H)$ for all D-graphs G and H. Hence, the term $\kappa(\Lambda(W, W))$ can be removed. $\square$

The *bipartite DH graphs* are built from isolated vertices by the addition of pendant edges and of false twins [1]. Hence, they form the set B defined by the two equations $B = \kappa(W')$ and $W' = \top \cup (W' \oplus W') \cup \Lambda(W', W')$.

3 Clique-width and substitution operations.

A *derived operation*[23] relative to an F-algebra $\mathbb{M}$ is defined by a term t in $T(F, \{u_1, ..., u_p\})$, *i.e.*, a term over F with *variables* (or *indeterminates*, *i.e*, nullary symbols to which values or terms can be substituted) $u_1, ..., u_p$. The corresponding p-ary function $t_{\mathbb{M}}$ is defined by evaluating t with p arguments from the domain of $\mathbb{M}$ as values of $u_1, ..., u_p$.

For an example using clique-width operations, the operation $\otimes$ on graphs of type $\{\top\}$ (cf. Definitions 1.6 and 2.5) satisfies the following equality for all D-graphs G, H :

$$G \otimes H = relab_{a \to \top}(add_{\top,a}(G \oplus relab_{\top \to a}(H))).$$

Hence, $\otimes$ is a derived operation defined by the term[24] $relab_{a\to\top}(add_{\perp,a}(u_1 \oplus relab_{\top\to a}(u_2)))$.

Our objective is to express the operations $\sigma[K, x_1, \dots, x_p]$ as derived operations over F_C, the signature upon which clique-width is based.

We let $Lin(F_C, \{u_q, ..., u_p\})$ be the set of terms in $T(F_C, \{u_q, ..., u_p\})$, $q \leq p$, where each variable u_i has a unique occurrence and no other nullary symbol occurs. Every such term defines a $(p-q+1)$-ary mapping on C-graphs denoted by $t_{\mathbb{G}}$. For pairwise disjoint graphs $H_q, \dots, H_p$, the vertex set of $t_{\mathbb{G}}(H_q, \dots, H_p)$ is $V_{H_q} \uplus \dots \uplus V_{H_p}$.

We define $T_{\perp}(F_C)$ as the set of terms that use none of the operations[25] $add_{a,\perp}$, $add_{\perp,a}$, $\overrightarrow{add}_{a,\perp}$, $\overrightarrow{add}_{\perp,a}$, $relab_h$ if $h(\perp) \neq \perp$, and no nullary symbol $\perp(x)$. We denote by $cwd^{\perp}(G)$ the minimal cardinality of $C - \{\perp\}$ such that[26] $G \equiv \boldsymbol{val}(t)$ for some term $t \in T_{\perp}(F_C)$. Clearly, $cwd(G) \leq cwd^{\perp}(G) + 1$. We have $cwd(T) = 3$ and $cwd^{\perp}(T) = 2$ for any tree T that is not a star.

Let K be a C-graph with vertex set $\{x_q, \dots, x_p\}$ defined by a term t in $T_{\perp}(F_C)$. Each vertex x_i occurs in a nullary symbol $\boldsymbol{a_i}(x_i)$ in t such that $a_i \neq \perp$. We define $\widehat{t} := t[u_q/\boldsymbol{a_q}(x_q), \dots, u_p/\boldsymbol{a_p}(x_p)] \in Lin(F_C, \{u_q, ..., u_p\})$.

[23] See, e.g. [14], Section 2.1.

[24] This term uses an auxiliary label $a \neq \perp, \top$. However, it defines graphs of type $\{\top\}$ from graphs of same type. The label a can be replaced by any other label different from $\top$ or $\perp$.

[25] These limitations on the use of $\perp$ make it an *inactive label* in [36].

[26] The equivalence $\equiv$ respects vertex labels, cf. Definition 1.2(a).

Lemma 3.1 : Let K be a C-graph with vertex set $\{x_1, \dots, x_p\}$ defined by $t \in T_{\perp}(F_C)$. Let $\widehat{t} := t[u_1/\boldsymbol{a_1}(x_1), \dots, u_p/\boldsymbol{a_p}(x_p)]$. For pairwise disjoint C-graphs $H_1, \dots, H_p$, we have:

$$\sigma[K, x_1, \dots, x_p](H_1, \dots, H_p) = \widehat{t}_{\mathbb{G}}(relab_{a_1}(H_1), \dots, relab_{a_p}(H_p)).$$

Proof: By induction on the structure of t.We recall that $relab_a$ is the relabelling that replaces by a every label except $\perp$.

If $t = \boldsymbol{a_1}(x_1)$, then $\widehat{t} = u_1$, K consists of the a_1-vertex x_1 and $\sigma[K, x_1](H_1) = relab_{a_1}(H_1) = \widehat{t}_{\mathbb{G}}(relab_{a_1}(H_1))$ (by the behaviour of labels in substitution).

If $t = t_1 \oplus t_2$, then, without loss of generality, we assume that the vertices of $K_1 := \boldsymbol{val}(t_1)$ are $x_1, \dots, x_i$ and those of $K_2 := \boldsymbol{val}(t_2)$ are $x_{i+1}, \dots, x_p$. We have $\widehat{t} = \widehat{t_1} \oplus \widehat{t_2}$. Then, since substitution distributes over disjoint union[27] and by induction :

$$\begin{aligned}
&\sigma[K, x_1, \dots, x_p](H_1, \dots, H_p) = \\
&\sigma[K_1, x_1, \dots, x_i](H_1, \dots, H_i) \oplus \sigma[K_2, x_{i+1}, \dots, x_p](H_{i+1}, \dots, H_p) = \\
&\widehat{t_1}_{\mathbb{G}}(relab_{a_1}(H_1), \dots, relab_{a_i}(H_i)) \oplus \widehat{t_2}_{\mathbb{G}}(relab_{a_i}(H_i), \dots, relab_{a_p}(H_p)) = \\
&\widehat{t}_{\mathbb{G}}(relab_{a_1}(H_1), \dots, relab_{a_p}(H_p)).
\end{aligned}$$

If $t = f(t_1)$ where f is $relab_h$ or $add_{a,b}$, then the result holds because, for every C-graph K with vertices $x_1, \dots, x_p$, we have :

$$\sigma[f(K), x_1, \dots, x_p](H_1, \dots, H_p) = f(\sigma[K, x_1, \dots, x_p](H_1, \dots, H_p)).$$

The equality to be proved follows then by induction. $\square$

We will denote by t_K and $\widehat{t_K}$ terms associated with K as above. We say that an operation $\sigma[K, x_1, \dots, x_p]$ has *width* k if $cwd^{\perp}(K) = k$. The operations $\oplus, \otimes, \Lambda$ and κ have respective widths 2,2,2 and 1.

Proposition 3.2 : If $G \equiv \boldsymbol{val}(s)$ for some term s in $T(\Sigma_C)$ whose operations have width at most k, then $cwd^{\perp}(G) \leq k$ and $cwd(G) \leq k+1$.

Proof : By induction on the structure of s, we define a term $\widetilde{s}$ in $T(F_C)$ such that $\boldsymbol{val}(\widetilde{s}) \equiv \boldsymbol{val}(s)$.

If $s = \boldsymbol{a}(w)$, then $\widetilde{s} := s$.

If $s = \sigma[K, x_1, \dots, x_p](s_1, \dots, s_p)$, then we define :

$\widetilde{s} := \widehat{t_K}[relab_{a_1}(\widetilde{s_1})/u_1, \dots, relab_{a_p}(\widetilde{s_p})/u_p]$.

It is clear that $\boldsymbol{val}(\widetilde{s}) \equiv \boldsymbol{val}(s)$.

The set of labels used in $\widetilde{s}$ is the set of all those used in the terms $\widehat{t_K}$ where $\sigma[K, x_1, \dots, x_p]$ occurs in s. We now bound $cwd^{\perp}(G)$. Without loss of generality, we can assume that all labels of the terms t_K for K occurring in s (via some $\sigma[K, x_1, \dots, x_p]$) are in a set C such that $C - \{\perp\}$ has cardinality k. Hence $cwd^{\perp}(G) \leq k$ and $cwd(G) \leq k+1$. $\square$

[27] This is clear from Definition 2.1.

If in s, all the operations of maximal width k do not use $\perp$ in their definitions by terms, then $cwd(G) = cwd^{\perp}(G) \leq k$.

Corollary 3.3 : Distance-hereditary graphs have clique-width at most 3
Proof : Distance-hereditary graphs are defined by means of the operations $\kappa, \Lambda, \oplus$ and $\otimes$ of width at most 2. $\square$

This result is known from [29, 37] with different proofs. As the operation Λ of width 2 needs $\perp$ in its defining term, we do not have $cwd(G) = cwd^{\perp}(G) \leq 2$. Proposition 4.9 of [36] establishes that conversely, G is distance-hereditary if $cwd^{\perp}(G) \leq 2$.

4 Split decomposition

The *split decomposition* of directed and undirected graphs has been defined and studied by Cunnigham in [18]. We will formulate it in terms of *graph-labelled trees,* as in [27, 28], also called *split-decomposition graphs* in [7]. We only consider undirected graphs in this section.

4.1 Definitions and basic facts

Definition 4.1 : *Graph-labelled trees and the graphs they describe.*

We denote by L_T the set of leaves of a tree T and by $Inc_T(v)$ the set of edges incident to a node v.

(a) A *graph-labelled tree*, denoted by $\mathcal{T}$, is a tree T with at least three nodes that is equipped, for each node $v \in N_T$, with a connected graph H_v, called a *component*, and a bijection $\rho_v{:}Inc_T(v) \to V_{H_v}$. The components are pairwise disjoint. We identify u and the unique vertex of H_u if u is a leaf.

Figure 1 shows a graph-labelled tree with leaves $1, ..., 8$ and internal nodes u, v, w, x. The components are surrounded by elipses. The dotted lines are the edges of the tree T. Each of them links two vertices of two different components, and each vertex x in a component H_v is incident to one and only one "dotted edge", namely, $\rho_v^{-1}(x)$.

(b) The corresponding *split-graph* $S(\mathcal{T})$ is the union of the components together with the edges $\rho_u(e)\rho_v(e)$ for $e = uv$, (the "dotted edges" of Figure 1). A path in $S(\mathcal{T})$ is *alternating* if no two consecutive edges are in a same component[28]. Between any two vertices x, y of $S(\mathcal{T})$, there is at most one alternating path. If there is one, we say that x is *accessible from y*, and we precise *through* z (or e) to indicate that this path goes through a particular vertex z (or edge e). For a vertex w of $S(\mathcal{T})$ belonging to a component H_u, we denote by $A(w)$

[28]Because of the bijections ρ_v, no two consecutive edges in a path can be tree-edges.

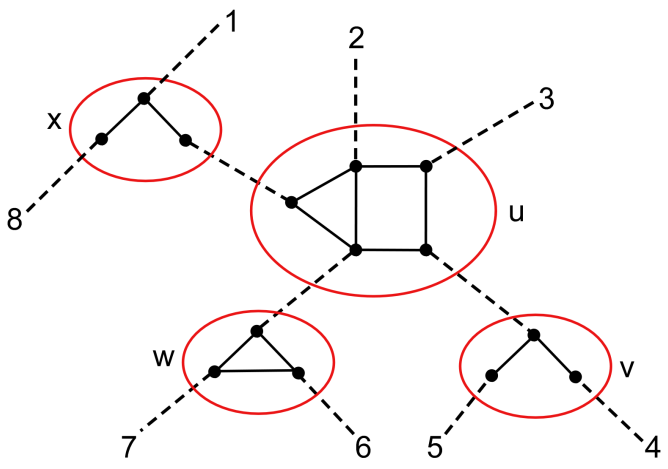

Figure 1: A graph-labelled tree $\mathcal{T}$ (cf. Definition 4.1(a).

(respectively by $P(w)$) the set of vertices accessible from w by a nonempty alternating path whose first edge is not in H_u (respectively, reachable from w by a nonempty path in $S(\mathcal{T})$ whose first edge is not in H_u).

(c) The *graph described by* $\mathcal{T}$, denoted by $G(\mathcal{T})$, has vertex set L_T and an edge uv if and only if u is accessible from v. It is connected ([27], Lemma 2.3) because the components are defined as connected[29].

We continue to examine the graph-labelled tree $\mathcal{T}$ of Figure 1 and the associated graph $G(\mathcal{T})$ in Figure 2. There is an alternating path between leaf 7 and leaf 1. Hence, they are adjacent vertices in $G(\mathcal{T})$. On any path between 3 and 7, there are two consecutive edges of H_u, hence, 7 is not adjacent to 3.

(d) Let $e = uv$ be an edge of T between two internal nodes. The *node-joining operation* (cf. [27]) contracts this edge, hence fuses u and v into a single new node say w, giving tree T'; the new component H'_w is defined as $H_u \uplus H_v$ minus the two vertices $\rho_u(e), \rho_v(e)$ and augmented with an edge between any vertex x in H_u and any vertex y in H_v such that $x\rho_u(e) \in E_{H_u}$ and $\rho_v(e)y \in E_{H_v}$. This graph is connected. We obtain a graph-labelled tree $\mathcal{T}'$ (nothing else is modified from $\mathcal{T}$) that describes the same graph. If $\rho_u(e)$ has degree r in H_u and $\rho_v(e)$ has degree s in H_s, the resulting component H'_w has a subgraph isomorphic to the complete bipartite graph $K_{r,s}$.

The opposite transformation is called *node-splitting*. It preserves also the defined graph. $\square$

Remark 4.2 : A graph G consisting of a single edge is defined by a graph-labelled tree one component of which is an edge. Otherwise, a single edge compo-

[29] If some components are not connected, the graph defined in this way is not connected. Actually, a disconnected graph is best described as the union of its connected components. Hence, we can require that components are connected.

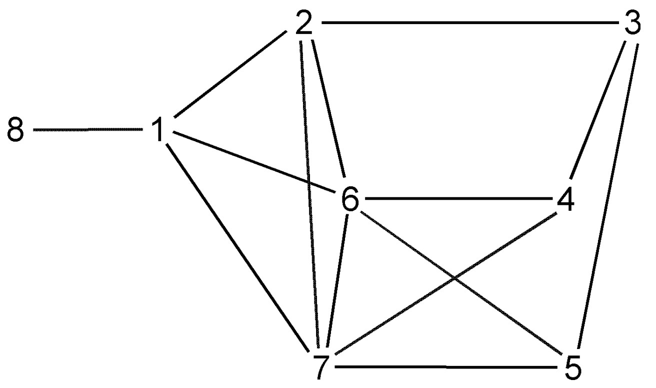

Figure 2: The graph $G(\mathcal{T})$ for $\mathcal{T}$ in Figure 1.

nent H_u can be eliminated by a node-joining of u with one of its two neighbours. This elimination being done as much as possible, the resulting tree has no node of degree 2.

However in a graph-labelled tree $\mathcal{T}$ that defines a graph with at least 2 vertices, it may be useful to insert a component consisting of a single edge (cf. Section 4.5 below) : consider an edge uv of T, replace it by two edges uw and wv where w is new node, and define the new component corresponding to w as consisting of a single edge.

The following lemma is implicit in [14, 27]. We will generalize it for directed graphs.

If uv is an edge of a tree T, we denote by $N_{T,u\backslash v}$ the set of nodes of the connected component of $T-u$ that contains v. Notation is in Definition 4.1(b).

Lemma 4.3 : Let $\mathcal{T}$ be a graph labelled tree and $G = G(\mathcal{T})$.

(1) For each vertex x of $S(\mathcal{T})$, the set $A(x) \cap L_T$ is not empty.

(2) Let x, y be distinct vertices of some component H. If xy is an edge of H, there is an alternating path between any leaf in $A(x)$ and any leaf of $A(y)$. Conversely, if an alternating path links a leaf in $A(x)$ and a leaf of $A(y)$, then this path goes through H, and more precisely, through x and y, and $xy \in E_H$. Each component is isomorphic to an induced subgraph of G.

(3) Let uv be an edge of T. There is an alternating path between any leaf in $A(x)$ where x is a neighbour of $\rho_u(uv)$ in H_u and any leaf in $A(y)$ where y is a neighbour of $\rho_v(uv)$ in H_v. Any such path goes through the edge $\rho_u(uv)\rho_v(uv)$.

Proof: (1) Let x belong to H_u. Then $x = \rho_u(uv)$ for some (unique) edge uv of T. We use induction on the cardinality of $N_{T,u\backslash v}$.

If $|N_{T,u\backslash v}| = 1$, then v is a leaf in $A(x)$ as $A(x) = \{v\} = \{\rho_v(uv)\}$.

Otherwise, H_v has an edge $\rho_v(uv)y$ and $y = \rho_v(vw)$ for some edge vw of T. Then $N_{T,v\backslash w} \subset N_{T,u\backslash v}$ and $A(y) \cap L_T \subseteq A(x) \cap L_T$. The set $A(y) \cap L_T$ is not empty by induction, so is $A(x) \cap L_T$.

(2) Let $xy = \rho_u(uv)\rho_u(uw)$ be an edge of component H_u. Let $z \in A(x) \cap L_T$ and $z' \in A(y) \cap L_T$. By connecting alternating paths between z and x, and z' and y with the edge xy, we get an alternating path between z and z'. Any such path must through xy as one checks from the definitions.

For each vertex x of H_u, let us choose a vertex $\widehat{x}$ of G in $A(x) \cap L_T$. By the previous observations, the induced subgraph of G whose vertices are the $\widehat{x}$'s is isomorphic to H_u.

(3) The proof is similar to that of (2). $\square$

The following corollary illustrates these notions in the basic case of trees.

Corollary 4.4 : Let $\mathcal{T}$ be a graph-labelled tree. The graph $G(\mathcal{T})$ is a tree if and only if :

(1) each component of $\mathcal{T}$ is a tree, and

(2) if $e = uv \in E_T$ is not a pendant edge, then at least one of $\rho_u(e)$ and/or $\rho_v(e)$ is a leaf of, respectively, H_u and/or H_v.

Proof : Assume $G(\mathcal{T})$ is a tree. By Lemma 4.3(2), each component of $\mathcal{T}$ is isomorphic to an induced subgraph of $G(\mathcal{T})$, hence is a tree since components are connected. For Property (2), if none of $\rho_u(e)$ and $\rho_v(e)$ is a leaf, then the node-joining of u and v (Definition 4.1(d)) merges H_u and H_v into a component that contains a complete bipartite graph $K_{r,s}$ such that $r, s \geq 2$. This component has a cycle and $G(\mathcal{T})$ is not a tree by (1).

Conversely, let $\mathcal{T}$ satisfy (1) and (2). Consider $e = uv$ satisfying Property (2) : if we apply to it the node-joining operation, the component H'_w created in this way is still a tree. The obtained graph-labelled tree $\mathcal{T}'$ satisfies (1) and (2) and describes $G(\mathcal{T})$. By repeating this operation until all edges are pendant, we obtain a graph-labelled tree that defines $G(\mathcal{T})$ and that has one component that is a tree, all others being leaves. Hence $G(\mathcal{T})$ is a tree. $\square$

A tree may be defined from several nonisomorphic graph-labelled trees. A stronger condition than (2) of Corollary 4.4, namely Condition (2) of Theorem 4.6, yields unicity, up to isomorphism, of the graph-labelled trees that define connected graphs.

Definition 4.5 : *Split decompositions.*

(1) A *split* of a graph G is a bipartition of V_G into two sets V_1 and V_2 having each at least 2 vertices, such that the edges between V_1 and V_2 induce a complete bipartite graph with at least one edge. These edges link any vertex of some set $A_1 \subseteq V_1$ and any vertex of some $A_2 \subseteq V_2$. Hence, G is $\kappa(G_1 \otimes G_2)$ where the

induced subgraphs $G_1 := G[V_1]$ and $G_2 := G[V_1]$ have labels in $D := \{\top, \bot\}$ in such a way that the vertices in $A_1 \cup A_2$ are labelled by $\top$ and the others by $\bot$.

(2) A graph is defined as *prime*[30] if it has at least 4 vertices and no split. The connected graphs with 3 vertices are the stars S_3 and the *triangles*, *i.e.*, the graphs isomorphic to K_3. A *star* S_n, $n \geq 3$, has a *center* and $n-1$ adjacent vertices that are leaves: S_n is a tree. Stars and cliques are not prime. No prime graph has less than 5 vertices. The cycles C_n for $n \geq 5$ are prime. □

Theorem 3 of [18], also proved as Theorem 2.9 in [27], states the following existence and unicity theorem.

Theorem 4.6 : Every connected graph with at least 3 vertices is $G(\mathcal{T})$ for a unique graph-labelled tree $\mathcal{T}$ such that :

(1) each component H_v is singleton, or prime, or is a clique K_n or a star S_n, for some $n \geq 3$,

(2) if $e = uv \in E_T$, then H_u and H_v are not both cliques, and, if they are both stars, then $\rho_u(e)$ and $\rho_v(e)$ are both centers or both leaves.

Unicity is understood up to isomorphism. □

Such a graph-labelled tree is called THE *split decomposition* of G. It is canonical because of its unicity, up to isomorphism. It can be obtained from an arbitrary graph-labelled tree that defines G by the node-splittings and the node-joinings of Definition 4.1(d) (see [27] for details). The resulting tree has no node of degree 2 (cf. Remark 4.2). We will also consider graph-labelled trees that need not be canonical.

Remarks 4.7 : (1) The split decomposition of a clique $K_n, n \geq 3$, has a unique component that is a clique. But any graph-labelled tree all components of which are cliques defines a clique. A clique $K_n, n \geq 3$ can be defined by a graph labelled tree all components of which are triangles or isolated vertices. By using node splitting, we can replace a component K_{r+s+2} where $r, s > 1$ by two components K_{r+1} and K_{s+1}.

(2) In the split decomposition of a tree, all components are stars, and if $e = uv \in E_T$, then $\rho_u(e)$ and $\rho_v(e)$ are both leaves of H_u and H_v by Corollary 4.5 and Theorem 4.7. Every tree with at least 3 nodes can be defined by a graph-labelled tree all components of which are stars S_3 or isolated vertices. As above, this can be achieved by node-splitting.

Examples 4.8 : *Distance-hereditary (DH) graphs and 3-leaf powers.*

A graph-labelled tree defines a distance-hereditary graph if and only if all its components are stars and cliques, as proved in [27], Section 3.1. This article also studies particular DH graphs called *3-leaf powers.* A 3-leaf power G is defined as follows from a tree S : V_G is L_S and two vertices are adjacent if and only if they are at distance at most 3 in S. A graph is a 3-leaf power if and only if it

[30] A different notion of prime graph is used in the theory of modular decomposition, cf. [31].

is obtained from a tree by substitutions of cliques to its vertices [2]. It follows that a graph is a 3-leaf power if and only if it is a clique or is $G(\mathcal{T})$ for some graph-labelled tree $\mathcal{T}$ having one component, say H_0, that is a tree whereas all others are cliques (an isolated vertex is a clique K_1). Hence, the set L of 3-leaf powers is defined, up to labels, by the two equations:

$$L = K \cup \Lambda(L, L),$$
$$K = \top \cup (K \otimes K).$$

The set K is that of cliques where all vertices are alive. The equation for L is derived from the first equation for rooted trees in Example 2.6(2). In the characterization of 3-leaf powers of [27], the component H_0 that is a tree is decomposed in the canonical way of Theorem 4.7. $\square$

4.2 Graph-labelled trees and substitution operations.

We will use D-graphs, where $D := \{\top, \bot\}$.

Definitions 4.9 : *Rooted graph-labelled trees and related notions.*

(a) Let $\mathcal{T}$ be a graph-labelled tree, with underlying tree T, not reduced to a single node. Let us select a node $r \in N_T$ and make it a root for T. We call then $\mathcal{T}$ a *rooted graph-labelled tree.* If r is a leaf, we call it a *leaf-rooted graph-labelled tree,* and otherwise, a *non-leaf-rooted graph-labelled tree.*

(b) If $u \in N_T$, we let[31] $N_u := \{x \in N_T \mid x \leq_T u\}$ and $V_u := \{x \in L_T \mid x \leq_T u\}$. Hence, $V_r = L_T = V_G$ and $V_u = \{u\}$ if $u \in L_T - \{r\}$.

(c) If $u \in N_T - L_T$ has father w, we say that $\rho_u(uw)$ is the *leader* of H_u, denoted by $\overline{u}$, and we define $H_u \backslash \overline{u}$ as the D-graph $H_u - \overline{u}$ (possibly not connected) where a vertex x is alive (labelled by $\top$) if it is adjacent to $\overline{u}$ in H_u and is dead (labelled by $\bot$) otherwise. We define also $H_r \backslash \overline{r} := H_r$, all its vertices being defined as dead. If r is a leaf, then $H_r \backslash \overline{r}$ is undefined.

(d) If $u \in N_T$, we define G_u as $G[V_u]$ labelled as follows :

(d.1) if $u = r$, then every vertex of $G_u = G$ is labelled by $\bot$,

(d.2) if $u \neq r$, a vertex y of G_u is labelled by $\top$ if it is in $A(z)$ for some neighbour z (in H_u) of the leader of H_u; otherwise, it is labelled by $\bot$.

Note that if r is a leaf and u is its neighbour in T, then $G = G_r = \kappa(\Lambda(\top(r), G_u)$. $\square$

[31] T will denote the rooted tree, without explicit mention of r ; in particular, $\leq_T$ depends on the choice of r.

For an example, consider the graph-labelled tree of Figure 1 in Section 4.1 with root x. The vertices 4 and 5 are alive in G_v, but not in G_u. The vertices 2,6,7 are alive in G_u.

The following lemma relates graph-labelled trees and substitution operations. Note that the graphs $H_u \backslash \overline{u}$ depend on the root r that is chosen for T.

Lemma 4.10 : Let $\mathcal{T}$ be a rooted graph-labelled tree that defines G. If u in $N_T - L_T$ has sons $u_1, ..., u_p$, and the corresponding p vertices of $H_u \backslash \overline{u}$ are $x_1, ..., x_p$ (that is, $x_i := \rho_u(uu_i)$), then we have :

$$G_u = (H_u \backslash \overline{u})[x_1 \leftarrow G_{u_1}, ..., x_p \leftarrow G_{u_p}].$$

Proof : Let $K := (H_u \backslash \overline{u})[x_1 \leftarrow G_{u_1}, ..., x_p \leftarrow G_{u_p}]$.

1) The vertex set of G_u is $V_u := \{x \in V_G \mid x \leq_T u\}$ $(V_G = L_T)$ hence, is the union the sets $V_{u_i} := \{x \in V_G \mid x \leq_T u_i\}$ that are the vertex sets of the graphs G_{u_i}. Hence, G_u and K have the same vertices.

2) As the graphs G_w are induced subgraphs of G, two vertices of G_{u_i} are adjacent in G_u if and only if they are in G as well as in G_{u_i}, hence also in K.

Consider vertices x of G_{u_i} and y of G_{u_j}, $j \neq i$. If they are adjacent in G_u, hence in G, they are linked by an alternating path, that must go through H_u, and not through its leader $\overline{u}$, and use its edge $\rho_u(u_i u)\rho_u(u_j u) = x_i x_j$, an edge of $H_u \backslash \overline{u}$. This path goes through the leader $\rho_{u_i}(u_i u)$ of H_{u_i}. Hence, x is alive in G_{u_i}. Similarly, y is alive in G_{u_j}. Hence xy is an edge of K.

Conversely, if $xy \in E_K$, then $x_i x_j$ is an edge of $H_u \backslash \overline{u}$, the vertex x is alive in G_{u_i} and y is alive in G_{u_j}. Going back to definitions, we have an alternating path between x and y. Hence, xy is a edge of G hence of G_u.

3) If u is the root r and is not a leaf, then all vertices of $H_u \backslash \overline{u}$ are dead, hence, so are those of K, as well as those of $G = G_r$.

Otherwise, let x be a vertex of G_{u_i} that is alive. Hence, there an alterning path P between x and the leader $\rho_{u_i}(u_i u)$ of H_{u_i}. If $\rho_u(u_i u) = x_i$ is a neighbour of the leader $\overline{u}$ of H_u, then x is alive in K. It is also in G_u because P can be extended into an alterning path from x to $\overline{u}$. If $\rho_u(u_i u) = x_i$ is not a neighbour of $\overline{u}$, then x is dead in K. It is also in G_u because P cannot be extended into an alterning path from x to $\overline{u}$.

If x is dead in G_{u_i}, then it is also in K and in G_u, because otherwise, the alternating path between x and $\overline{u}$ would give a path P as above. $\square$

4.3 From graph-labelled trees to grammars

If $\mathcal{M}$ is a finite set of connected (unlabelled) graphs having at least 2 vertices, we define $\mathcal{G}(\mathcal{M})$ as the set of graphs described by graph-labelled trees whose components are in $\mathcal{M}$. These graphs are connected (Definition 4.1(c)).

As before, $D := \{\top, \bot\}$. For each $H \in \mathcal{M}$, we define $H_\bot$ as H with all vertices labelled by $\bot$. We denote by $\Sigma_\mathcal{M}$ the set of operations $\sigma[H_\bot, x_1, ..., x_p]$ for $H \in \mathcal{M}$ and by $\Sigma'_\mathcal{M}$ the set of operations $\sigma[H\backslash x, x_1, ..., x_p]$ for $H \in \mathcal{M}$ and $x \in V_H$ (cf. Definition 4.9(c) for the notation $H\backslash x$).

Theorem 4.11 : If $\mathcal{M}$ is a finite set of connected graphs having at least 2 vertices, then $\mathcal{G}(\mathcal{M})$ is the set S defined by the two equations :

$$S = \kappa(\top) \cup \cup_{\sigma \in \Sigma_\mathcal{M}} \sigma(U, ..., U) \text{ and}$$
$$U = \top \cup \cup_{\sigma \in \Sigma'_\mathcal{M}} \sigma(U, ..., U).$$

Another grammar for $\mathcal{G}(\mathcal{M})$ consists of :

$$S' = \kappa(\top) \cup \kappa(\Lambda(\top, U))$$

and the above equation that defines U.

Proof : Let G belong to S. If it is a dead isolated vertex $\kappa(\top)$, it is in $\mathcal{G}(\mathcal{M})$. Otherwise, it is defined by a finite term $t = \sigma(t_1, ..., t_p)$ where $\sigma \in \Sigma_\mathcal{M}$ and $t_1, ..., t_p$ are terms in $T(\Sigma'_\mathcal{M} \cup \{\top\})$. By Lemma 4.10, this term represents a non-leaf-rooted graph-labelled tree $\mathcal{T}$. The component at the root is isomorphic to H such that $\sigma = \sigma[H, x_1, ..., x_p]$. The terms $t_1, ..., t_p$ represent the subtrees of T issued from the p sons of the root. Hence, G is described by a rooted graph-labelled tree with components in $\mathcal{M}$, and so, $G \in \mathcal{G}(\mathcal{M})$.

Let conversely $G \in \mathcal{G}(\mathcal{M})$. If it is a dead isolated vertex, then it is in S, defined by $\kappa(\top)$. Otherwise it is defined by a non-leaf rooted graph-labelled tree T, hence, by a term $t = \sigma(t_1, ..., t_p)$ as above. The subtrees of T issued from the sons of the root are defined by the terms $t_1, ..., t_p$.

The second grammar is based on leaf-rooted graph-labelled trees. The proof is similar, by the remark in Definition 4.9(d). $\square$

The equation $W = \top \cup \Lambda(\top, U)$ with U as above defines the *rooted graphs* in $\mathcal{G}(\mathcal{M})$ defined as those having exactly one live vertex (labelled by $\top$). We will apply this remark to DH graphs in Section 4.5.

4.4 Clique-width bounds from graph-labelled trees

Theorem 4.12 : Let G be a connected graph defined by a rooted graph-labelled tree $\mathcal{T}$ such that each operation $\sigma[H_u\backslash\overline{u}, x_1, ..., x_p]$ has width at most $k \geq 2$. Then $cwd^\bot(G) \leq k$ and $cwd(G) \leq k + 1$.

Proof: Immediate consequence of Proposition 3.2 and Theorem 4.11. $\square$

The next lemma gives an upper-bound to the widths of the terms in $T_\bot(F_C)$ that define the D-graphs $H_u\backslash\overline{u}$.

Lemma 4.13 : Let H be a D-graph such that $cwd(H) = k$. Then $cwd^{\perp}(H) \leq k + \min\{k, \left|V_H^{live}\right|, \left|V_H^{dead}\right|\}$.

Proof: We have $cwd^{\perp}(H) \leq cwd^{*}(H) \leq 2k$ by Lemma 1.3 (with $|\tau(G)| \leq 2$).

For proving that $cwd^{\perp}(H) \leq k + \left|V_H^{live}\right|$, we consider a term that defines H with a set C' of k labels different from $\perp$. Assume that $V_H^{live} = \{x_1, ..., x_p\}$. We add new labels $c_1, ..., c_p$ to C' so that c_i will only label x_i. We transform t into t' accordingly. In particular, to take a typical case, if at some position in t the operation $add_{a,b}$ adds edges between x_i, labelled at this point by a (there may be other a-labelled vertices) and b-labelled vertices, then we replace it by $add_{c_i,b} \circ add_{a,b}$.

Hence, we can use $p + k$ labels different from $\perp$.

If $V_H^{dead} = \{x_1, ..., x_p\}$. We do a similar construction. $\square$

Theorem 4.14 : Let G be defined by a graph-labelled tree $\mathcal{T}$ whose components have maximal clique-width m and maximal degree d, then $m \leq cwd(G) \leq m + \min\{m, d\} + 1 \leq 2m + 1$.

Proof : The inequality $m \leq cwd(G)$ follows from Lemma 4.3(2) since clique-width is monotone with respect to the induced subgraph relation.

We now prove the other inequality[32]. For each component H_u, the number of live vertices in $H_u \backslash \overline{u}$ is at most d. Hence, Lemma 4.13 gives $cwd^{\perp}(H_u \backslash \overline{u}) \leq m + \min\{m, d\}$ and Theorem 4.12 gives $cwd^{\perp}(G) \leq m + \min\{m, d\}$, so that $cwd(G) \leq m + \min\{m, d\} + 1 \leq 2m + 1$. $\square$

For an example, if G is a tree that is not a star and is defined by $\mathcal{T}$ whose components are stars with three nodes, hence of clique-width 2, we have $m = 2$ and $cwd(G) = 3$.

Remark 4.15 : *A bound based on rank-width.*

Rank-width is another graph complexity measure initially defined for undirected graphs[33], denoted by rwd. It is based on ternary trees (without root) that define *layouts* of the considered graphs. The DH graphs are those of rank-width 1, as proved in [37].

Rank-width is related to clique-width by the inequalities $rwd(G) \leq cwd(G) \leq 2^{rwd(G)+1} - 1$, cf. [37]. Furthermore, if $G = G(\mathcal{T})$ for some graph-labelled tree $\mathcal{T}$, and m is the maximal rank-width of a component H_u, then $rwd(G) = m$ (Theorem 4.3 of [33]). Hence, if $cwd(H_u) \leq m$ for all u, we get $rwd(G) \leq m$ and $cwd(G) \leq 2^{m+1} - 1$.

Example 4.16 : *Parity graphs.*

[32] By using monadic second-order transductions [7] proves that $cwd(G)$ is bounded in terms of m by a superexponential function.

[33] The extension to directed graphs is in [34].

A graph is a *parity graph* if for any two vertices, the induced paths joining them have the same parity. Bipartite graphs and DH graphs are parity graphs. The article [5] establishes that the parity graphs are the graphs having a split decomposition whose components are cliques and bipartite graphs. We do not obtain a finite grammar as bipartite graphs, whence also parity graphs, have unbounded clique-width.

4.5 Unambigous grammars for cographs and distance-hereditary graphs

We first examine some of the operations $\sigma[H, x_1, ..., x_p]$ that arise in split decompositions.

Observation 4.17 : *Substitution operations related to split decompositions.*
Case 1: H is a clique $K_p, p \geq 2$ whose vertices $x_1, ..., x_p$ are all alive. We have :

$$\sigma[K_p, x_1, ..., x_p](G_1, ..., G_p) = G_1 \otimes ... \otimes G_p.$$

Case 2 : $H = S_p \backslash x_p$ where $p \geq 3$ and S_p has center x_1 that is alive, all other vertices being dead. We have :

$$\sigma[S_p \backslash x_p, x_1, ..., x_{p-1}](G_1, ..., G_{p-1}) = \Lambda(\Lambda(...\Lambda(G_1, G_2), G_3), ..., G_{p-1}))...))$$
$$= \Lambda(G_1, G_2 \oplus G_3 \oplus ... \oplus G_{p-1}) \text{ by Equality (5) of Definition 2.5.}$$

Case 3 : $H = S_p \backslash x_p$ where $p \geq 3$, S_p has center x_p and all vertices of $S_p \backslash x_p$ are alive. Then we have :

$$\sigma[S_p \backslash x_p, x_1, ..., x_{p-1}](G_1, ..., G_{p-1}) = G_1 \oplus ... \oplus G_{p-1}.$$

Constructions 4.18 : *Grammars for DH graphs revisited.*
A connected DH graph without live vertices is defined by a graph-labelled tree $\mathcal{T}$ whose components are stars, cliques and single vertices. By rooting T at a leaf, we obtain from Theorem 4.11 the following grammar, where S defines the connected DH graphs :

$$S = \kappa(\top) \cup \kappa(\Lambda(\top, U)),$$
$$U = \top \cup (U \oplus U) \cup (U \otimes U) \cup \Lambda(U, U).$$

Here is an alternative construction, where the chosen root is not a leaf. By means of node splittings (Definition 4.1(d)), we can transform $\mathcal{T}$ as above into a graph-labelled tree whose components are stars S_3, triangles K_3, together with one component[34] K_2 : for $n \geq 4$, a component (isomorphic to) K_n can be split

[34] By Remark 4.2.

into K_3 and K_{n-1}, and a component S_n can be split into S_3 and S_{n-1} where the center of S_3 is linked to a leaf of S_{n-1}.

Let us take as root the component K_2. We obtain from Theorem 4.11 the following equations :

$$S = \kappa(\top) \cup \kappa(U \otimes U),$$
$$U = \top \cup (U \oplus U) \cup (U \otimes U) \cup \Lambda(U, U),$$

that are the two equations of Proposition 2.7(2).

The equations of Proposition 2.7 can be used to generate DH graphs having a given number n of vertices, but not, at least immediately, with equal probability for each fixed n. The reason is that because of the associativity and commuttivity of $\oplus$ and $\otimes$, and also because of Equality (5) of Definition 2.5, this grammar is ambigous. For the same reason, it cannot be used for counting[35] the number of DH graphs having n vertices. However, it can be transformed so as to allow that.

Construction 4.19 : *An unambigous grammar for cographs.*
Cographs are DH and defined by the equation :

$$C = \top \cup (C \oplus C) \cup C \otimes C.$$

They have a canonical description as follows, where $C_\otimes$ (resp. $C_\oplus$) denotes the set of connected (resp. disconnected) cographs with at least two vertices. A ($\top$-labelled, abstract) cograph G is :

a single vertex $\top$,

or it is connected and of the form $G_1 \otimes ... \otimes G_p$, $p \geq 2$, where $G_1, ..., G_p \in C_\oplus \uplus \{\top\}$,

or it is disconnected and of the form $G_1 \oplus ... \oplus G_p$, $p \geq 2$, where $G_1, ..., G_p \in C_\otimes \uplus \{\top\}$.

The corresponding term is (or represents) the (canonical) modular decomposition of G [31]. For a finite or countable set $\mathcal{L}$ of labelled graphs, we define:

$\oplus_{\geq 2}(\mathcal{L})$ as the set of labelled graphs $G_1 \oplus ... \oplus G_p$ and

$\otimes_{\geq 2}(\mathcal{L})$ as the set of labelled graphs $G_1 \otimes ... \otimes G_p$,

where, in both cases, $p \geq 2$ and $G_1, ..., G_p \in \mathcal{L}$.

With these "metaoperations", we can define cographs by the three equations:

$$C = \top \cup C_\oplus \cup C_\otimes,$$
$$C_\oplus = \oplus_{\geq 2}(\top \cup C_\otimes),$$
$$C_\otimes = \otimes_{\geq 2}(\top \cup C_\oplus).$$

[35] The term *enumerating* creates confusion with the problem of *listing* graphs or configurations in graphs.

These three equations form an *unambigous grammar* because of the unicity of the decomposition recalled above, *and* because $G_1 \oplus ... \oplus G_p = G_{\pi(1)} \oplus ... \oplus G_{\pi(p)}$ for any permutation π of the indices, and similarly for $\otimes$. Hence, $\oplus_{\geq 2}$ can be seen as an operation of variable arity extended to sets. One obtains a bijection of the set connected (abstract) cographs with the rooted trees whose internal nodes have at least two sons. The sons of a node form a set and not a sequence. These trees, called *hierarchies*, have been used in [38] to evaluate the number of cographs of a given size.

Construction 4.20 : *An unambigous grammar for DH graphs.*

For distance-hereditary graphs, the situation is similar, by using canonical split decompositions. The grammars given in Construction 4.18 are ambigous. We can obtain an unambigous one for *rooted and connected DH graphs*. Rooted means that one vertex is distinguished as alive and all others are dead. We only generate such graphs having at least 3 vertices.

Theorem 4.6 and Example 4.8 yield the following description that we give as a grammar written with the two above metaoperations:

$$\begin{aligned}
&D = \Lambda(\top, D_\otimes \cup D_\oplus \cup D_\Lambda),\\
&D_\otimes = \otimes_{\geq 2}(\top \cup D_\oplus \cup D_\Lambda),\\
&D_\oplus = \oplus_{\geq 2}(\top \cup D_\otimes \cup D_\Lambda),\\
&D_\Lambda = \Lambda(J, D_\oplus) \cup \Lambda(J, \top \cup D_\otimes \cup D_\Lambda),\\
&J = \top \cup D_\oplus \cup D_\otimes.
\end{aligned}$$

The equation $D = \Lambda(\top, D_\otimes \cup D_\oplus \cup D_\Lambda)$ defines the root by $\top$ and $D_\otimes \cup D_\oplus \cup D_\Lambda$ correspond to the three types of components H_u of its son u, respectively Cases 1,3 and 2 in Observations 4.17.

The equation $D_\otimes = \otimes_{\geq 2}(\top \cup D_\oplus \cup D_\Lambda)$ corresponds to a component that is a clique (cf. Case 1). The righthand side does not include $D_\otimes$ because two clique components cannot be neighbour in a canonical split decomposition.

The equation $D_\oplus = \oplus_{\geq 2}(\top \cup D_\otimes \cup D_\Lambda)$ corresponds to a star component whose center is the leader. The righthand side does not include $D_\oplus$ because two star components cannot be linked by their centers.

The rules for D_Λ correspond a star component H_u whose center is not the leader. The set J corresponds to the son of u linked to the center. It does not does not include D_Λ because two star components cannot be linked by two leaves. The term $\Lambda(J, \top \cup D_\otimes \cup D_\Lambda)$ corresponds to a star $S_3 = H_u$, and the term $\Lambda(J, D_\oplus)$ to larger stars.

In this grammar, an equation like $D_\otimes = \otimes_{\geq 2}(\top \cup D_\oplus \cup D_\Lambda)$ creates no ambiguity because the three sets $\{\top\}$, $D_\oplus$ and D_Λ are disjoint and the metaoperation $\otimes_{\geq 2}$ avoids the ambiguities created by the associativity and commutativity of $\otimes$. That the full grammar is unambigous follows from the unicity part of Theorem 4.6.

These rules appear in Appendix A of [4], written so as to yield corresponding generating functions. We recall that unambiguity is essential for counting purposes, and also for random generation (cf. [25]).

Distance-hereditary graphs without root (without any vertex distinguished as the unique live one) are obtained by adding the rule $E = \kappa(D)$, but the resulting grammar becomes ambigous because the derivation trees corresponding to different roots are different. A more complex grammar that is appropriate for counting the DH graphs without root is given in [4]. It uses rules with substractions so as to avoid double counting. (They apply the equality $|A \cup B| = |A| + |B| - |A \cap B|$.) This article also handles 3-leaf powers (cf. Example 4.8) in a similar way.

5 Directed graphs

We now extend our results to directed graphs. Cunnigham defines a *canonical split decomposition* for the directed graphs that are strongly connected (Theorem 1 in [18]). An undirected graph can be seen as a directed one where each *arc* (directed edge) has an opposite one. It is connected if and only if the corresponding directed graph is strongly connected. Hence, Theorem 4.7 is a special case of a more general one for directed graphs[36].

We will use *graph-labelled trees* and *split decomposition graphs* as in [7]. In order to extend to directed graphs the results of Section 4, we will revise the notion of substitution of Section 2 for graphs with labels that encode the directions of arcs. The set of live vertices is partitioned into three sets, designated by the tags $\top, +, -$, attached to the labels of the sets C used to construct graphs with the clique-width operations of Definition 1.2.

We study directed graphs. However, the graphs representing graph-labelled trees will have undirected edges as well as arcs.

5.1 Substitution

Definition 5.1 : *Substitutions of directed graphs to vertices*

We let $D := \{\bot, +, -, \top\}$ be ordered in such a way that $\bot < + < \top$, $\bot < - < \top$, and $+$ and $-$ are incomparable. Let K be a D-graph with vertex set $\{x_1, \dots, x_p\}$ and $H_1, \dots, H_p$ be pairwise disjoint D-graphs, that are disjoint from K. We define a D-graph $G := K[x_1 \leftarrow H_1, \dots, x_p \leftarrow H_p]$:

$$V_G := V_{H_1} \uplus \dots \uplus V_{H_p},$$
$$\pi_G(v) := \inf\{\pi_{H_i}(v), \pi_K(x_i)\} \text{ if } v \in V_{H_i}.$$

Its arcs are as follows, for $u, v \in V_G$ (they do not depend on π_K) :

[36] We thought better to begin with undirected graphs, because the formal setting is much simpler and most of Graph Structure Theory and Graph Algorithmics is devoted to undirected graphs.

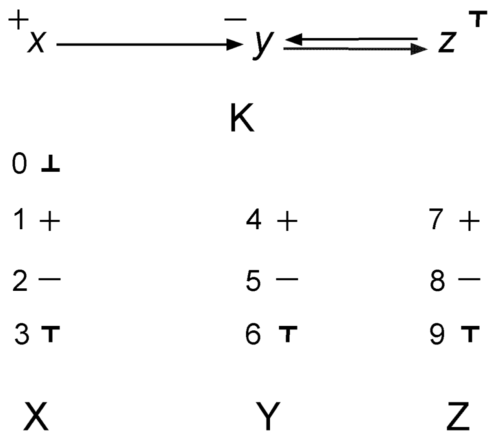

Figure 3: Graphs from Example 5.2

$uv \in E_G$ if and only if :

$uv \in E_{H_i}$ for some i,

or $\pi_{H_i}(u) \in \{\top, +\}$, $\pi_{H_j}(v) \in \{\top, -\}$ and $x_i x_j \in E_K$ (and so $i \neq j$).

Lemma 5.13 below motivates this definition. If we consider an undirected graph as a directed graph where each arc has an opposite one, and whose vertices are labelled by $\perp$ or $\top$, then, Definition 5.1 gives the same notion of substitution as Definition 2.1.

Example 5.2 : Let K have vertices x, y and z, respectively labelled by $+, -$ and $\top$, and arcs xy, yz and zy. Let X be the edgeless graph with vertices 0,1,2,3 labelled respectively by $\perp, +, -, \top$. Let similarly Y have vertices 4,5,6 labelled by $+, -, \top$ and Z, vertices 7,8,9 labelled by $+, -, \top$. The graphs K, X, Y, Z and $G := K[x \leftarrow X, y \leftarrow Y, z \leftarrow Z]$ are shown in Figures 3 and 4. The labels of vertices 0,1,5,7,8 and 9 are as in X, Y and Z. Those of 2,3,4 and 6 are respectively $\perp = \inf\{+, -\}$, $+ = \inf\{+, \top\}$, $\perp = \inf\{+, -\}$ and $- = \inf\{-, \top\}$, cf. Definition 5.1(a). □

We obtain graph operations, as in Section 2, that we will use to describe the directed graphs defined by graph-labelled trees.

Definition 5.3 : *Graph operations based on substitution.*
For each D-graph K with vertex set enumerated as $\{x_1, \ldots, x_p\}$, we define as follows a p-ary operation on D-graphs denoted by $\sigma[K, x_1, \ldots, x_p]$:

$$\sigma[K, x_1, \ldots, x_p](H_1, \ldots, H_p) := K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p]$$

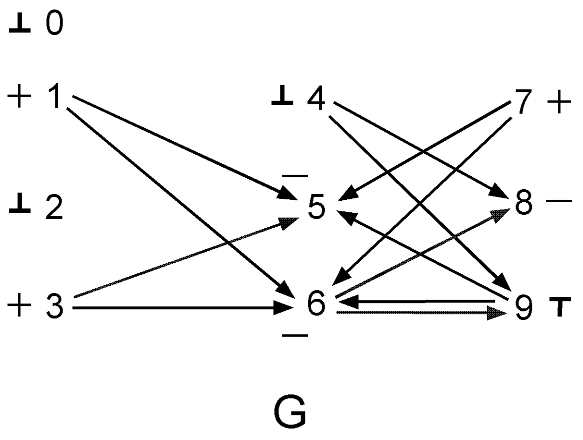

Figure 4: Graph G of Example 5.2.

where $H_1, \dots, H_p$ are pairwise disjoint and disjoint from K. If they are not, we replace them by isomorphic copies, so that $\sigma[K, x_1, \dots, x_p]$ becomes a p-ary operation on abstract D-graphs. □

If we extend Definition 5.1, so that K can have other vertices than $x_1, \dots, x_p$, then Proposition 2.2 is still valid.

We will express substitutions in terms of clique-width operations. For directed graphs, we use the operations $\overrightarrow{add}_{a,b}$ that create arcs from a-labelled vertices to b-labelled ones. Our objective is to bound the clique-width of $G := K[x_1 \leftarrow H_1, \dots, x_p \leftarrow H_p]$ as $O(\max\{cwd(K), cwd(H_1), \dots, cwd(H_p)\})$.

Definitions 5.4 : *Clique-width operations and substitution.*
The set $D := \{\bot, \top, +, -\}$ is ordered by Definition 5.1. Let $K, H_1, \dots, H_p$ be D-graphs and $G := K[x_1 \leftarrow H_1, \dots, x_p \leftarrow H_p]$.

We assume that the graphs $K, H_1, \dots, H_p$ are defined, up to their labels in D, by terms in $T(F_C)$ where $C \cap D = \emptyset$. We define $C^\# := \{(a, \alpha, \beta) \mid a \in C, \alpha, \beta \in D, \beta \leq \alpha\}$ and $f : C^\# \rightarrow D$ such that $f((a, \alpha, \beta)) := \beta$ for all (a, α, β) in $C^\#$.

We will define the D-graph G by a term t_G in $T(F_{C^\#})$ in such a way that $G = relab_f(\boldsymbol{val}(t_G))$.

Assume that $K = \boldsymbol{val}(t_K)$ for $t_K \in T(F_C)$. Let $x_1, \dots, x_p$ be the vertices of K. Each vertex x_i is defined by a nullary symbol $\boldsymbol{a_i}(x_i)$ in t_K. We construct a term $\widehat{t_K}$ in $T(F_{C^\#}, \{u_1, \dots, u_p\})$ as follows :

- we replace each $\boldsymbol{a_i}(x_i)$ by the variable u_i,

- we replace each operation $relab_h$ by $relab_{\widehat{h}}$ where $\widehat{h}((a, \alpha, \beta)) := (h(a), \alpha, \beta)$ for all $(a, \alpha, \beta) \in C^\#$,

- we replace each operation $\overrightarrow{add}_{a,b}$ by the composition (in any order) of the operations $\overrightarrow{add}_{(a,\alpha,\beta),(b,\alpha',\beta')}$ such that $\alpha \in \{\top, +\}, \alpha' \in \{\top, -\}$ and $\beta, \beta' \in D$, $\beta \leq \alpha, \beta' \leq \alpha'$.

Furthermore, for each $i := 1, ..., p$, we define $h_i : C^{\#} \to C^{\#}$ by $h_i((a, \alpha, \beta)) := (a_i, \beta, \inf\{\beta, \pi_K(x_i)\})$ for $(a, \alpha, \beta) \in C^{\#}$.□

Lemma 5.5 : Let $K, H_1, \ldots, H_p$,G, t_K and $\widehat{t_K}$ be as in Definition 5.4. Assume that for each i, we have a term $t_i \in T(F_{C^{\#}})$ such that $H_i = relab_f(\boldsymbol{val}(t_i))$. Then :

$K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p] = relab_f(\boldsymbol{val}(\widehat{t_K}[relab_{h_1}(t_1)/u_1, ..., relab_{h_p}(t_p)/u_p]))$.

Proof : Let

$G := K[x_1 \leftarrow H_1, \ldots, x_p \leftarrow H_p]$ and

$G' := relab_f(\boldsymbol{val}(\widehat{t_K}[relab_{h_1}(t_1)/u_1, ..., relab_{h_p}(t_p)/u_p]))$.

These two graphs have the same vertex set, $V_{H_1} \uplus \ldots \uplus V_{H_p}$. We compare their labels.

Let $u \in V_{H_i}$ and (a, α, β) be its label in $\boldsymbol{val}(t_i)$. Its label is β in H_i, and it is $(a_i, \beta, \inf\{\beta, \pi_K(x_i)\})$ in $relab_{h_i}(t_i)$. Its label in G' is thus $\inf\{\beta, \pi_K(x_i)\}$ because the relabellings in $\widehat{t_K}$ do not modify the third components of labels. It is the same in G by Definition 5.1.

We now compare the arcs of G and G'.

Case 1 : $u, v \in V_{H_i}$. If $uv \in E_{H_i}$, it is also an arc of G and of G'. If $uv \notin E_{H_i}$, it is not an arc of G either. And it is not an arc of G' because the labels of u and v have the same first components, namely a_i, in $relab_{h_i}(t_i)$ and the relabellings in $\widehat{t_K}$ maintain this equality. Hence, no arc between u and v is created by the operations $\overrightarrow{add}_{(a,\alpha,\beta),(b,\alpha',\beta')}$ of $\widehat{t_K}$ (where we must have $a \neq b$). Hence, $uv \notin E_{G'}$.

Case 2 : $u \in V_{H_i}, v \in V_{H_j}$, $i \neq j$. If $uv \in E_G$, then $x_i x_j$ is an arc of K and the labels of u and v in H_i and H_j are respectively $\alpha \in \{\top, +\}$ and $\alpha' \in \{\top, -\}$ by Definition 5.1. Let us now consider G'. At some position p in t_K the arc $x_i x_j$ is created by an operation $\overrightarrow{add}_{a,b}$.

In $\boldsymbol{val}(relab_{h_i}(t_i))$ and $\boldsymbol{val}(relab_{h_j}(t_j))$, the labels of u and v are respectively (a_i, α, β) and (a_j, α', β') for some β and β', and a_i and a_j are relabelled in t_K into a and b at position p. The labels of u and v are thus (a, α, β) and (b, α', β') in $\widehat{t_K}$ at the place corresponding to p in the construction of $\widehat{t_K}$ from t_K ($\overrightarrow{add}_{a,b}$ is replaced by a composition of arc additions). Hence, uv is an arc of G', created by $\overrightarrow{add}_{(a,\alpha,\beta),(b,\alpha',\beta')}$.

Hence every arc of G is one of G'. The proof is similar in the other direction. Hence, $G = G'$. □

No label $(a, \perp, \perp)$ occurs in an arc addition operation of $\widehat{t_K}$. Furthermore, $f((a, \perp, \perp)) = \perp$ for each $a \in C$. Hence, all labels $(a, \perp, \perp)$ can be replaced by the unique label $(c, \perp, \perp)$ for some fixed $c \in C$. It follows that $\widehat{t_K}$ and the relabellings h_i only use the following $8\,|C| + 1$ labels :

$(a, \top, \top), (a, \top, +), (a, \top, -), (a, \top, \bot), (a, +, +), (a, +, \bot),$
$(a, -, -), (a, -, \bot)$ for all $a \in C$ and $(c, \bot, \bot)$.

By using this remark, we obtain:

Proposition 5.6 : If a graph G is defined up to vertex labels by a composition of operations $\sigma[K, x_1, \ldots, x_p]$ such that each graph K has clique-width at most k, then, $cwd(G) \leq 8k + 1$.

Proof: Let G be defined by a term over nullary symbols and operations $\sigma[K, x_1, \ldots, x_p]$ such that $cwd(K) \leq k$. The terms t_K can be written with the labels of a set C of k labels. By composing the terms $\widehat{t_K}$ and the relabellings h_i according to Lemma 5.5, we obtain a term in $T(F_{C\#})$ that defines G by using at most $8k + 1$ labels. Hence, $cwd(G) \leq 8k + 1$. □

5.2 Directed graph-labelled trees.

Definition 5.7 : *Graph-labelled trees describing directed graphs.*

In the following definition, unspecified notions and notation are as in Definition 4.1. Graph labelled-trees will be represented by graphs with (directed) arcs and (undireceted) edges. Let us consider an edge as a pair of opposite arcs. Then a directed path or walk[37] can go through an edge, that is, through one of the two arcs of this edge.

(a) A *directed graph-labelled tree*, denoted by $\mathcal{T}$ is an undirected tree T with at least 3 nodes, that is equipped, for each node $v \in N_T$, with a directed graph H_v, called a *component*, and a bijection $\rho_v : Inc_T(v) \to V_{H_v}$. Components are pairwise disjoint and need not be connected, see Example 5.8(2) below. If v is a leaf, then H_v has a single vertex that we identify with v.

(b) We define $S(\mathcal{T})$ as the graph consisting of the union of the components augmented with the (undirected) edges $\rho_u(e)\rho_v(e)$ for $e = uv$, (cf. Figure 5, where the edges $\rho_u(e)\rho_v(e)$ are shown by dotted lines). The arcs are inside the components. A directed path or walk in $S(\mathcal{T})$ is *alternating* if no two consecutive arcs are in a same component. Hence, in a directed path, arcs alternate with edges. In a directed walk, an edge can be traversed consecutively several times.

There is at most one alternating path from a vertex x to a vertex y, but there may also exist one from y to x. This is the case if and only if each arc zz' on this path inside a component has an opposite arc $z'z$ (we are considering paths, not walks).

In view of Proposition 5.9, for a vertex x in a component H_u, we define $A^-(x)$ as the set of vertices y of $S(\mathcal{T})$ accessible from x by an alternating path (from x to y) whose first edge or arc is not in H_u, and $A^+(x)$ as the set of vertices w of $S(\mathcal{T})$ such that there is an alternating path from w to x whose last edge or arc is not in H_u.

[37] A *walk* is like a path but it can go several times through a vertex or an arc.

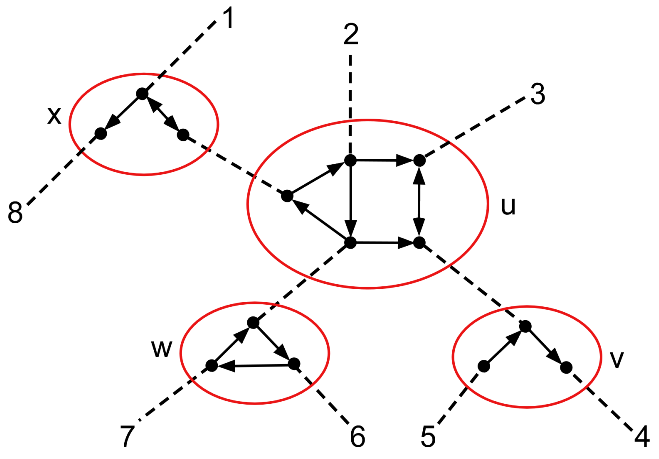

Figure 5: The directed graph-labelled tree T of Example 5.8(1).

(c) The *graph described by* $\mathcal{T}$, denoted by $G(\mathcal{T})$, has vertex set L_T (the set of leaves of T) and an arc uv if and only if there is an alternating path from u to v. If there is a directed path $u_1 \to u_2 ... \to u_p$ in $G(\mathcal{T})$, the concatenation of the alternating paths corresponding to the arcs $u_i u_{i+1}$ forms an alternating walk from u_1 to u_p.

(d) The *node-joining operation*[38] (called *elimination* of an edge e of T in [7]) is defined as follows. If $e = uv$ is an edge between two internal nodes of T, its contraction fuses u and v into a single node say w, giving tree T', and replaces the graphs H_u and H_v by $H'_w := (H_u \uplus H_v) - \{\rho_u(e), \rho_v(e)\}$ augmented with an arc from any vertex x in H_u to any vertex y in H_v such that $x\rho_u(e) \in E_{H_u}$ and $\rho_v(e)y \in E_{H_v}$. We obtain a directed graph-labelled tree $\mathcal{T}'$ that describes the same graph : this is easy to check by considering alternating paths. By iterating as much as possible this elimination step, we obtain a star whose central component is isomorphic to $G(\mathcal{T})$.

The opposite transformation called *node-splitting* also preserves the defined graph. $\square$

Theorem 2 of [18] defines canonical decompositions for strongly connected graphs whose components are cliques, stars and particular graphs called *cycles of transitive tournaments* that have clique-width at most 4 by Proposition 4.16 of [7]. We will not use this difficult notion. We will describe directed, possibly disconnected graphs by directed graph-labelled trees, either canonical or not.

Examples 5.8 : (1) Figures 5 shows a directed graph-labelled tree $\mathcal{T}$ and Figure 6 the graph $G(\mathcal{T})$. The double arrows in the components H_x and H_u

[38] Similar to that in Definition 4.1(d).

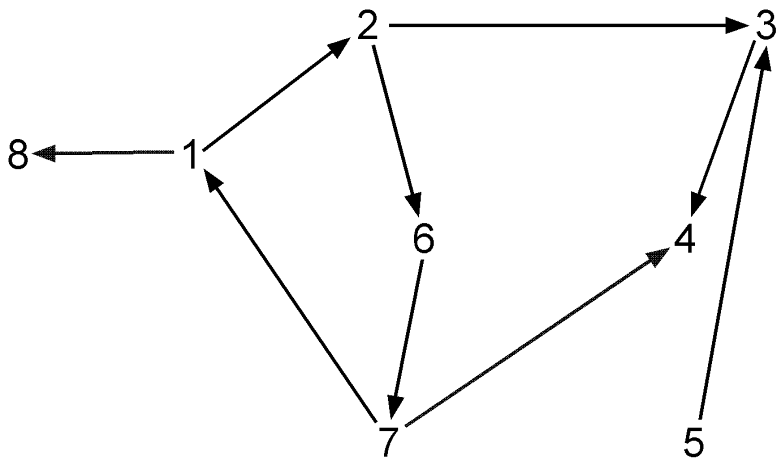

Figure 6: The graph defined by $\mathcal{T}$ of Figure 5.

indicate pairs of opposite arcs. For a comparison with Figure 1, there is here no alternating path between 2 and 7, in either direction. Hence, these two vertices are not adjacent in $G(\mathcal{T})$.

(2) A directed graph-labelled tree may define a disconnected graph although its components are connected. As a small example, consider $S(\mathcal{T})$ defined as $x - z' \longrightarrow z - u \longleftarrow u' - y$, whose internal components are $z' \longrightarrow z$ and $u \longleftarrow u'$. (The undirected edges between components are $x - z', z - u$ and $u' - y$). Then $G(\mathcal{T})$ consists of the two isolated vertices x and y. Eliminating the edge $z - u$ yields a component consisting of two isolated vertices u' and z'. Because of this observation, it is pointless to require that components be connected. $\square$

Proposition 5.9 : Let G be defined by a directed graph-labelled tree $\mathcal{T}$ whose components are strongly connected.

(1) For each vertex x of $S(\mathcal{T})$, we have $A^+(x) \cap L_T \neq \emptyset$ and $A^-(x) \cap L_T \neq \emptyset$.

(2) If xy is an arc of H_u, then $zz' \in E_G$ for all $z \in A^+(x) \cap L_T$ and $z' \in A^-(y) \cap L_T$. Conversely, if $x = \rho_u(uv)$ and $y = \rho_u(uw)$ are distinct vertices of H_u, if $z \in L_T \cap N_{T,u\backslash v}$, $z' \in L_T \cap N_{T,u\backslash w}$ and $zz' \in E_G$, then $z \in A^+(x) \cap L_T$, $z' \in A^-(y) \cap L_T$ and xy is an arc of H_u.

(3) G is strongly connected.

Note that T has at least two leaves and so, that G has at least two vertices.

Proof : (1) Let x be a vertex of a component H_u. Hence, $x = \rho_u(uv)$ for some node v. We use induction on the cardinality of $N_{T,u\backslash v}$ (the set nodes of T reachable from by a path going through v; cf. Lemma 4.3). If it is 1, then v is a leaf and $A^+(x) \cap L_T = A^-(x) \cap L_T = \{v\}$.

Otherwise, we have in H_v an arc zy such that $y = \rho_v(uv)$ and $z = \rho_v(vw)$ for some edge vw of T. We have $N_{T,v\backslash w} \subset N_{T,u\backslash v}$, hence, by induction, $A^+(z) \cap L_T \neq \emptyset$. As $A^+(z) \cap L_T \subseteq A^+(x) \cap L_T$ we have $A^+(x) \cap L_T \neq \emptyset$. The proof that $A^-(x) \cap L_T \neq \emptyset$ is similar[39].

(2) Just consider alternating paths, as in the proof of Lemma 4.3.

(3) The node-joining operation preserves the strong connectedness of the components as one checks from Definition 5.7(d). By repeating this operation, one obtains a directed graph-labelled tree that defines G and consists of one "central" strongly connected component and leaves. This component is isomorphic to G, hence G is strongly connected. □

Proposition 5.10 : Let G be defined by a directed graph-labelled tree $\mathcal{T}$. Then G is strongly connected if and only if all components of $\mathcal{T}$ are strongly connected.

Proof : The "if" direction is proved in the previous proposition. For the converse, let x and y be distinct vertices of a component H_u. Let $s \in A^+(x) \cap L_T$ and $t \in A^-(y) \cap L_T$. There is a path in G from s to t. Each arc of this path corresponds to an alternating path in $S(\mathcal{T})$. The concatenation of these paths is an alternating walk in $S(\mathcal{T})$. It is not necessarily a path because some edges of the tree may be traversed twice. (In the example of Figure 6 the path $2 \longrightarrow 6 \longrightarrow 7 \longrightarrow 4$ corresponds to a walk in $S(\mathcal{T})$ (Figure 5) that traverses twice the edge between H_u and H_w). This walk must enter H_u first via x and exit it last via y. Its arcs belonging to H_u form a directed path from x to y. Hence, H_u is strongly connected. □

Remark 5.11 : Even if $G(\mathcal{T})$ is strongly connected, some components of $\mathcal{T}$ may not be isomorphic to induced subgraphs of $G(\mathcal{T})$ (by contrast with Lemma 4.3(2)). Consider for an example a directed graph-labelled tree having two internal components isomorphic to the directed cycles $\overrightarrow{C}_3$. It defines $\overrightarrow{C}_4$ that does not contain $\overrightarrow{C}_3$ as an induced subgaph. We will compare below the clique-widths of a graph and its components.

5.3 Evaluating graph-labelled trees by means of substitutions.

We will use the set of labels $D := \{\perp, +, -, \top\}$.

Definitions 5.12 : *Rooted graph-labelled trees and related notions.*

(a) Let $\mathcal{T}$ be a directed graph-labelled tree with underlying tree T and $G := G(\mathcal{T})$. Let us select a node $r \in N_T$ and make it a root for T. As in

[39]The sets $A^-(x) \cap L_T$ and $A^+(x) \cap L_T$ may be different. In the example of Figure 7, we have $A^-(a) \cap L_T = \{1\}$ and $A^+(x) \cap L_T = \{0, 1\}$.

Definition 4.9(a,b), for u in N_T, we define $N_u := \{x \in N_T \mid x \leq_T u\}$ and $V_u := \{x \in L_T \mid x \leq_T u\} \subseteq V_G$. Hence, $V_r = L_T = V_G$ and $V_u = \{u\}$ if $u \in L_T - \{r\}$.

(b) Let $u \in N_T$. The *leader* of a component H_u such that $u \neq r$ is the vertex $\rho_u(wu)$ denoted by $\overline{u}$, where w is the father of u. We define a D-graph $H_u \backslash\backslash \overline{u}$ as the follows :

> if $u \neq r$ and u not leaf, we define $H_u \backslash\backslash \overline{u} := H_u - \overline{u}$ where a vertex x is labelled as follows : its label is $\top$ if $x\overline{u}$ and $\overline{u}x$ are in E_{H_u}, it is $+$ if $x\overline{u} \in E_{H_u}$ and $\overline{u}x \notin E_{H_u}$, it is $-$ if $\overline{u}x \in E_{H_u}$ and $x\overline{u} \notin E_{H_u}$, and it is $\perp$ if $x\overline{u}$ and $\overline{u}x$ are not in E_{H_u};
>
> if $u \neq r$ and u is a leaf, then $H_u \backslash\backslash \overline{u}$ is undefined (or is empty).
>
> if $u = r$, we define $H_u \backslash\backslash \overline{u} := H_r$, and all its vertices as dead (we use the notation $H_u \backslash\backslash \overline{u}$ for uniformity, although $\overline{u}$ is not defined).

As in Definition 4.9, the graphs $H_u \backslash\backslash \overline{u}$ depend on the chosen root r.

(c) If $u \in N_T$, we define $G_u := G[V_u]$ labelled as follows:

> If $u = r$, all vertices of $G_u = G$ are dead.
>
> Otherwise, a vertex x has label $\top$ if there are alternating paths from x to $\overline{u}$ and from $\overline{u}$ to x; it has label $+$ if there is an alternating path from x to $\overline{u}$ and no such path from $\overline{u}$ to x; it has label $-$ if there is an alternating path from $\overline{u}$ to x and no such path from x to $\overline{u}$ and label $\perp$ if there are no alternating paths between x and $\overline{u}$.
> $\square$

The following lemma, stated for the objects of the previous definition, generalizes Lemma 4.10.

Lemma 5.13 : If $u \in N_T - L_T$ has sons $u_1, ..., u_p$ and the corresponding p vertices of $H_u \backslash\backslash \overline{u}$ are $x_1, ..., x_p$ (that is, $x_i := \rho_u(uu_i)$), then we have $G_u = (H_u \backslash\backslash \overline{u})[x_1 \leftarrow G_{u_1}, ..., x_p \leftarrow G_{u_p}]$.

Proof : Let $K := (H_u \backslash\backslash \overline{u})[x_1 \leftarrow G_{u_1}, ..., x_p \leftarrow G_{u_p}]$. As in the proof of Lemma 4.10, the vertex sets of G_u and K are the same and the arcs of G_{u_i} are the same as in G_u and K.

We consider x in G_{u_i} and y in G_{u_j}, $j \neq i$. If xy is an arc of G, there is an alternating path from x to y. It must go through H_u via the arc $\rho_u(u_i u)\rho_u(u_j u) = x_i x_j$ in $H_u \backslash\backslash \overline{u}$. This path goes through the leader $\rho_{u_i}(u_i u)$ of H_{u_i}. Hence, x is alive in G_{u_i} and has label $+$ or $\top$. Similarly, y has label $-$ or $\top$ in G_{u_j}. Hence xy is an arc of K.

Conversely, if $xy \in E_K$, then $x_i x_j$ is an arc of $H_u \backslash\backslash \overline{u}$, x has label $+$ or $\top$ in G_{u_i} and y has label $-$ or $\top$ in G_{u_j}. Going back to definitions, we have an alternating path from x and y built from alternating paths from x to x_i, and from x_j to y, and the arc x_ix_j. Hence, xy is an arc of G, hence of G_u.

It remains to compare the vertex labels in K and in G_u.

Let x be a vertex of G_{u_i} labelled by $+$ in G_u. There is an alternating path from x to the leader $\overline{u}$ of H_u and no such path from $\overline{u}$ to x. Hence, there is an alternating path from x to $\overline{u_i}$ and an arc in H_u from $\rho_u(uu_i)$ to $\overline{u}$. Hence the label of $\rho_u(uu_i)$ is either $+$ or $\top$. The label of x in G_{u_i} is either either $+$ or $\top$, hence its label in K is either $+$ or $\top$. If it would be $\top$, we would have an arc in H_u from $\overline{u}$ to $\rho_u(uu_i)$ and an alternating path from $\overline{u}$ to x and x would have label $\top$ in G_u. Hence x has label $+$ in K.

The proofs are similar for the other labels. $\square$

Theorem 5.14 : Let G be defined by directed graph-labelled tree $\mathcal{T}$ whose components have clique-width at most k. Then $cwd(G) \leq 8k+1$.

Proof: From Proposition 5.6 and Lemma 5.13, along the lines of Theorem 4.11.

If one chooses a leaf as root r, and its unique son is u, then $G = \Lambda'(\perp, G_u)$ where $\Lambda'(G,H) := \sigma[K, x_1, x_2]$ and K consists of the the arcs x_1x_2 and x_2x_1, with x_1 is labelled by $\top$ and x_2 by $\perp$. Hence, Λ' generalizes for $\{\top, \perp, +, -\}$-graphs the operation Λ of Definition 2.5. $\square$

Example 5.15 : There exist graph-labelled trees $\mathcal{T}$ that have components of arbitrary large clique-width but define graphs without arcs, hence of clique-width 1. For an example, we take $\mathcal{T}$ to be a non-leaf-rooted graph-labelled tree with any connected graph H as root component. We attach to each vertex x of H a path of the form $x - v_x \longrightarrow v'_x - w'_x \longleftarrow w_x - \widetilde{x}$ (cf. Example 5.8(2)). Then $G(\mathcal{T})$ consists of the isolated vertices $\widetilde{x}$ because there is no directed alternating path between $\widetilde{x}$ and any different $\widetilde{y}$. $\square$

For strongly connected graphs, we have a better situation.

Proposition 5.16 : There is a function f such that $cwd(H) \leq f(cwd(G))$ whenever G is strongly connected and H is a component of some directed graph-labelled tree that defines it.

We need some technical definitions.

Definition 5.17 : *Arc contractions.*

Contracting the edges or arcs of set F in a graph G yields a graph $G \backslash F$, usually defined up to isomorphism or with vertices that are subsets of V_G. This is not convenient for our proof that uses logic.

If F is a set of arcs of a directed graph G, we denote by $x \sim_F y$ the fact that vertices x and y are linked by a path whose arcs can be traversed in either direction. If X is a set of vertices containing one and only one vertex of each equivalence class of $\sim_F$, we denote by $G \backslash (F, X)$ the directed graph H such that

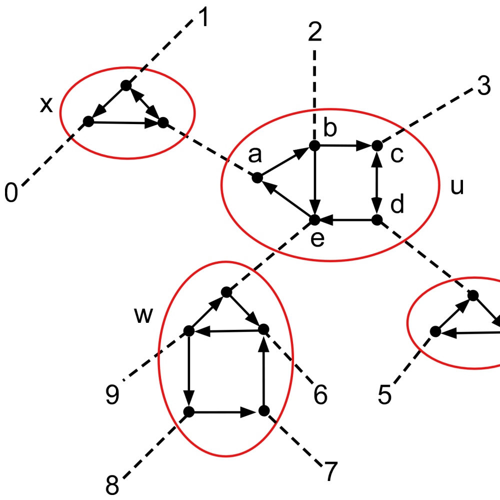

Figure 7: The directed graph-labelled tree $\mathcal{T}$ of Example 5.19.

$V_H := X$ and $xy \in E_H$ if and only if $x \neq y$ and there is in G an arc $\overline{xy}$ such that $\overline{x} \sim_F x$ and $\overline{y} \sim_F y$.

Proof of Proposition 5.16 : Let G be strongly connected and defined by a directed graph-labelled tree $\mathcal{T}$. Let H_u be a component not reduced to a single vertex. We will construct an induced subgraph G' of G such that H_u is isomorphic to $G' \backslash F$ for some set of arcs F. That $G' \backslash F$ is in some sense monadic second-order definable from G yields the existence of f by Corollary 7.38(2) of [14].

Step 1 : By performing node-joinings (cf. Definition 5.7(d)) that do not involve H_u, we obtain a directed graph-labelled tree $\mathcal{T}'$ that defines G, such that $u \in N_{T'}$, $H'_u = H_u$ and all nodes that are not leaves are neighbours of u. The graph-labelled tree of Figure 7 is of this type.

Step 2 : From now on, we assume that $\mathcal{T}$ satisfies this condition. We recall from Definition 5.7 that we identify v and the vertex in H_v if H_v has only one vertex, so that $L_T = V_G$. We will define an injective mapping $h : V_{H_u} \to V_G$, an induced subgraph G' of G whose vertex set contains $h(V_{H_u})$ and a set $F \subseteq E_{G'}$ such that h is an isomorphism $H_u \to G' \backslash (F, X)$ for some set X.

For each $x \in V_{H_u}$, we define $h(x)$ and a few related objects by one of the following three cases.

Case 1 : $x = \rho_u(uy)$ where $uy \in E_T$ and $y \in L_T = V_G$. Then, we define $h(x) := y$. In Figure 7, we have $h(b) = 2$ and $h(c) = 3$ by these conditions.

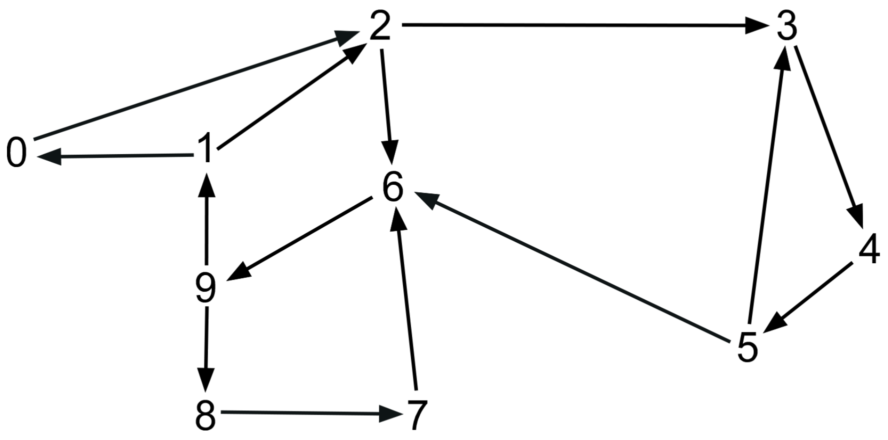

Figure 8: The strongly connected graph G defined by the graph-labelled tree of Figure 7.

Case 2 : $x = \rho_u(uv)$ where $v \notin L_T$ and there are in H_v two opposite arcs between $z := \rho_v(uv)$ and some vertex $\overline{z}$. We choose one such $\overline{z}$ and we define $h(x)$ as the vertex $y \in V_G = L_T$ such that $\rho_v(vy) = \overline{z}$. In Figure 7, we have in this way $h(a) = 1$.

We denote by Y_0 the set of vertices $h(x)$ obtained by these two cases.

Case 3 : $x = \rho_u(uv)$ where $v \notin L_T$ and the conditions of Case 2 do not hold. We let $z := \rho_v(uv)$. There is in H_v a directed cycle $z \to z_1 \to ... \to z_p \to z$ with $p \geq 2$ because this graph is strongly connected by Proposition 5.10. We choose one such that p is minimal. It follows that there is no arc in H_v between z and any of $z_2, ..., z_{p-1}$, and from z_i to z_j if $j > i + 1$.

Let $y_1, ..., y_p$ be the vertices of G such that $z_i = \rho_v(vy_i)$ for each i. We have a path $y_1 \to ... \to y_p$ such that there is no arc $y_i y_j$ with $j > i + 1$. We define $h(x) := y_1$, $h'(x) := y_p$, $V_x := \{y_2, ..., y_{p-1}\}$ and F_x as the set of arcs from $\{h(x)\} \uplus V_x$ to $\{h'(x)\} \uplus V_x$.

In Figure 7, we have in H_w a minimal directed cycle $\rho_w(uw) \to z_1 \to z_2 \to \rho_w(uw)$, $y_1 = h(e) = 6$ and $y_2 = h'(e) = 9$.

We denote by Y_1 and Y_2 the sets of such vertices $h(x)$ and, respectively, $h'(x)$. We denote by Y_3 the union of the sets V_x, and by F the union of the sets F_x. We define G' as the induced subgraph $G[Y_0 \uplus Y_1 \uplus Y_2 \uplus Y_3]$ and $X := Y_0 \uplus Y_1$.

In Figure 7, we have $Y_0 = \{1, 2, 3\}$, $Y_1 = \{4, 6\}, Y_2 = \{5, 9\}$ and $Y_3 = \emptyset$. The corresponding graph is in Figure 8. The set F consists of the arcs 45 and 69. The vertex set of G' is {1,2,3,4,5,6,9}. By contracting the arcs 45 and 69 of G', we obtain a graph isomorphic to H_u by : $a \longmapsto 1, b \longmapsto 2, c \longmapsto 3, d \longmapsto 4, e \longmapsto 6$.

Claim 1 : The mapping h is an isomorphism : $H_u \to G' \backslash (F, X)$.

Proof : By the definitions, h is a bijection : $V_{H_u} \to X = V_{G'\backslash(F,X)}$.

Let xx' be an arc of H_u. There is in $\mathcal{T}$ an alternating path from $h(x)$ or $h'(x)$ to $h(x')$. For proving this, we should conider the nine cases depending on the three possible cases for defining $h(x)$ and $h(x')$. We only consider the following two, the others being similar or straightforward.

Case (a) : $h(x)$ is defined by Case 2 and $h(x')$ by Case 3.

Let $x, z, \overline{z}$ and $y = h(x)$ be as in Case 2. Let $z' \to z'_1 \to ... \to z'_{p'} \to z'$, $y'_1, ..., y'_{p'}$ be as in Case 3 for x', with $h(x') := y'_1$. There is an alternating path $y - \overline{z} \to z - x \to x' - z' \to z'_1 - y'_1$ and $y'_1 = h(x')$. Hence we have $yy'_1 = h(x)h(x') \in E_{G'}$. This arc is also in $G'\backslash(F,X)$.

Case (b) : $h(x)$ and $h(x')$ are defined by Case 3. In H_v we have a minimal directed cycle $z \to z_1 \to ... \to z_p \to z$ with $p \geq 2$. We let $y_1, ..., y_p$ be the corresponding vertices of G' so that we have $h(x) := y_1$ and $h'(x) := y_p$. There is an alternating path $y_p - z_p \to z - x \to x' - z' \to y'_1$. Hence we have $y_p y'_1 = h'(x)h(x') \in E_{G'}$. As $h'(x) \sim_F h(x)$, the arc $h(x)h(x')$ is in $G'\backslash(F,X)$.

Conversely, let yu be an arc of $G'\backslash(F,X)$. We have $\overline{yu} \in E_{G'}$ such that $\overline{y} \sim_F y$, $\overline{u} \sim_F u$ and there is an alternating path from $\overline{y}$ to $\overline{u}$. As $y \neq u$, $\overline{y}$ and $\overline{u}$ are not in a same set $\{h(x), h'(x)\} \uplus V_x$, and so, we have $\overline{y} = h(x)$ or $\overline{y} = h'(x)$ and $\overline{u} = u = h(x')$ for some arc xx' of H_u. It follows that $y = h(x)$, hence $yu = h(x)h(x')$, which proves that h is an isomorphism. $\square$

Step 3 : We now prove that the transformation of G into $G'\backslash(F,X)$ is monadic second-order definable. We let G and the sets Y_0, Y_1, Y_2, Y_3 and F be as above in Step 2.

Claim 2 : Let x be as in Case 3. Let P be a directed path in G' of the form $y_1 = h(x) \to s_1 \to s_2 \to ... \to s_q$ with $s_1, ..., s_{q-1}$ in Y_3 and $s_q \in Y_2$. Then $V_x = \{s_1, ..., s_{q-1}\}$ and $s_q = h'(x)$.

Proof. We use the notation of Case 3.

The arc $y_1 s_1$ is defined by an alternating path of the form : $y_1 - z_1 \to t_1 - t'_1 \to ... - s_1$. By the conditions of Case 3, $t_1 \neq z$. Hence $t'_1 = s_1$. We may have $t_1 = z_p$. In this case $s_1 = y_p = h'(x)$, $V_x = \emptyset$ and the assertion holds. Otherwise, t_1 is some z_i. We cannot have $i > 2$ by the minimality of p (cf. Case 3). Hence $t_1 = z_2$ and $s_1 = y_2$.

We now consider $y_2 s_2$, the next arc on P. We have an alternating path $y_2 - z_2 \to t_2 - t'_2 \to ... - s_2$. By the definitions, if $t_2 = z_p$, then $s_2 = y_p = h'(x)$, $V_x = \{y_2\}$ and the assertion holds. Otherwise, t_2 is some z_i. We cannot have $i = 1$ or 2, otherwise P is a cycle, and neither $i > 3$ by the minimality of p. Hence $t_2 = z_3$ and $s_2 = y_3$. The proof continues similarly. Considering the next arc $y_3 s_3$ yields that $s_3 = y_4$, either in Y_2 (in that case $y_4 = y_p = h'(x)$) or in Y_3.

Finally, we can prove that the directed path $y_1 = h(x) \to s_1 \to s_2 \to ... \to s_q$ is identical to $y_1 \to y_2 \to y_3 \to ... \to y_p = h'(x)$. We have $V_x = \{s_1, ..., s_{q-1}\} = \{y_2, y_3, ..., y_{p-1}\}$ and $s_q = y_p = h'(x)$. $\square$

The graph $G'\backslash(F,X)$ is obtained from G by fusing the vertices $h(x)$ and $h'(x)$ for all vertices x as in Case 3, by keeping $h(x)$ as result of such a fusion and removing all vertices not in $X := Y_0 \cup Y_1$. Our next aim is to describe the pairs $(h(x), h'(x))$ by a monadic second-order formula.

Claim 3 : There exists a monadic second-order formula $\theta(Y_0, Y_1, Y_2, Y_3, y, z)$ such that, if G, Y_0, Y_1, Y_2, Y_3 are as in Step 2, then for every two vertices y, z of G, we have :

> $y = h(x)$ and $z = h'(x)$ for some vertex x of H_u if and only if $G \models \theta(Y_0, Y_1, Y_2, Y_3, y, z)$.

Proof : By Claim 2, the formula $\theta(Y_0, Y_1, Y_2, Y_3, y, z)$ need only express the following :

> $y \in Y_1$, $z \in Y_2$ and there exists a directed path from y to z whose intermediate vertices are in Y_3.

Since transitive closure, whence path properties, can be expressed in monadic second-order logic (cf. [14]), we can construct such a formula θ. □

To finish the proof, we define $\overline{\theta}(Y_0, Y_1, Y_2, Y_3, y, z)$ as :

$$y = z \vee \theta(Y_0, Y_1, Y_2, Y_3, y, z) \vee \theta(Y_0, Y_1, Y_2, Y_3, z, y).$$

and $\alpha(Y_0, Y_1, Y_2, Y_3, y, z)$ as :

$$\exists \overline{y}, \overline{z}.[\overline{\theta}(Y_0, Y_1, Y_2, Y_3, \overline{y}, y) \wedge \overline{\theta}(Y_0, Y_1, Y_2, Y_3, \overline{z}, z) \wedge edg(y, z)].$$

Given a directed graph G and pairwise disjoint sets of vertices Y_0, Y_1, Y_2, Y_3, we define $H(Y_0, Y_1, Y_2, Y_3)$ as the directed graph with vertex set $Y_0 \cup Y_1$ and arcs yz such that $G \models \alpha(Y_0, Y_1, Y_2, Y_3, y, z)$. In the words of [14], this graph is obtained from G by a monadic second-order transduction taking as arguments (called there *parameters*) pairwise disjoint sets of vertices Y_0, Y_1, Y_2, Y_3. By Corollary 7.38(2) of [14], there exists a function f such that $cwd(H(Y_0, Y_1, Y_2, Y_3)) \leq f(cwd(G))$ for all such graph G and sets Y_0, Y_1, Y_2 and Y_3. If G, H_u, Y_0, Y_1, Y_2 and Y_3 are[40] as in Steps 1 and 2, then $H(Y_0, Y_1, Y_2, Y_3)$ is isomorphic to H_u. This completes the proof. □

This proof does not give a good bound for f. It is an open question whether $cwd(H_u) \leq cwd(G)$ or even $cwd(H_u) = O(cwd(G))$ in such a case.

Remark 5.18 : The special case of Proposition 5.16 where H_u is a component of the canonical split-decomposition of a strongly connected graph can be derived from Theorem 4.21 in [7]. The proof of Proposition 4.16 of [7] giving that result is incorrect : Lemma A.2.3 shows correctly that $cwd(H) \leq 4cwd(G)$ if H is obtained from G by fusing any two vertices, but, in order to prove the statement, one must fuse the vertices of several pairs (as we do above for defining $G' \backslash (F, X)$ from G'), hence, one does not obtain a bounding function f as claimed. □

[40] For some sets Y_0, Y_1, Y_2, Y_3, the graph $H(Y_0, Y_1, Y_2, Y_3)$ may not be isomorphic to any component H_u of a graph-labelled tree. Nevertheless $cwd(H(Y_0, Y_1, Y_2, Y_3)) \leq f(cwd(G))$

5.4 Related work

Kanté and Rao have defined in [35] the *displit decomposition* of a directed graph. For an undirected graph, it is the same as the split decomposition. It is incomparable with the split decomposition of [18] because the prime components are different. However, every connected directed graph has a unique decomposition. Furthermore, for an appropriate notion of rank-width for directed graphs, they obtain that the rank-width of a graph is the least upper-bound of the rank-widths of the components of its displit decomposition (cf. Remark 4.15).

They also characterize the directed graphs of rank-width at most 1 in a way that generalizes the various characterizations of distance-hereditary graphs, in particular that of [37].

We think that the results of this section and those of [7] about the existence of monadic second-order transformations between directed graphs and their canonical split decompositions can be extended to displit decompositions.

6 Conclusion

Our purpose was to clarify the relationships between split-decompositions for directed and undirected graphs, substitutions to vertices and the related graph grammars, and also to obtain good bounds on the clique-widths of the defined graphs. For doing that we have generalized, in Definitions 2.1 and 5.1, the notion of substitution used in the theory of modular decomposition .

One open problem is the study of the operations defined in Definition 2.5 and their equational properties.

The methods of Section 5 should help to investigate particular classes of directed graphs regarding their clique-width and their generation by unambigous grammars (if possible), along the lines of Section 4.5. Manipulating grammars so as to reach unambiguity in view of counting and random generation has proved useful in Section 4.5. It would be interesting to investigate in a similar way the undirected graphs whose prime components are cycles. These graphs have bounded clique-width by Theorem 4.14 and the upper-bound of 4 to the clique-width of cycles. This fact encourages to try to define them by grammars.

References

[1] H.-J. Bandelt and H. Mulder, Distance-hereditary graphs, *Journal of Combinatorial Theory, Series B*, **41** (1986) 182–208.

[2] A. Brandstädt and V.B. Le, Structure and linear-time recognition of 3-leaf powers. *Inf. Process. Lett.* **98** (2006) 133-138.

[3] M.S. Chang, S.Y. Hsieh and G.H. Chen, Dynamic programming on distance-hereditary graphs, *Proceedings of ISAAC 1997, Algorithms and Computation*, *Lec. Notes Comput. Sci* 1350, Springer, 1997, 344-353.

[4] C. Chauve, É. Fusy and J. Lumbroso, An exact enumeration of distance-hereditary graphs. *Proceedings of ANALCO* (14th Workshop on Analytic Algorithmics and Combinatorics), Barcelona, January 2017, Proceedings pp. 31-45

[5] S. Cicerone and G. Di Stefano, On the extension of bipartite to parity graphs. *Discrete Applied Mathematics* **95** (1999) 181-195.

[6] B. Courcelle, An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theor. Comput. Sci.* **55** (1987) 141-181.

[7] B. Courcelle, The monadic second-order logic of graphs XVI : Canonical graph decompositions. *Logical Methods in Computer Science* **2** (2006).

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

[9] B. Courcelle, Clique-width and edge contraction. *Inf. Process. Lett.* **114** (2014) 42-44.

[10] B. Courcelle, From tree decompositions to clique-width terms, *Discrete Applied Mathematics*, **248** (2018) 125-144.

[11] B. Courcelle, On quasi-planar graphs : clique-width and logical description. 2018, *Discrete Applied Mathematics*, this issue, https://doi.org/10.1016/j.dam.2018.07.022.

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

[13] B. Courcelle and I. Durand, Computations by fly-automata beyond monadic second-order logic, *Theor. Comput. Sci*, **619** (2016) 32-67.

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

[15] B. Courcelle, P. Heggernes, D. Meister, C. Papadopoulos and U. Rotics, A characterisation of clique-width through nested partitions, *Discrete Applied Maths*, **187** (2015) 70-81.

[16] B. Courcelle and M. Kanté, Graph operations characterizing rank-width, *Discrete Applied Mathematics* **157** (2009) 627-640.

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

[18] W. Cunningham, Decomposition of directed graphs, *SIAM. J. on Algebraic and Discrete Methods*, **3** (1981) 214–228.

[19] R. Diestel, *Graph theory*, Springer, 2006.

[20] R. Downey and M. Fellows, *Fundamentals of parameterized complexity*, Springer-Verlag, 2013.

[21] I. Durand, TRAG: Term Rewriting Automata and Graphs, a software developped since 2015, http://dept-info.labri.u-bordeaux.fr/~idurand/trag

[22] I. Durand and M. Raskin, TRAG-WEB: Term Rewriting Automata and Graphs (online), Web interface under development, 2018, https://trag.labri.fr

[23] M. Fellows, F. Rosamond, U. Rotics and S. Szeider, Clique-width is NP-complete. *SIAM J. Discrete Math.* **23** (2009) 909-939.

[24] E. Fischer J. Makowsky and E. Ravve, Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics* **156** (2008) 511-529.

[25] P. Flajolet, P. Zimmermann and B. Van Cutsem, A calculus for the random generation of labelled combinatorial structures. *Theor. Comput. Sci.* **132** (1994) 1-35.

[26] F. Fomin, S. Oum, D. Thilikos, Rank-width and tree-width of H-minor-free graphs. *Eur. J. Comb.* **31** (2010) 1617-1628.

[27] E. Gioan and C. Paul, Split decomposition and graph-labelled trees: Characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Applied Mathematics* **160** (2012) 708-733.

[28] E. Gioan, C. Paul, M. Tedder and D. Corneil, Practical and efficient split decomposition via graph-labelled trees. *Algorithmica* **69**(2014): 789-843.

[29] M. Golumbic and U. Rotics, On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.* **11** (2000) 423-443.

[30] F. Gurski, The behavior of clique-width under graph operations and graph transformations. *Theory Comput. Syst.* **60** (2017) 346-376.

[31] M. Habib and C. Paul, A survey of the algorithmic aspects of modular decomposition. *Computer Science Review* **4** (2010) 41-59.

[32] P. Heggernes, D. Meister and C. Papadopoulos, Characterising the linear clique-width of a class of graphs by forbidden induced subgraphs, *Discrete Applied Mathematics* **160** (2012) 888-90.

[33] P. Hlinený, S. Oum, D. Seese and G. Gottlob, Width parameters beyond tree-width and their applications. *Comput. J.* **51** (2008) 326-362.

[34] M. Kanté and M. Rao, The rank-width of edge-coloured graphs. *Theory Comput. Syst.* **52** (2013) 599-644.

[35] M. Kanté and M. Rao, Directed rank-width and displit decomposition. *Proceedings of WG 2009*, *Lecture Notes in Computer Science* **5911** (2010) 214-225.

[36] D. Meister, Clique-width with an inactive label. *Discrete Mathematics* **337** (2014) 34-64.

[37] S. Oum, Rank-width and vertex-minors, *Journal of Combinatorial Theory, Series B*, **95** (2005) 79–100.

[38] V. Ravelomanana and L.Thimonier, Asymptotic enumeration of cographs. *Electronic Notes in Discrete Mathematics* :**7** (2001) 58-61.