Algorithmica (2000) 27: 348–381
DOI: 10.1007/s004530010024

Algorithmica
© 2000 Springer-Verlag New York Inc.

Definability Equals Recognizability of Partial 3-Trees and *k*-Connected Partial *k*-Trees[1]

D. Kaller[2]

Abstract. We consider graph decision problems on partial 3-trees that can be solved by a finite-state, leaf-to-root tree automaton processing a width-3 tree decomposition of the given graph. The class of yes-instances of such a problem is said to be *recognizable* by the tree automaton. In this paper we show that any such class of recognizable partial 3-trees is *definable* by a sentence in CMS logic. Here, CMS logic is the extension of Monadic Second-order logic with predicates to count the cardinality of sets modulo fixed integers. We also generalize this result to show that recognizability (by a tree automaton that processes width-k tree decompositions) implies CMS-definability for k-connected partial k-trees. The converse—definability implies recognizability—is known to hold for any class of partial k-trees, and even for any graph class with an appropriate definition of recognizability. It has been conjectured that recognizability implies definability over partial k-trees; but a proof was previously known only for $k \leq 2$. This paper proves the conjecture, and hence the equivalence of definability and recognizability, over partial 3-trees and k-connected partial k-trees. We use techniques that may lead to a proof of this equivalence over all partial k-trees.

Key Words. Partial k-trees, Treewidth, Monadic Second-order logic, Tree decompositions, Tree automata.

1. Introduction.

Many NP-hard graph problems are known to have linear-time algorithms over partial k-trees (see [1] for a survey of early results). Takamizawa et al. [26] described a general technique to construct such algorithms over series-parallel graphs (i.e., partial 2-trees). A number of more general formalisms were later developed [4]–[6], [22], including several based on the Monadic Second-order (or MS) logic [3], [9], [12], [14]. Often, it is very easy to define a given graph decision problem with a sentence in MS logic; such a sentence can be automatically translated into a linear-time dynamic-programming algorithm to solve the problem over partial k-trees. Although this automatic translation is intractable, it is nevertheless of major theoretical interest. The resultant dynamic-programming algorithms are modeled by finite-state machines called *tree automata*. To use such a machine, the input graph must first be parsed into a *tree decomposition*; this parsing can also be done in a (theoretically) efficient way [7]. The time complexity bounds for the parsing and for the dynamic-programming are both linear in the size of the partial k-tree, with very large constants (which grow superexponentially with k).

[1] These results were presented at the 22nd Workshop on Graph-Theoretic Concepts in Computer Science [20], and they form part of the author's Ph.D. dissertation [19].
[2] School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6. Current address: Pivotal Corporation, 300–224 West Esplanade, North Vancouver, British Columbia, Canada V7M 3M6. dkaller@pivotal.com.

Received February 3, 1997; revised July 24, 1997. Communicated by H. L. Bodlaender.

A partial k-tree is a graph that can be constructed by gluing together *basic* graphs—each with $k + 1$ or fewer vertices—along the global shape of a tree. This tree, together with the corresponding basic graphs, constitutes a width-k tree decomposition [25] of the graph so constructed. Arnborg et al. [2] showed how to parse a given graph into a width-k tree decomposition (or conclude that the graph is not a partial k-tree) in polynomial time; later, Bodlaender [7] gave a linear-time algorithm to do this. A tree decomposition can be encoded as a rooted tree with nodes labeled from a finite set of basic graphs. A tree automaton processes a tree decomposition by assigning each of its nodes to one of a finite number of states—computed as a function of the node's label and the states of its children. The tree is accepted iff its root is thus assigned to a designated accepting state. Only constant time is needed to compute the state of each node; so a tree automaton decides in linear time whether or not to accept a tree decomposition. In order for a tree automaton to be a decision algorithm over partial k-trees, it must accept either all of the width-k tree decompositions of a given graph, or none of them (as graphs do not have unique tree decompositions).

A subclass of the partial k-trees is said to be *recognized* by a tree automaton that accepts exactly the tree decompositions of graphs in that subclass. Courcelle [10] was the first to show that if a sentence of the *Counting Monadic Second-order* (or CMS) logic defines a subclass (say Π) of the partial k-trees, then there exists a tree automaton to recognize Π. Here, CMS logic is the extension of MS logic with predicates to count the cardinality of sets modulo fixed integers. Similar results were given by Arnborg et al. [3] and by Borie et al. [9]. These results have far-reaching consequences: The question of whether or not a given graph belongs to a certain class is general enough to capture many commonly studied decision problems, and a large number of these problems can be encoded as CMS sentences [9], [11], [12], [19]. For many such classes (such as Hamiltonian graphs or 3-colorable graphs), the "counting" predicates are not needed: these classes can be defined by MS sentences. On the other hand, the "counting" predicates are needed to define, for example, the class of graphs with an even number of vertices: these graphs cannot be defined by an MS sentence, but they can be defined by a CMS sentence [12]. In other cases, it is not possible for a CMS sentence to define all the graphs with a given property, but a CMS sentence can nevertheless define the partial k-trees with the property; an example of this is given by the class of graphs for which the vertex set can be partitioned into fixed-size independent sets [16]. Courcelle's result [10], [12] says that the partial k-trees in any of these classes can be recognized in linear time by a tree automaton. In fact, most of the standard NP-complete graph problems [15] can be solved in linear time over partial k-trees in this way.

The "definability implies recognizability" result is given by Courcelle [12] for classes of structures (more general than partial k-trees) that can be constructed recursively using a finite set of "gluing" operations. The notion of "recognizability" must be generalized appropriately for such structures. The recognizability of partial k-trees by tree automata is analogous to the recognizability of regular sets by conventional finite-state automata, and it is well known that a set of finite strings is recognizable iff it can be defined by a regular expression [17], [24]. However, for a long time it was an open question whether CMS logic analogously characterizes the sets of partial k-trees that are recognizable by tree automata. Although CMS-definability was known to imply recognizability, the converse implication had not been established. The converse was known to hold only for partial 1-trees [12], partial 2-trees [13], and, with the results presented here, partial 3-trees and k-connected partial k-trees. Also, Kabanets [18] showed the converse holds for *partial k-paths* (i.e., partial k-trees that can be composed by gluing together basic graphs along the global shape of a path). Very recently, Lapoire [21] finally settled the question, establishing the equivalence for all partial k-trees.

Courcelle [12] shows how CMS predicates encoding the structure of a tree decomposition could be used to encode whether it is accepted by a given tree automaton. Hence, to prove that recognizability implies CMS-definability, we need only show that CMS predicates can indeed encode the structure of a fixed tree decomposition of any partial k-tree. This is easy for $k = 1$ because a tree is (roughly speaking) its own tree decomposition. In fact, MS logic is sufficient to encode a fixed tree decomposition of any partial 1-tree [12]; the "counting" feature of CMS logic is needed only to simulate a tree automaton on that encoded structure. Courcelle [13] also showed that MS logic can encode a fixed tree decomposition of any partial 2-tree. In this paper we extend this result, showing that MS logic can encode a fixed tree decomposition of any partial 3-tree or k-connected partial k-tree. We then appeal to Courcelle's result [12] to conclude that recognizability implies CMS-definability over those classes of graphs.

The rest of this paper is organized as follows: In Section 2 we review graph-theoretic notation and preliminaries. In Section 3 we discuss partial k-trees and tree decompositions. In Section 4 we describe a general approach for proving that MS logic can encode a fixed tree decomposition of a partial k-tree, and in Section 5 we explain how this enables CMS logic to encode the behavior of a tree automaton. In Section 6 we show that any connected partial k-tree admits a *simple* tree decomposition—this can be used to decompose a partial k-tree into *trunk-graphs* (which are partial 3-paths). In Section 7 we decompose a 2-connected partial 3-tree into a hierarchy of these trunk-graphs, and we develop several important properties of the hierarchy. In Section 8 we show how those properties enable MS predicates to encode the vertex set and edge set of each trunk-graph, and in Section 9 we show how they enable MS predicates to encode the structure of a *pyramid* in each trunk-graph. These results are combined in Section 10 to show that MS predicates can describe a fixed tree decomposition of any 2-connected partial 3-tree. In Section 11 we draw the conclusion that recognizability implies CMS-definability of partial 3-trees. In Section 12 we generalize the proof to k-connected partial k-trees. Finally, in Section 13, we make concluding remarks and discuss open problems.

2. Preliminaries.

The graphs in this paper are finite, simple, loop-free, and undirected. The vertex set of a graph G is denoted by $V(G)$, and its edge set is denoted by $E(G)$. We write $G' \sqsubseteq G$ to indicate that G' is a subgraph of G. The subgraph of G induced by $V' \subseteq V(G)$ is denoted by $G_{[V']}$. If V' is a set (but not necessarily a subset of $V(G)$), then $G\backslash V'$ is the subgraph of G induced by $V(G) - V'$. For a singleton set $\{v\}$, we may write simply $G\backslash v$ for $G\backslash\{v\}$. For a subgraph G' of G, we may write simply $G\backslash G'$ for $G\backslash V(G')$.

A *cut-set* of a graph G is a subset V' of $V(G)$ such that G has fewer components than $G\backslash V'$. The unique element of a singleton cut-set is called a *cut-vertex*. A cut-set (or cut-vertex) V' is said to *separate* any pair of vertices that are in the same component of G, but in different components of $G\backslash V'$.

If b is a vertex of a rooted tree T, then T_b is the subtree of T that is rooted at b. In this paper a *leaf* of a rooted tree is either a degree-1 vertex other than the root, or a degree-0 vertex (which is the root). This definition is for the convenience of having a unique leaf in any path that is rooted at one of its endpoints.

DEFINITION 2.1. Suppose T is a rooted tree. A *trunk* in T is a rooted path $P \sqsubseteq T$ between some vertex v of T and some leaf of T_v; v is then taken to be the root of P.

Suppose P and P' are paths in some graph: If $V(P) \cap V(P') = \emptyset$, then P and P' are called *vertex-disjoint* paths. If $v \in V(P) \cap V(P')$ implies that v is an endpoint of both P and P', then they are called *internally vertex-disjoint* paths. We adopt the following definition for the connectivity of a graph.

DEFINITION 2.2. An *ℓ-connected* graph (for $\ell \in \mathbb{N}$) is a graph for which there are ℓ internally vertex-disjoint paths between each pair of nonadjacent vertices.

It is more usual to define an ℓ-connected graph, alternatively, as a graph from which at least ℓ vertices must be removed in order to obtain either a disconnected graph, or the graph with a single vertex [8]. Lemma 2.3 (first given by Menger [23]) shows that these two characterizations are equivalent for graphs with more than ℓ vertices.

LEMMA 2.3. *Let $\ell \in \mathbb{N}$, and suppose G is a connected graph on $\ell + 1$ or more vertices. G contains ℓ internally vertex-disjoint paths between each pair of vertices iff there is no cut-set of G with cardinality less than ℓ.*

By Definition 2.2, a graph on ℓ or fewer vertices is ℓ-connected iff it is a clique; by the alternative definition, no such small graph would be ℓ-connected.

DEFINITION 2.4. A *block* of a graph G is a 2-connected subgraph of G that is induced by a maximal subset of $V(G)$.

Under our notion of connectivity (Definition 2.2) a block may consist of a single vertex or a pair of adjacent vertices.

3. Partial *k*-Trees and Tree Decompositions.

A *k-tree* is either the clique on k vertices, or a graph that can be obtained (recursively) from a k-tree G by adding a new vertex and making it adjacent to any k distinct vertices that induce a clique in G. A *partial k-tree* is a subgraph of a k-tree. For example, a graph is a partial 0-tree iff its edge set is empty; a graph is a partial 1-tree iff it is a forest. Series-parallel graphs and outerplanar graphs are subclasses of the partial 2-trees; Halin graphs form a subclass of the partial 3-trees. It is not hard to show (see, e.g., [28]) that a graph is a partial k-tree iff it admits a *width-k tree decomposition*:

DEFINITION 3.1. A *tree decomposition* of a graph G is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X} = \{X_a\}_{a \in V(T)}$ is a collection of subsets of $V(G)$, indexed by the nodes of T, for which the following three properties are satisfied:

- $\bigcup_{a \in V(T)} X_a = V(G)$.
- Each edge of G has both endpoints in some set $X_a \in \mathcal{X}$.
- If $a, b, c \in V(T)$ such that b lies on the path between a and c, then $X_a \cap X_c \subseteq X_b$.

We refer to elements of $V(T)$ as *nodes*, so as not to confuse them with vertices of G. The set X_b is called the *bag* indexed by $b \in V(T)$. If no bag in $\mathcal{X}$ contains more than $k+1$ vertices, then (T, χ) is called a *width-k* tree decomposition. If T' is a subgraph of T, then $\mathcal{X}_{T'}$ is the collection of bags indexed by $V(T')$; and $X_{T'}$ is the the union of those bags:

$$\mathcal{X}_{T'} = \{X_a \in \mathcal{X} \mid a \in V(T')\}, \qquad X_{T'} = \bigcup_{a \in V(T')} X_a.$$

We refer to the subgraph of G that is induced by $X_{T'}$ as the subgraph *underlying* T'. It will be useful to designate k or fewer vertices of a partial k-tree as *terminals*:

DEFINITION 3.2. A *terminal set* of a partial k-tree G is a proper subset V' of $V(G)$ with cardinality $|V'| \leq k$, such that G admits a width-k tree decomposition in which V' is a subset of some bag.

We assume that any graph G has a specially designated (possibly empty) terminal set, denoted by $V_{\text{term}}(G)$. We can then construct a tree decomposition such that these terminals all belong to the bag indexed by a designated root:

DEFINITION 3.3. A *rooted* tree decomposition of a graph G is a triple $(T, r, \mathcal{X})$ where T is a rooted tree; $r \in V(T)$ is the root; (T, χ) is a tree decomposition of G; and $V_{\text{term}}(G) \subseteq X_r$.

If (T, r, χ) is a rooted tree decomposition, then we may refer to X_r as the *root bag*; and we may refer to any bag indexed by a leaf of T as a *leaf bag*. For ease of expression, if $a \in V(T)$ is the parent (or child, ancestor, descendant, etc.) of $b \in V(T)$, then we may simply say that X_a is the parent (or child, ancestor, descendant, etc.) of X_b.

DEFINITION 3.4. Suppose (T, r, χ) is a rooted tree decomposition of a graph G; and let $b \in V(T)$. If $b = r$, then each vertex of $X_b - V_{\text{term}}(G)$ is called a *drop* vertex of b; otherwise, each vertex of $X_b - X_p$ is called a *drop* vertex of b, where $p \in V(T)$ is the parent of b. If $v \in X_b$ is not a drop vertex of b, then v is called a *nondrop* vertex of b.

We use the notion of a "drop" vertex to assign each bag of a width-k tree decomposition to one of a constant number (dependent only on k) of equivalence classes:

DEFINITION 3.5. A *basic k-graph* is a graph on $k+1$ or fewer vertices—each labeled with a distinct integer between 1 and $k+1$, and each designated as either a "drop" or a "nondrop" vertex. Two basic graphs, B and B', are *equivalent* if there is an isomorphism between $V(B)$ and $V(B')$ that respects the labels and designations of each vertex.

The collection of these equivalence classes is called the *k-derivation alphabet* (denoted by Σ_k).

The next proposition follows easily from Definitions 3.1 and 3.5, providing us with a labeled tree to encode any given tree decomposition. Such a tree will be input to a tree automaton, as discussed in Section 5.

PROPOSITION 3.6. *Let Σ_k be the k-derivation alphabet; and suppose (T, r, χ) is a width-k rooted tree decomposition of a graph G. There exists a function $\sigma\colon V(T) \to \Sigma_k$, such that*

- *each vertex of G can be labeled with an integer between* 1 *and* $k + 1$, *such that*
- *for each $b \in V(T)$, the subgraph $G_{[X_b]}$ is equivalent to $\sigma(b)$, where a vertex of $G_{[X_b]}$ is designated as a "drop" vertex iff it is a drop vertex of b.*

Such a function σ is called a derivation function *for G on the tree T.*

In order to modify tree decompositions, we use the following two operations to *contract* edges and *split* nodes.

DEFINITION 3.7. Suppose (T, r, χ) is a rooted tree decomposition such that $b \in V(T)$ is the parent of $c \in V(T)$. Let T' be the tree obtained from $T\backslash\{b, c\}$ by adding a new node (say b') and making it adjacent to each node that was adjacent to b or c. Let $r' = b'$, if $r \in \{b, c\}$; otherwise, let $r' = r$. Let $\mathcal{X}' = \mathcal{X} - \{X_b, X_c\} \cup \{X_{b'}\}$, where $X_{b'} = X_b \cup X_c$. We say that $(T', r', \mathcal{X}')$ is obtained from (T, r, χ) by *contracting* $\{b, c\}$.

DEFINITION 3.8. Suppose (T, r, χ) is a rooted tree decomposition such that $b \in V(T)$. Let T' be a tree obtained from $T\backslash b$ by adding two new nodes (say b' and b'') with the edge $\{b', b''\}$, and also adding one edge (either $\{a, b'\}$ or $\{a, b''\}$) whenever $\{a, b\} \in E(T)$. Let $r' = b'$, if $r = b$; otherwise, let $r' = r$. Let $\mathcal{X}' = \mathcal{X} - \{X_b\} \cup \{X_{b'}, X_{b''}\}$ for some $X_{b'}, X_{b''} \subseteq X_b$. We say that $(T', r', \mathcal{X}')$ is obtained from (T, r, χ) by *splitting* b.

To use this operation, we must specify how to choose the edges incident to b' and b'', and how to choose the bags $X_{b'}$ and $X_{b''}$.

4. Counting Monadic Second-Order Logic.

A graph $G = (V, E)$ can be interpreted as a logical structure over the universe $V \cup E$, with a predicate $Edge(e, v)$ that holds whenever $v \in V$ is an endpoint of $e \in E$. Many different graph properties can then be expressed as logical sentences consisting of the following symbols: individual variables (to represent vertices or edges); set variables (to represent sets of vertices or edges); the equality ($=$) and membership ($\in$) symbols; existential ($\exists$) and universal ($\forall$) quantifiers; the logical operators $\wedge$ ("and"), $\vee$ ("or"), $\neg$ ("not"), $\Rightarrow$ ("implies"), and $\Leftrightarrow$ ("if and only if"); the *Edge* predicate; and unary predicates $\mathbf{card}_{\ell,c}$ for nonnegative integer constants ℓ, c (with $\ell < c$). If S is a set, then $\mathbf{card}_{\ell,c}(S)$ is true iff S has cardinality $\ell (\bmod c)$. Courcelle [12] has shown that these "counting" predicates do give the logic additional expressive power.

By a CMS "formula," we mean a string of the symbols described above, constructed such that the usual syntactic rules of logic are observed; quantification is allowed over both individual and set variables. If a CMS formula has no occurrence of an unquantified variable, then it is called a *CMS sentence*. We write $G \models \Phi$ to indicate that a CMS sentence Φ is true on the *evaluation* graph G. This sentence defines a certain class (say Π) of graphs: that is, $G \models \Phi$ iff $G \in \Pi$.

In a CMS formula Φ, some number (say ℓ) of the unquantified variables may be designated as *arguments*: This defines a *CMS predicate*—which we denote by the symbol "Φ" followed by an ℓ-tuple of its arguments. If every unquantified variable has been designated as an argument, then Φ *encodes* a certain relation (say $\mathcal{L}$): that is, $(v_1, v_2, \ldots, v_\ell) \in \mathcal{L}$ iff $\Phi(v_1, v_2, \ldots, v_\ell)$ is true on the evaluation graph. For example, a CMS predicate encoding the (symmetric and irreflexive) edge relation can be encoded as follows.

$$\Phi_{\text{adj}}(u, v) \equiv \neg(u = v) \wedge (\exists e)(Edge(e, u) \wedge Edge(e, v)). \tag{4.1}$$

The following lemma shows that the transitive closure of Φ_{adj} can also be encoded:

LEMMA 4.2. *If a CMS predicate can encode a binary relation on the vertex set of a graph, then a CMS predicate can encode the transitive closure of that relation.*

PROOF. See Lemma 3.7 of [12]. □

Using the transitive closure of Φ_{adj}, it is easy to express (for example) the property that a graph is connected. When writing CMS formulae in this paper, we often use high-level expressions (such as "G is connected") rather than providing a detailed translation into the low-level logical symbols of CMS. Refer to [3], [9], [11], or [12] for a further discussion of encoding such expressions.

We now generalize the notion of a CMS predicate, allowing it to be defined with unquantified variables that are not arguments.

DEFINITION 4.3. Suppose Φ is a CMS predicate defined with ℓ arguments and also some other unquantified variables (called *parameters*); the arguments and parameters may denote individual vertices or edges, or sets of them. After letting each parameter be substituted by some fixed value, let $\mathcal{L}$ be the relation such that $(v_1, v_2, \ldots, v_\ell) \in \mathcal{L}$ iff $\Phi(v_1, v_2, \ldots, v_\ell)$ is true on the evaluation graph. We say that Φ is an *existentially defined* predicate encoding $\mathcal{L}$; or (more simply) that Φ *existentially encodes* $\mathcal{L}$.

If Φ is an existentially defined CMS predicate, then its parameters (say $x_1, x_2, \ldots, x_d$) can be existentially quantified at the outermost level of a CMS sentence as follows:

$$(\exists x_1, x_2, \ldots, x_d)(\Phi'). \tag{4.4}$$

Here, Φ' is a CMS formula that does not contain any unquantified variable other than $x_1, x_2, \ldots, x_d$; so the predicate Φ can be used in writing the CMS sentence 4.4. Two (or more) existentially defined predicates may be used together, in sentences of this form, by quantifying all of their parameters at the outermost level. Thus these predicates can depend upon a common list of parameters, subject to a common set of conditions. We use this approach to encode the structure of a rooted tree decomposition of the evaluation graph G. To do this, some of the vertices of G are designated as *witnesses*—each representing a node of the tree decomposition.

DEFINITION 4.5. Suppose **Bag** and **Parent** are binary CMS predicates. These predicates are said to *describe* a rooted tree decomposition (T, r, χ) of a graph G if there exists a one-to-one function $f\colon \mathcal{X} \to V(G)$, such that

- **Bag**(v, X) holds iff $v = f(X)$, and
- **Parent**(p, c) holds iff $f^{-1}(p)$ and $f^{-1}(c)$ exist, and $f^{-1}(p)$ is the parent of $f^{-1}(c)$.

For $b \in V(T)$, we then refer to the vertex $f(X_b)$ as the *witness* of b (or the *witness* of X_b).

We existentially define these CMS predicates (**Bag** and **Parent**) to describe a rooted tree decomposition (T, r, χ) of any partial 3-tree or k-connected partial k-tree. These predicates can then be used to encode a rooted tree (isomorphic to T) on the set of witnesses: i.e., the following subset of $V(G)$:

$$V(T) = \{v \in V(G) \mid (\exists X)\,\mathbf{Bag}(v, X)\}\,. \tag{4.6}$$

In Section 5 we show how the **Bag** and **Parent** predicates can be used to write a CMS sentence 4.4, where Φ' encodes whether the described tree decomposition is accepted by a particular tree automaton. The validity of this CMS sentence is, of course, dependent upon "correct" values having been chosen for the parameters. The following lemma shows that Φ' can be written to ensure that "correct" values are indeed chosen.

LEMMA 4.7. *Suppose* **Bag** *and* **Parent** *are existentially defined CMS predicates with parameters* $x_1, x_2, \ldots, x_d$. *There exists a CMS predicate* Φ *with* d *arguments and zero parameters such that* $\Phi(c_1, c_2, \ldots, c_d)$ *is true iff* **Bag** *and* **Parent** *describe a width-*k *rooted tree decomposition of the evaluation graph when each parameter* x_i *is substituted by the value* c_i $(1 \le i \le d)$.

PROOF. Let $G = (V, E)$ be the evaluation graph. Within the scope of quantification of $x_1, x_2, \ldots, x_d$, the predicate Φ can identify $V(T) \subseteq V$ as in (4.6). Now, Φ can easily verify that **Bag** maps one unique bag to each $v \in V(T)$. By Lemma 4.2, a CMS predicate can encode the transitive closure of **Parent**, and this can be used to verify that a tree on $V(T)$ is encoded. Now, it is not difficult for CMS logic to encode the three properties itemized in Definition 3.1. Thus, to verify that a *width-k* tree decomposition is described, Φ need only check that **Bag**$(v, X) \Rightarrow$ "$|X| \le k + 1$." $\square$

We will need the following lemmas later in this paper:

LEMMA 4.8. *A CMS predicate can existentially encode edge directions over any subset of the edges of an* (*undirected*) *partial* k*-tree.*

PROOF. Suppose G is a partial k-tree; and let $E' \subseteq E(G)$. We can encode edge directions over E' using a binary CMS predicate Φ with $k+2$ parameters $V_1, V_2, \ldots, V_{k+1}$ and E''.

$$\Phi(u, v) \equiv (\exists e \in E') \left(Edge(e, u) \wedge Edge(e, v) \wedge \neg(u = v) \right.$$
$$\left. \wedge \left((e \in E'') \Leftrightarrow \bigvee_{i=1}^{k} \bigvee_{j=i+1}^{k+1} (u \in V_i \wedge v \in V_j) \right) \right).$$

Let $V_1, V_2, \ldots, V_{k+1}$ be independent sets partitioning $V(G)$. For any edge $e \in E'$, assume without loss of generality that its endpoints are $u \in V_i$ and $v \in V_j$ where $i < j$. Thus, if e belongs to the edge subset E'', then $\Phi(u, v)$ is true and $\Phi(v, u)$ is false. Otherwise, $\Phi(u, v)$ is false and $\Phi(v, u)$ is true. The set E'' consists of the edges directed from a vertex in a lower-indexed set V_i to a vertex in a higher-indexed set V_j $(i < j)$. Therefore, any choice of edge directions over E' can be encoded by choosing an appropriate subset E'' of E'. □

LEMMA 4.9. *A CMS predicate can existentially encode a constant-length string of bits for each vertex and each edge of a graph.*

PROOF. We need only provide ℓ parameters $X_1, X_2, \ldots, X_\ell$ to represent a string of ℓ bits for each vertex or edge (say x): The ith bit $(1 \leq i \leq \ell)$ is turned on iff $x \in X_i$. □

5. Tree Automata.

A tree automaton is a finite-state machine that can be used to decide whether a partial k-tree G belongs to a certain class of graphs. For this purpose, the automaton's input is the tree T of a width-k rooted tree decomposition of G, with the nodes of T labeled by a derivation function for G (Proposition 3.6). By restricting the input to be a labeled path (rather than any tree), we obtain a machine that is equivalent to a conventional finite-state automaton.

DEFINITION 5.1. A *tree automaton* over an alphabet Σ is a triple $\mathcal{A} = (\mathcal{S}, \mathcal{S}_A, f)$ where $\mathcal{S}$ is a finite set of (say ℓ) *states* $S_1, S_2, \ldots, S_\ell$; $\mathcal{S}_A \subseteq \mathcal{S}$ is the set of *accepting* states; and $f\colon \Sigma \times \mathbb{N}^\ell \to \mathcal{S}$ is the *transition function*.

The *input* to $\mathcal{A}$ is a rooted tree T labeled by $\sigma\colon V(T) \to \Sigma$. Each node b of T is then recursively assigned to the state $f(\sigma(b), |S_1 \cap C|, |S_2 \cap C|, \ldots, |S_\ell \cap C|)$, where $C \subseteq V(T)$ consists of the children of b. The tree T is *accepted* by $\mathcal{A}$ iff its root is thus assigned to a state in $\mathcal{S}_A$.

If we say that a tree automaton processes width-k tree decompositions, we mean that its alphabet is the k-derivation alphabet Σ_k (see Definition 3.5). If we say that it accepts a tree decomposition (T, χ) of G, we mean that it accepts the tree T labeled by $\sigma\colon V(T) \to \Sigma_k$ whenever σ is a derivation function for G.

DEFINITION 5.2. Let Π be a class of partial k-trees; and suppose $\mathcal{A}$ is a tree automaton that processes width-k tree decompositions. We say that $\mathcal{A}$ *recognizes* Π if the following statements are equivalent:

- (T, χ) is a width-k tree decomposition of some graph in Π.
- $\mathcal{A}$ accepts (T, χ).

Since graphs do not have unique tree decompositions, this means that if a tree automaton recognizes a class of graphs, then it behaves consistently—either accepting or rejecting all of the tree decompositions of any given graph. Courcelle [12] (see also [3] and [9]) has shown that this type of tree automaton exists for any class of CMS-definable partial k-trees. In fact, the specifications of the tree automaton are inherent in the CMS sentence; a decision algorithm can be obtained automatically from the logical description of the graph class.

THEOREM 5.3. *If Φ is a CMS sentence, then (for each $k \in \mathbb{N}$) there exists a tree automaton that recognizes the intersection of $\{G \mid G \models \Phi\}$ with the class of partial k-trees.*

The reader may be more familiar with an alternative definition of a tree automaton [27] with a ternary transition function $f'\colon \Sigma \times \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ and an initial state S_0: Here, the input must be a *binary* tree, and each node b is assigned to the state $f'(\sigma(b), S, S')$, where $S, S' \in \mathcal{S}$ are the states of its children (for a nonleaf b) or $S = S' = S_0$ (for a leaf b). However, if f' is commutative in its second and third arguments, then it can be simulated by the transition function f (of Definition 5.1) by restricting the sum of the ℓ numeric arguments to be zero or two. For our purposes, the commutativity is no real restriction, because we are using tree automata to process tree decompositions, and there is no notion of order among the children of a tree decomposition node. In fact, for our purposes, the transition function of Definition 5.1 can (conversely) be simulated by the alternative one. This is because a partial k-tree always admits a width-k tree decomposition with a binary tree. Hence, a tree automaton is no weaker if we restrict its input to binary tree decompositions. We prefer to work with general tree decompositions, however, because we are unable to encode binary tree decompositions in CMS logic. Courcelle [12, Proposition 5.4] has established that a tree automaton (in the sense of Definition 5.1) can be simulated by CMS logic—provided a tree decomposition has been encoded. Hence, to prove the converse of Theorem 5.3, we need only develop CMS predicates to describe (Definition 4.5) a tree decomposition. In this paper, we show how this can be done (under certain restrictions).

LEMMA 5.4. *Suppose a subclass Π of the partial k-trees is recognized by some tree automaton over Σ_k. Suppose further that there exist CMS predicates describing a width-k rooted tree decomposition of any evaluated partial k-tree. It follows that a CMS sentence Φ can be written such that $G \models \Phi$ iff $G \in \Pi$.*

PROOF. Let $\mathcal{A}$ be the tree automaton that recognizes Π. Let **Bag** and **Parent** be CMS predicates describing (Definition 4.5) a width-k rooted tree decomposition of the evaluated partial k-tree, say G; and let (T, r, χ) be the tree decomposition thus described. Hence, the CMS sentence Φ must be written such that $G \models \Phi$ iff (T, r, χ) is accepted. To do this, a set $V(T) \subseteq V(G)$ of witnesses is identified, as in (4.6), and this set is partitioned into sets $S_1, S_2, \ldots, S_\ell$ representing the states of $\mathcal{A}$. The tree decomposition is accepted iff the witness of r is thus assigned to an accepting state. The CMS statement Φ need only verify that each node $b \in V(T)$ is assigned to the "correct" set: i.e., the one representing the state $f(\sigma(b), |S_1 \cap C|, |S_2 \cap C|, \ldots, |S_\ell \cap C|)$, where the following CMS formula is satisfied: $c \in C \;\Leftrightarrow\; c \in V(T) \wedge \mathbf{Parent}(b, c)$. Because the transition function is periodic in its last ℓ arguments, this can be encoded using the **card** predicates of CMS logic—see Proposition 5.4 of [12] for details. $\square$

6. Simple Tree Decompositions.

To construct a CMS-encodable tree decomposition, we begin with a *simple* tree decomposition, and then modify it. In this section we discuss how a *trunk-graph* is obtained by adding certain edges to the subgraph underlying a trunk (Definition 2.1) in a simple tree decomposition; and we show that a trunk-graph can be represented by a *pyramid* structure.

DEFINITION 6.1. A *simple* tree decomposition is a rooted tree decomposition (T, r, χ) for which each node b of T satisfies the following properties:

P1. There is exactly one drop vertex of b.
P2. The subgraph underlying T_b is connected.
P3. If V' is a subset of the nondrop vertices of b, then V' is not a cut-set of the subgraph underlying T_b.

We will need the following consequence of these properties:

FACT 6.2. *Suppose (T, r, χ) is a simple tree decomposition. If $b \in V(T)$ is the parent of $c \in V(T)$, then X_c contains the drop vertex of b.*

We now show that any connected partial k-tree G admits a simple tree decomposition, provided no cut-set of G is comprised exclusively of designated terminals (Definition 3.2). For example, $V_{\text{term}}(G) = \emptyset$ is suitable for this purpose; and provided G is ℓ-connected with $|V(G)| \geq \ell + 1$, it is easy to find a suitable terminal set of cardinality ℓ.

LEMMA 6.3. *If G is a connected partial k-tree for which no subset of $V_{\text{term}}(G)$ is a cut-set, then G admits a width-k rooted tree decomposition (T, r, χ) such that r satisfies properties* P1, P2, *and* P3.

PROOF. Suppose G is a connected partial k-tree for which no subset of $V_{\text{term}}(G)$ is a cut-set; and let (T, χ) be a width-k tree decomposition of G. Without loss of generality, assume $V_{\text{term}}(G)$ is a subset of some bag in $\mathcal{X}$; and choose the root r of T such that $V_{\text{term}}(G) \subseteq X_r$. For any child c of r, we can assume that X_c is not a subset of X_r; for otherwise we could contract the edge $\{r, c\}$ (see Definition 3.7). Now, if $V_{\text{term}}(G) = X_r$, then we can augment X_r with some vertex of $X_c - X_r$. Furthermore, if

$X_r - V_{\text{term}}(G)$ contains more than one vertex, then we can split r into two nodes (see Definition 3.8) such that the bag indexed by the new root is $V_{\text{term}}(G) \cup \{v\}$, for some vertex $v \in V(G) - V_{\text{term}}(G)$. So without loss of generality, assume $|X_r - V_{\text{term}}(G)| = 1$. Therefore, (T, r, χ) is a width-k rooted tree decomposition of G such that r satisfies P1; r also satisfies P2 and P3 because G is connected and no subset of $V_{\text{term}}(G)$ is a cut-set. □

LEMMA 6.4. *If G is a connected partial k-tree for which no subset of $V_{\text{term}}(G)$ is a cut-set, then G admits a width-k simple tree decomposition.*

PROOF. By Lemma 6.3, G admits a width-k rooted tree decomposition (T, r, χ) such that r satisfies P1, P2, and P3. Assume, inductively, that T' is a connected subgraph of T for which $r \in V(T')$ and each node of T' satisfies P1, P2, and P3. Suppose c is a child of some node $b \in V(T')$; and let G^c be the subgraph underlying T_c. Without loss of generality, assume $G^c \backslash X_b$ is connected; for otherwise we could create a copy of $(T_c, c, \mathcal{X}_{T_c})$ for each component (say H) of $G^c \backslash X_b$, restricting the bags in this copy to contain only the vertices of $V(H) \cup X_c$. Assume also that each vertex in $X_c \cap X_b$ is adjacent to one or more vertices of $G^c \backslash X_b$; for otherwise we could delete the violating vertices of $X_b \cap X_c$ from each bag in $\mathcal{X}_{T_c}$. Thus G^c is connected, and no subset of the nondrop vertices of c is a cut-set of G^c. We can now assume, without loss of generality (by Lemma 6.3), that c satisfies P1, P2, and P3. It follows inductively that there exists a tree decomposition of G in which all nodes satisfy these properties. □

In Section 7 we decompose a partial k-tree by recursively choosing trunks (Definition 2.1) in the tree of a simple tree decomposition.

DEFINITION 6.5. Suppose (T, r, χ) is a rooted tree decomposition of a graph G; and let P be a trunk in T. The *trunk-graph* of P is obtained from $G_{[X_P]}$ as follows: Add an edge between each pair of vertices contained in the intersection $X_p \cap X_c$, for each child $c \in V(T \backslash P)$ of each node $p \in V(P)$. The terminal set of this trunk-graph consists of the nondrop vertices of the root of P.

Definition 6.5 yields a trunk-graph with tree decomposition $(P, \mathcal{X}_P)$. Any graph that admits this type of *path decomposition* is called a *partial k-path*.

DEFINITION 6.6. A *simple path decomposition* is a simple tree decomposition $(P, r, \mathcal{X})$ for which P is a path and r is an endpoint of P. A *simple partial k-path* is any graph that admits a width-k simple path decomposition.

If (T, r, χ) is a simple tree decomposition, and $P \sqsubseteq T$ is a trunk rooted at r', then it is not necessarily the case that $(P, r', \mathcal{X}_P)$ is a *simple* path decomposition of $G_{[X_P]}$, because the nodes of P do not necessarily satisfy properties P2 and P3 relative to the subgraph $G_{[X_P]}$. However, each node of P does satisfy these properties relative to the trunk-graph of P:

LEMMA 6.7. *Suppose (T, r, χ) is a simple tree decomposition; and let $r' \in V(T)$. If P is a trunk in T that is rooted at r', then $(P, r', \mathcal{X}_P)$ is a simple path decomposition of the trunk-graph of P.*

PROOF. Let G be the graph underlying T; and let R be the trunk-graph of a trunk P rooted at r'. Since $(P, r', \mathcal{X}_P)$ is a path decomposition of $G_{[X_P]}$, and R is obtained from $G_{[X_P]}$ by adding edges only between pairs of vertices contained in a common bag of $\mathcal{X}_P$, it follows that $(P, r', \mathcal{X}_P)$ is a path decomposition of R. It is clear that each node of this path decomposition satisfies property P1. To complete the proof, we need only show that each node satisfies properties P2 and P3 relative to the trunk-graph R.

For the leaf b of P, the subgraph (of R) underlying P_b is identical to the subgraph (of G) underlying T_b; so P2 and P3 are satisfied by the leaf of P. Assume inductively that all three properties are satisfied by $b \in V(P)$, and let a be the parent of b. By Fact 6.2, X_b contains the drop vertex (say v) of a. Suppose $u \in X_a - X_b$ such that $\{u, v\} \notin E(G)$. By property P3 (relative to G), there is a path between u and v with one or more internal vertices in $X_{T_a} - X_a$. It follows that a has a child $c \in V(T \backslash P)$ for which $u, v \in X_c$. So, by Definition 6.5, u and v are adjacent in R. Therefore, v is adjacent to each vertex of $X_a - X_b$. The remaining vertices of the subgraph underlying P_a are the vertices in X_{P_b}; and X_{P_b} induces (inductively) a connected subgraph that is not cut by any subset of $X_a \cap X_b$. Therefore, the subgraph underlying P_a is connected (hence, P2); and v belongs to every subset of X_a that is cut-set of the subgraph underlying P_a (hence, P3). It follows inductively that $(P, r', \mathcal{X}_P)$ is a simple path decomposition of R. $\square$

We now show that any simple partial k-path (i.e., any trunk-graph) contains a *pyramid* consisting of k vertex sequences. We use the pair $(A, \rightarrow)$ to denote such a sequence, where "$x \rightarrow y$" means that $y \in A$ immediate follows $x \in A$. We use "$\rightarrow^+$" to denote the transitive closure of "$\rightarrow$"; and we use "$\rightarrow^*$" to denote its reflexive, transitive closure.

DEFINITION 6.8. Suppose R is a simple partial k-path. A *pyramid* in R consists of a vertex $v_1 \in V(R)$ and k vertex sequences $(A_i, \rightarrow_i)$, $1 \le i \le k$, with the following properties:

D1. $\{A_1, A_2, \ldots, A_k\}$ is a partition of $V(R) - \{v_1\}$.
D2. For $1 \le i \le k$: $v \in A_i$ is adjacent to v_1 only if v is the first vertex of $(A_i, \rightarrow_i)$.
D3. For $1 \le i \le k$: v belongs to $A_i \cap V_{\text{term}}(R)$ only if v is the last vertex of $(A_i, \rightarrow_i)$.
D4. For $1 \le i \le k$: two vertices $u, v \in A_i$ are adjacent (i.e., $\{u, v\} \in E(R)$) only if $u \rightarrow_i v$.
D5. For $2 \le \ell \le k$: if $i_1, i_2, \ldots, i_\ell$ are distinct indices between 1 and k, and each A_{i_j} $(1 \le j \le \ell)$ contains two distinct vertices (say $u_{i_j} \rightarrow_{i_j}^+ u'_{i_j}$), then not all of the following are edges of R: $\{u_{i_1}, u'_{i_2}\}$, $\{u_{i_2}, u'_{i_3}\}, \ldots, \{u_{i_{\ell-1}}, u'_{i_\ell}\}$, $\{u_{i_\ell}, u'_{i_1}\}$.

The vertex v_1 is called the *apex* of the pyramid; and each sequence $(A_i, \rightarrow_i)$ is called an *axis* of the pyramid. An edge $e \in E(R)$ is an *apical* edge if one of its endpoints is the apex; e is an *axial* edge if both endpoints belong to the same axis; otherwise e is a *cross* edge.

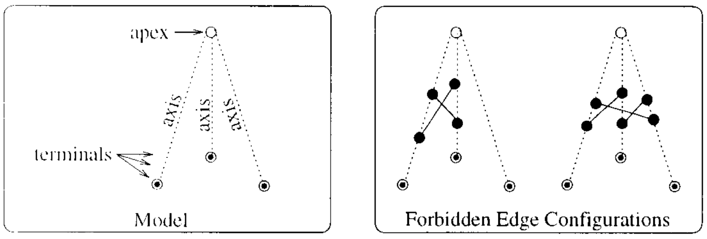

Fig. 1. A pyramid in a simple partial 3-path.

Figure 1 illustrates a pyramid in a simple partial 3-path with three terminals. If there are fewer terminals, then not every axis ends with one. The apex may be adjacent only to the first vertex of each axis. Axial edges exist only between consecutive vertices. Property D5 says that pairs and triples of cross edges are forbidden to “criss-cross” as illustrated. To construct a pyramid, we visit the bags of a simple path decomposition in order from the leaf to the root: a vertex being seen for the last time is called a *drop* vertex (Definition 3.4); and a vertex being seen for the first time is called an *add* vertex:

DEFINITION 6.9. Suppose $(P, r, \mathcal{X})$ is a simple path decomposition; and let $b \in V(P)$. If b is the leaf of P, then each vertex of X_b is called an *add* vertex of b; otherwise, each vertex of $X_b - X_c$ is called an *add* vertex of b, where $c \in V(P)$ is the child of b. If $v \in X_b$ is not an add vertex of b, then v is called a *nonadd* vertex of b.

LEMMA 6.10. *There exists a pyramid in any simple partial k-path.*

PROOF. Suppose R is a simple partial k-path; and let $(P, r, \mathcal{X})$ be a width-k simple path decomposition of R. Choose the apex of the pyramid to be the drop vertex of the leaf of P; and let each nondrop vertex of the leaf become the first element of a distinct axis. Now, assume inductively that $b \in V(P)$ is the parent of $c \in V(P)$ such that the vertices of X_{P_c} have already been assigned to axes; and let h be the number of nonadd vertices of b (so $h = |X_c \cap X_b| \leq k$). Inductively, each vertex in $X_c \cap X_b$ is the last element of a distinct axis, say $(A_i, \rightarrow_i)$ for $1 \leq i \leq h$; and Fact 6.2 allows us to assume the last element of $(A_h, \rightarrow_h)$ is the drop vertex of b. There are at most $k + 1 - h$ add vertices of b; so each can be placed at the end of a distinct axis $(A_i, \rightarrow_i)$, for $h \leq i \leq k$. It is easy to see that this approach will satisfy properties D1–D4.

Suppose property D5 is not satisfied. So there exist $\ell \geq 2$ edges $\{u_{i_1}, u'_{i_2}\}$, $\{u_{i_2}, u'_{i_3}\}$, $\ldots$, $\{u_{i_{\ell-1}}, u'_{i_\ell}\}$, $\{u_{i_\ell}, u'_{i_1}\}$ where $u_{i_j} \rightarrow^{+}_{i_j} u'_{i_j}$ $(1 \leq j \leq \ell)$. Let b be the closest node to the leaf of P such that X_{P_b} contains all 2ℓ of these endpoints. Without loss of generality, assume that u'_{i_1} is an add vertex of b; hence, u_{i_ℓ} also belongs to X_b, and so does u'_{i_ℓ}. Thus $(A_{i_\ell}, \rightarrow_{i_\ell})$ is the unique axis containing more than one vertex of X_b. Therefore, u'_{i_ℓ} is an add vertex of b, but it is adjacent to a vertex $u_{i_{\ell-1}} \notin X_b$ (a contradiction). □

Each of the axes in a pyramid gives part of an elimination order for the vertices of the corresponding partial k-path. In Section 10 we show how these orders can be interleaved to obtain a fixed path decomposition.

7. A Trunk Hierarchy. In this section we decompose a 2-connected partial 3-tree G into a hierarchy of trunk-graphs (Definition 6.5). We begin with a width-3 simple tree decomposition of G, and recursively choose trunk-graphs satisfying three special properties.

DEFINITION 7.1. Suppose (T, r, χ) is a rooted tree decomposition of a graph G. A *trunk hierarchy* of G is a collection $\mathcal{R}$ of trunk-graphs obtained by partitioning $V(T)$ into a collection of trunks—then taking the trunk-graph of each one. We say that this hierarchy is *admitted* by (T, r, χ).

Suppose $\mathcal{R}$ is the trunk hierarchy obtained by partitioning the nodes of T into trunks. Let T' be the subgraph of T obtained by contracting each such trunk into a single node. The trunk-graphs in $\mathcal{R}$ have an obvious one-to-one correspondence with $V(T')$. We use the terms "root," "child," "parent," etc., with implied reference to T', when speaking of these trunk-graphs.

REMARK. The pair $(T', \mathcal{R})$ is similar to a tree decomposition: A trunk-graph R_b corresponds to each node b of T'; each vertex of G belongs to at least one of these trunk-graphs; and each edge of G has both endpoints in some trunk-graph. Furthermore, if R_p is the parent of R_c, then $V_{\text{term}}(R_c) = V(R_p) \cap V(R_c)$, and R_p has an edge (possibly not an edge of G though) between each pair of vertices that are terminals of R_c.

Throughout this section G is a 2-connected partial 3-tree with either two or three terminals. Without loss of generality, assume that no subset of $V_{\text{term}}(G)$ is a cut-set of G; hence, by Lemma 6.4, G admits a width-3 simple tree decomposition. We will show how this tree decomposition can be perturbed so that it admits a hierarchy $\mathcal{R}$ of trunk-graphs, each satisfying Properties 7.2, 7.4, and 7.6. Our discussion describes how to obtain a simple tree decomposition (T, r, χ) with a trunk P rooted at r; the corresponding trunk-graph then becomes the root of $\mathcal{R}$. Following this, a trunk-graph can be chosen recursively from the subgraph underlying T_c, for each child $c \in V(T \backslash P)$ of each node $p \in V(P)$.

The first property gives an order on the vertex set of each trunk-graph. In Section 8 we define a CMS predicate to identify these vertices inductively, in this order:

PROPERTY 7.2. The vertices of each trunk-graph $R \in \mathcal{R}$ can be ordered $v_1, v_2, \ldots, v_{|V(R)|}$ such that, for each $i = 2, 3, \ldots, |V(R)|$, there is a nonterminal vertex v_j of R (where $1 \leq j \leq i - 1$) for which at least one of the following conditions is satisfied:

C1. v_i and v_j are adjacent (in G).
C2. R has a child $R' \in \mathcal{R}$ for which $V_{\text{term}}(R') = \{v_i, v_j\}$.
C3. R has a child $R' \in \mathcal{R}$ for which $V_{\text{term}}(R') = \{v_i, v_j, v_{j'}\}$, where $j' \leq i - 1$; and there is a path in $G \backslash \{v_j, v_{j'}\}$ between v_i and some terminal in $V_{\text{term}}(G)$.

We will show that Property 7.2 is enforced if each trunk-graph corresponds to a *centered* trunk of a simple tree decomposition (T, r, χ). This is a trunk P, rooted at r, for which each node satisfies two properties in addition to P1, P2, and P3 (Definition 6.1). When we refer to a vertex v as an *add* vertex of $p \in V(P)$, we mean that Definition 6.9 is to be interpreted relative to the path decomposition $(P, r, \mathcal{X}_P)$: that is, $v \in X_p - X_c$, for the child $c \in V(P)$ of p; but v may belong to the bag $X_{c'}$, for any other child $c' \in V(T \backslash P)$.

PROPOSITION 7.3. *G admits a width-3 simple tree decomposition (T, r, χ) such that T contains a* centered *trunk—this is a trunk P, rooted at r, in which each node $b \in V(P)$ satisfies the following properties*:

P4. *If b has children $c \in V(P)$ and $c' \in V(T \backslash P)$, then $X_{c'}$ contains at most one vertex of $X_b - X_c$.*

P5. *If v is an add vertex of b, then either $v \in V_{\text{term}}(G)$, or v is adjacent to some vertex of $G \backslash X_{T_b}$.*

PROOF. Suppose (T, r, χ) is a simple tree decomposition of G; and let $b \in V(T)$. It follows from property P1 (Definition 6.1) and Fact 6.2 that a trunk $P \sqsubseteq T$ whose nodes all satisfy P4 can be found with a greedy search from the root of T. We now show how to perturb the tree decomposition so that P5 is also satisfied by each node of the perturbed trunk. This process is illustrated in Figure 2.

Suppose $b \in V(P)$ fails to satisfy P5; so b has an add vertex (say v) that is not a terminal; and if $u \in V(G)$ is adjacent to v, then both u and v belong to a common bag in $\mathcal{X}_{T_b}$. Let $a \in V(P)$ be the ancestor of b such that v is the drop vertex of a; and let $a' \in V(P)$ be the child of a (possibly $a' = b$). It follows from property P3 (Definition 6.1) that $X_a \subseteq X_{a'}$; so we can contract (Definition 3.7) the edge $\{a, a'\}$ without violating P2 or P3. The contracted node has two drop vertices (including v); so we delete v from each bag indexed by $T \backslash T_b$. Now, b is the only node with two drop vertices; so we split b (Definition 3.8) into b' and b'', with b' the parent of b''. Let $X_{b'} = X_b - \{v\}$; let $X_{b''} = X_b$; and for each child c of b, let its parent become b'' if $v \in X_c$, and b' otherwise. It is not difficult to verify that each node of T continues to satisfy P1, P2, and P3. Furthermore, P4 is satisfied by each node of the trunk (say P') obtained from P by contracting $\{a, a'\}$ and replacing b with b'. Since $X_{P'}$ contains fewer vertices than X_P, this operation can be applied repeatedly until each node of the trunk also satisfies P5. $\square$

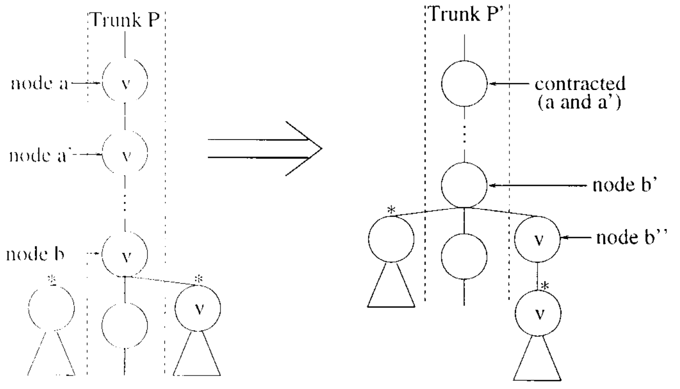

Fig. 2. Enforcing P5 ("$*$" indicates there may be multiple similar subtrees).

After choosing a centered trunk P, we modify the simple tree decomposition so that the trunk-graph of the (modified) centered trunk satisfies two additional properties:

PROPERTY 7.4. If y and z are vertices of a trunk-graph $R \in \mathcal{R}$, then there is a path $H \sqsubseteq R$ between y and z that satisfies the following conditions:

H1. No internal vertex of H is a terminal of R.

H2. If $\{u, v\} \in E(H) - E(G)$, then G contains two internally vertex-disjoint paths between u and v for which no internal vertex belongs to R, nor to any ancestor of R.

Since the trunk-graph R admits a simple path decomposition $(P, r, \mathcal{X}_P)$, it follows from Definition 6.1 that there is a path H satisfying H1 between any pair of vertices of R. If this path does not satisfy H2, then we can use the following proposition to "promote" vertices belonging to bags indexed by descendants of P:

PROPOSITION 7.5. *Suppose P is a centered trunk of a width-3 simple tree decomposition (T, r, χ) of G; and let $b \in V(P)$. Suppose further that v is an add vertex of b; and let u be the drop vertex of b. Let G' be the subgraph of G induced by $V(G) - X_P \cup \{u, v\}$. If x is a cut-vertex of G' that separates u from v, then there exists a centered trunk P' of some width-3 simple tree decomposition of G, such that $X_{P'} = X_P \cup \{x\}$.*

PROOF. By property P3 (Definition 6.1), b has a child $c \in V(T \backslash P)$ such that $u, v \in X_c$, as shown on the left of Figure 3. If there is a cut-vertex x of G' that separates u from v, then this child c is unique, and x belongs to $X_{T_c} - X_b$. Let the (say ℓ) components of the subgraph induced by $X_{T_c} - X_b - \{x\}$ be enumerated $G'_1, G'_2, \ldots, G'_\ell$. For $1 \leq i \leq \ell$, let V_i be the largest subset of $X_b \cup \{x\}$ in which each vertex is adjacent to at least one vertex of G'_i; and let G_i be the subgraph of G induced by $V(G'_i) \cup V_i$. A width-3 tree decomposition of G_i is obtained from $(T_c, \mathcal{X}_{T_c})$ by deleting from each bag all vertices not in $V(G_i)$, and then adding x to the bag indexed by each node on the path between c and some node whose bag originally contained x. It now follows from Lemma 6.4 that there exists a width-3 simple tree decomposition $(T_i, r_i, \mathcal{X}_i)$ of G_i, where $V_{\text{term}}(G_i) = V'_i$.

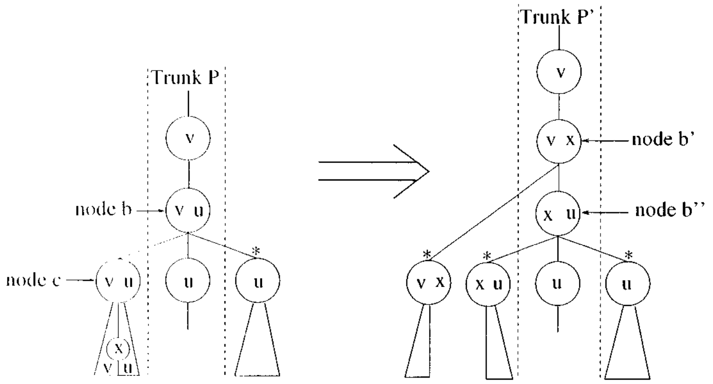

Fig. 3. Promoting a vertex ("*" indicates there may be multiple similar subtrees).

To construct $(T', r', \mathcal{X}')$, we split b (Definition 3.8) into two nodes b', b'' with b' the parent of b''. Let P' be the trunk (in T') thus derived from P. Let $X_{b'}$ be $X_b \cup \{x\} - \{u\}$; let $X_{b''}$ be $X_b \cup \{x\} - \{v\}$; and let each child $c' \in V(T \backslash T_c)$ of b become a child of b''. To complete the construction, let each r_i $(1 \leq i \leq \ell)$ become a child of either b' or b'', depending upon whether v or u is contained in X_{r_i}. It is not difficult to verify that each node of T' satisfies properties P1, P2, and P3; and that each node of P' also satisfies P4 and P5. □

The next property will be used for encoding the axes of a pyramid in each trunk-graph $R \in \mathcal{R}$. For each axis $(A, \rightarrow)$, the induced subgraph $R_{[A]}$ consists of a collection of paths (by condition D4 of Definition 6.8). This property ensures there are enough cross-edges so that a CMS formula can determine the order of these paths within each axis.

PROPERTY 7.6. Each trunk-graph $R \in \mathcal{R}$ admits a simple path decomposition $(P, r, \mathcal{X}_P)$ for which the following statement is satisfied whenever $a \in V(P)$ is an ancestor of $b \in V(P)$: if there are two internally vertex-disjoint paths $H, H' \sqsubseteq R$ between the drop vertex of b and some vertex in X_a, then there is an internal vertex (say x) of either H or H' such that x is either a nondrop vertex of a or nonadd vertex of b, or x is adjacent to some vertex in $V(R) - V(H) - V(H')$.

We can enforce this property by "demoting" vertices from a centered trunk P of a simple tree decomposition (T, r, χ), as described below:

PROPOSITION 7.7. *Suppose that P is a centered trunk of a width-*3 *simple tree decomposition (T, r, χ), and that Proposition* 7.5 *cannot be applied to P. If $(P, r, \mathcal{X}_P)$ does not satisfy the statement of Property* 7.6, *then there exists a centered trunk P' of some width-*3 *simple tree decomposition of G, such that Proposition* 7.5 *cannot be applied to P', and $|X_{P'}| < |X_P|$.*

PROOF. Let R be the trunk-graph of P; and suppose $(P, r, \mathcal{X}_P)$ does not satisfy the statement of Property 7.6. So let $a \in V(P)$ be an ancestor of $b \in V(P)$ such that there are two internally vertex-disjoint paths H and H' between the drop vertex (say u) of b and some vertex (say w) in X_a; no internal vertex is adjacent to any vertex in $V(R) - V(H) - V(H')$; no internal vertex belongs to $V_{\mathrm{term}}(R)$; and no internal vertex belongs to a bag indexed by a descendant of b or an ancestor of a. Figure 4 shows the paths H and H' with vertex sequences $(u, x \cdots y, w)$ and $(u, x' \cdots y', w)$.

Let G' be the subgraph consisting of those components of $G \backslash \{u, w\}$ containing one or more vertices of $V(H) \cup V(H')$. Let the (say ℓ) components of G' be enumerated $G'_1, G'_2, \ldots, G'_\ell$. Using an argument similar to that used in the proof of Proposition 7.5, we obtain a width-3 simple tree decomposition $(T_i, r_i, \mathcal{X}_i)$ of the subgraph of G induced by $V(G'_i) \cup \{u, w\}$, with $\{u, w\}$ as the terminal set.

We can construct the tree decomposition $(T', r', \mathcal{X}')$ from (T, r, χ) as follows: We delete those subtrees $T_c \sqsubseteq T$ whose bags contain only vertices of $V(G') \cup \{u, w\}$; each such subtree is rooted at a child c of a node on the path (in T) between a and b. We

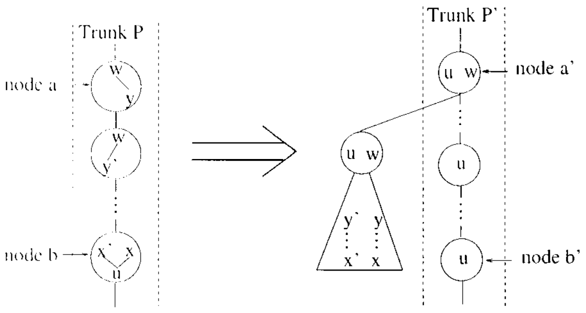

Fig. 4. Demoting vertices.

delete any vertices of G' from each bag of that path, and then add u to each of them. Now, let each r_i ($1 \le i \le \ell$) become a child of a. To complete the construction, we contract (Definition 3.7) each edge between a node $p \in V(P)$ and its parent whenever p no longer has a drop vertex. It is not difficult to verify that each resulting node satisfies properties P1–P5. Furthermore, this operation does not create any new instances where Proposition 7.5 can be applied—since there are two internally vertex-disjoint paths between u and w whose internal vertices now belong to bags indexed by descendants of the trunk P'. □

LEMMA 7.8. *G admits a width-*3 *simple tree decomposition for which the trunk-graph corresponding to some centered trunk satisfies Properties* 7.4 *and* 7.6.

PROOF. Let (T, r, χ) be a simple tree decomposition of G such that $P \sqsubseteq T$ is a centered trunk. Without loss of generality, assume neither Proposition 7.5 nor 7.7 can be applied to P. It follows immediately that the trunk-graph (say R) of P satisfies Property 7.6.

CLAIM. *Let v_1 be the drop vertex of the leaf of P. For $v \in V(R)$, there is a path $H \sqsubseteq R$ between v_1 and v that satisfies conditions* H1 *and* H2 (*of Property* 7.4).

Since v_1 is not a terminal, it follows that there is a path satisfying H1 and H2 between any pair of vertices in R. To prove the lemma, then, we need only prove the above claim.

Since each vertex of the leaf bag is adjacent to v_1, the claim is satisfied for all those vertices. Suppose the claim is false, and let b be the closest node to the leaf such that a vertex $v \in X_b$ violates the claim. By Fact 6.2, the drop vertex (say u) of b satisfies the claim. Now v would also satisfy the claim if there were an edge of G between u and v. It follows from property P3 (Definition 6.1) that b has a unique child $c \in V(T \setminus P)$ for which $u, v \in X_c$. Since Proposition 7.5 cannot be applied, there is no cut-vertex separating u from v in the subgraph induced by $X_{T_c} - X_b \cup \{u, v\}$. Therefore, by Lemma 2.3, there are two internally vertex-disjoint paths between u and v for which each internal vertex belongs to $X_{T_c} - X_b$ (contradicting the supposition that v violates the claim). □

THEOREM 7.9. *If G is a 2-connected partial 3-tree, then some width-3 simple tree decomposition of G admits a trunk hierarchy $\mathcal{R}$ such that each trunk-graph $R \in \mathcal{R}$ satisfies Properties* 7.2, 7.4, *and* 7.6.

PROOF. Let (T, r, χ) be a width-3 simple tree decomposition of G. Without loss of generality (by Lemma 7.8), assume that T contains a centered trunk whose trunk-graph satisfies Properties 7.4 and 7.6. Assume recursively, for each child c of each node of this centered trunk, that $(T_c, c, \mathcal{X}_{T_c})$ also contains a centered trunk whose trunk-graph satisfies Properties 7.4 and 7.6. Let $\mathcal{R}$ be the collection of these trunk-graphs. To complete the proof, we need only show that the vertices of each trunk-graph can be placed in a sequence to satisfy Property 7.2. So suppose $R \in \mathcal{R}$ is the trunk-graph of $P \sqsubseteq T$. Let the first vertex v_1 of the sequence be the drop vertex of the leaf of P. Let the remaining vertices of the leaf bag come next in the sequence; each such vertex is adjacent to v_1, so it satisfies condition C1 (of Property 7.2). We complete the sequence by induction on nodes of P.

Suppose $b, c \in V(P)$ such that b is the parent of c; and assume we have already placed the vertices of X_c in the sequence. Since G is 2-connected, there are at most two vertices in $X_b - X_c$; let these add vertices come next in the sequence (in either order). We need only show that each such vertex (say v_i) satisfies condition C1, C2, or C3. If v_i is adjacent to the drop vertex (say v_j), then v_i satisfies C1. Otherwise, it follows from property P3 that b has a child c' for which $v_j, v_i \in X_b \cap X_{c'}$; and $X_b \cap X_{c'}$ yields the terminal set of some child of R. If $X_b \cap X_{c'}$ contains only the vertices v_i and v_j, then condition C2 is satisfied. Otherwise, by property P1, $X_b \cap X_{c'}$ contains at most one other vertex (say $v_{j'}$). By property P4, $v_{j'} \in X_c$; so $j' \leq i - 1$. By property P5, either $v_i \in V_{\text{term}}(R)$, or v_i is adjacent (in R) to some vertex (say v) of $R \backslash X_{P_b}$. Such a vertex v is an add vertex of some ancestor of b; thus P5 can be applied recursively to show that there is a path in $G \backslash \{v_j, v_{j'}\}$ between v_i and some terminal in $V_{\text{term}}(G)$. Therefore, condition C3 is satisfied. □

8. Encoding a Trunk Hierarchy.

In this section we develop CMS predicates to encode the vertex and edge sets for each trunk-graph in a trunk hierarchy of a 2-connected partial 3-tree G. In Section 9 we show that a pyramid can be encoded in each trunk-graph; and in Section 10 we combine these results to obtain CMS predicates describing a tree decomposition of G.

Throughout this section $G = (V, E)$ is a 2-connected partial 3-tree with either two or three terminals, and $\mathcal{R}$ is a trunk hierarchy admitted by a simple tree decomposition of G. Without loss of generality (by Theorem 7.9), we assume each trunk-graph $R \in \mathcal{R}$ satisfies Properties 7.2 and 7.4. Note that each vertex in $V(G) - V_{\text{term}}(G)$ is a nonterminal vertex of exactly one trunk-graph in $\mathcal{R}$.

DEFINITION 8.1. If v is a nonterminal vertex of G, then $R(v)$ denotes the unique trunk-graph in $\mathcal{R}$ such that v is a nonterminal vertex of $R(v)$.

By Lemma 4.9, a CMS predicate can encode any constant amount of information pertaining to the role of each vertex v in the trunk-graph $R(v)$. This allows a nonterminal vertex (say v_1) to be designated for each trunk-graph. In order to identify the other vertices of $R(v_1)$ inductively, we use a (nonproper) vertex coloring:

PROPOSITION 8.2. *$V(G)$ can be partitioned into* 13 color classes *such that the nonterminal vertices of each trunk-graph $R \in \mathcal{R}$ belong to a common color class (say C); no terminal of R belongs to C; and if $t \in V_{\text{term}}(R) - V_{\text{term}}(G)$, then no vertex of $R(t)$ belongs to C.*

PROOF. First, we arbitrarily color each terminal of G. Then we repeatedly take some trunk-graph $R \in \mathcal{R}$ whose terminals are all colored, and we color its nonterminal vertices. For each $t \in V_{\text{term}}(R)$, there are at most four color classes that cannot be used for this. □

We identify the vertices of a trunk-graph R in an order $v_1, v_2, \ldots, v_{|V(R)|}$ given by Property 7.2. Each vertex v_i ($2 \leq i \leq |V(R)|$) is identified with the help of a unique edge incident to an already identified nonterminal vertex v_j (where $j \leq i - 1$). Property 7.2 gives three different conditions whereby v_i may be identified: For condition C1, $\{v_i, v_j\}$ is an edge of G; so Lemma 4.9 allows us to encode that these endpoints belong to the same trunk-graph. Otherwise, both v_i and v_j are terminals of some child (say R') of $R(v_1)$; in this case, v_i will be identified with the help of a vertex (say v) that is adjacent to v_j (as illustrated in Figure 5). We need the following lemma to show that v_i is the "first" correctly colored vertex that separates v from the terminals of G.

LEMMA 8.3. *Suppose a trunk-graph $R \in \mathcal{R}$ has a child $R' \in \mathcal{R}$; and let $v_i \in V_{\text{term}}(R')$, $v_j \in V_{\text{term}}(R') - V_{\text{term}}(R)$. Let U be the union of the nonterminal vertices over R' and all descendants of R'; and let G' be the subgraph of G induced by $U \cup \{v_i\}$. If $u, v \in U$ such that $\{v_j, v\} \in E(G)$ and u belongs to the same color class as v_i, then $G' \backslash u$ contains a path between v and v_i.*

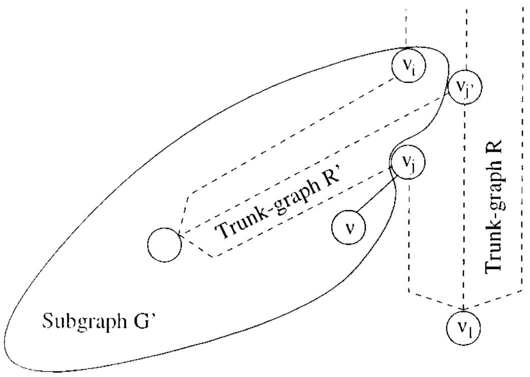

Fig. 5. G' is not cut by any other vertex with the color of v_i.

PROOF. Suppose $\{v_j, v\} \in E(G)$, for some vertex $v \in U$. Let $R_0 = R$ and $R_1 = R'$; and let $d \geq 1$ such that $R_d = R(v)$ and $R_{\ell-1}$ is the parent of R_ℓ $(1 \leq \ell \leq d)$. Choose terminals $t_\ell \in V_{\text{term}}(R_\ell)$ such that $t_1 = v_i$, and, for $2 \leq \ell \leq d$, $t_\ell \in V_{\text{term}}(R_\ell) - V_{\text{term}}(R_{\ell-1})$.

Since $\{v_j, v\} \in E(G)$, it follows that v_j is a terminal of each R_ℓ $(1 \leq \ell \leq d)$. By Proposition 8.2, the color of v_i is distinct from the color of the nonterminal vertices of each R_ℓ $(1 \leq \ell \leq d)$. Suppose that $u \in U$ has the same color as v_i; so u is not a vertex of any R_ℓ $(0 \leq \ell \leq d)$. By Property 7.4, then, there is a path with endpoints $v, t_d \in V(R_d)$, such that each internal vertex is in $U - \{u\}$; and, for $1 \leq \ell \leq d-1$, there is a path with endpoints $t_{\ell+1}, t_\ell \in V(R_\ell)$, such that each internal vertex is in $U - \{u\}$. By concatenating these paths, we obtain the required path between v and v_i. $\square$

We now show how CMS predicates can determine the structure of $R(v_1)$, for a designated nonterminal vertex v_1. The vertex set of $R(v_1)$ is the minimal set (say V') containing v_1 such that if $v_j \in V'$ has the same color as v_1, and there is an edge (say e) incident to v_j, then certain other vertices are also in V'. The opposite endpoint of such an edge e is either another vertex of $R(v_1)$, or some nonterminal vertex v of a descendant of $R(v_1)$. In the latter case, another vertex v_i of $R(v_1)$ is found using Lemma 8.3.

LEMMA 8.4. *Binary CMS predicates* **trunk**, **trunk-edge**, *and* **term**$_j$ $(1 \leq j \leq 3)$ *can be existentially defined such that there is a subset A of $V(G) - V_{\text{term}}(G)$ containing exactly one nonterminal vertex of each trunk-graph in $\mathcal{R}$; and*

- **trunk**(v_1, V') *holds iff $v_1 \in A$, and V' is the vertex set of $R(v_1)$; and*
- **trunk-edge**(u, v) *holds iff $\{u, v\}$ is an edge of some trunk-graph in $\mathcal{R}$; and*
- *for $v_1 \in A$: if t is a terminal of $R(v_1)$, then* **term**$_j(v_1, t)$ *holds for a unique index j; and* **term**$_1(v_1, t) \vee$ **term**$_2(v_1, t) \vee$ **term**$_3(v_1, t)$ *holds only if t is a terminal of $R(v_1)$.*

PROOF. Let A be comprised of the first vertex of each trunk-graph in an order given by Property 7.2. Suppose V' is the vertex set of $R(v_1)$, for some $v_1 \in A$. To encode **trunk**(v_1, V'), we first identify a superset V'' of V', such that V'' does not contain any nonterminal vertex of any descendant of $R(v_1)$. Each vertex of V'' is identified inductively, using one of the three conditions of Property 7.2. Throughout this proof, $v_j \in V''$ is either a nonterminal vertex of $R(v_1)$, or an extra vertex in $V'' - V'$ that can be weeded out later. In either case, our CMS formula forces any vertex v_i to belong to V'' if it interacts with v_j according to one of the conditions of Property 7.2.

For each edge incident to v_j, we encode (by Lemma 4.8) whether its opposite endpoint also belongs to $R(v_j)$. Thus a vertex is identified for membership in V'' whenever condition C1 is satisfied. To identify a vertex v_i satisfying condition C2 or C3, we use any edge $\{v, v_j\}$ such that both v_i and v_j are terminals of $R(v)$. Let $R_1 \in \mathcal{R}$ be the child of $R(v_j)$ such that either $R_1 = R(v)$, or R_1 is an ancestor of $R(v)$. By Lemma 4.9, we can encode (for the edge $\{v, v_j\}$) whether R_1 has two or three terminals in total. If there are two, then condition C2 may be applied to identify a vertex v_i for membership in V''. If there are three, then condition C3 may be applied to identify v_i.

Condition C2: $V_{\text{term}}(R_1) = \{v_i, v_j\}$. If v_i happens to be a terminal of G, then the CMS formula can easily identify the correct terminal. Otherwise, since G is 2-connected, v_i is a cut-vertex of $G \backslash v_j$ that separates v from $V_{\text{term}}(G)$. By Lemma 8.3, if u is another cut-vertex of $G \backslash v_j$ that separates v from $V_{\mathrm{term}}(G)$, then the component of $G \backslash \{v_j, v_i\}$ that contains v is a proper subgraph of the component of $G \backslash \{v_j, u\}$ that contains v. Hence, a CMS formula can identify v_i as the cut-vertex that places v into a minimal component.

Condition C3: $V_{\mathrm{term}}(R_1) = \{v_i, v_j, v_{j'}\}$ *where* $j' \leq i - 1$. A CMS formula can encode the following: If $v_{j'} \in V''$, and v_i is a cut-vertex of $G \backslash \{v_j, v_{j'}\}$ separating v from $V_{\mathrm{term}}(G)$, placing v into a minimal component of $G \backslash \{v_i, v_j, v_{j'}\}$ (over all choices of v_i), then v_i also belongs to V''. Extra vertices (of $V'' - V'$) may be identified in this way if the vertex $v_{j'}$ is not chosen "correctly."

The vertex order of Property 7.2 provides an inductive argument that V'' contains each vertex of $R(v_1)$. The CMS formula can state that V'' is a minimal set satisfying the requirements described above. Hence, if $v_i \in V'' - V'$ was chosen by condition C2 or C3, then (by Property 7.4) v_i cannot be a nonterminal vertex of any descendant of $R(v_1)$, unless either v_j or $v_{j'}$ also is. The minimality of V'', then, prevents this from happening.

Now, to identify the set $V' \subseteq V''$, we ensure that if condition C3 is used to identify a vertex v_i, then it cuts $G \backslash \{v_j, v_{j'}\}$ such that v is placed in a minimal component over all choices of $v_{j'} \in V''$. It follows from Lemma 8.3 that exactly the vertices of $R(v_1)$ will be identified in this way.

To encode the predicates $\mathbf{term}_i(v_1, t)$, $1 \leq i \leq 3$, we note that each vertex of $R(v_1)$ is identified with a unique edge. Hence, that edge can encode (by Lemma 4.9) the index of any terminal it is used to identify. Then we encode that **trunk-edge**(u, v) holds iff either $\{u, v\}$ is an edge of G, or both u and v are terminals of a common trunk-graph. □

9. Encoding a Pyramid.

In this section we restrict our attention to any trunk-graph R in a trunk hierarchy of a 2-connected partial 3-tree G. We develop CMS predicates encoding the axes of some pyramid (Definition 6.8) in R. These predicates shall be defined over the universe $V(R) \cup E(R)$; and in Section 10 we use them to encode a fixed path decomposition of R over the universe $V(G) \cup E(G)$. Although R may contain edges that are not edges of G, we show (in Section 10) how such edges can be represented within a CMS formula.

Throughout this section R is a trunk-graph belonging to a trunk hierarchy of a 2-connected partial 3-tree. Assume that R satisfies Property 7.6, and let $(P, r, \mathcal{X}_P)$ be a width-3 simple path decomposition of R given by that property. Let $(A_i, \rightarrow_i)$, $1 \leq i \leq 3$, be the axes of a pyramid in R. Without loss of generality, we assume the apex of the pyramid is the drop vertex of the leaf of P.

CLAIM 9.1. *For each $b \in V(P)$, either b has no add vertex, or some add vertex of b belongs the the same axis as the drop vertex of b.*

PROOF. This can easily be enforced within the proof of Lemma 6.10. □

By Lemma 8.4, a CMS predicate can determine the vertex set $V(R)$, and can associate each terminal $t \in V_{\mathrm{term}}(R)$ with a distinct index j $(1 \leq j \leq 3)$. If $v \in V(R) - V_{\mathrm{term}}(R)$, then R is the only trunk-graph having v as a nonterminal vertex. So by Lemma 4.9, we can encode which set A_i $(1 \leq i \leq 3)$ contains each nonterminal vertex of R. Similarly, it

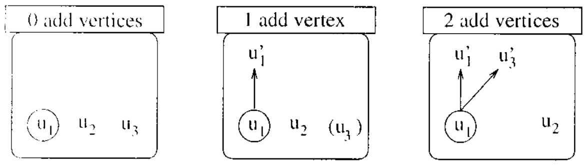

Fig. 6. The bags of a simple path decomposition (u_1 is the drop vertex; u'_1 and u'_3, when shown, are add vertices).

can be encoded which axis contains the terminal associated with each index. Therefore, we can use the sets $V_{\text{term}}(R)$, A_1, A_2, and A_3 to encode the orders "$\longrightarrow_i$" $(1 \le i \le 3)$ existentially.

Each nonleaf node of P has at most two add vertices. By property P3 (Definition 6.1), if u_1 is the drop vertex of $b \in V(P)$, then there is an edge of R between u_1 and each add vertex of b. By Claim 9.1, some such edge is axial (unless b has no add vertex). Figure 6 illustrates the three possible situations, depending on how many add vertices b has:

0. Hence, b has exactly two nonadd, nondrop vertices.
1. Hence, b has at least one nonadd, nondrop vertex (and possibly a second).
2. Hence, b has exactly one nonadd, nondrop vertex.

Each vertex in Figure 6 is named u_i or u'_i $(1 \le i \le 3)$; the subscript indicates that the vertex belongs to A_i. We assume without loss of generality that the drop vertex u_1 belongs to A_1. The figure shows edges directed from u_1 to each add vertex.

By property D4 (Definition 6.8), each set A_i $(1 \le i \le 3)$ induces a collection of paths in R:

Definition 9.2. For $1 \le i \le 3$, a *chain* in A_i is a component (say H) in the subgraph $R_{[A_i]}$. If H does not contain a terminal of R, and no vertex of H is adjacent to the apex, then H is said to be an *internal chain* in A_i.

By Lemma 4.8, a CMS predicate can encode the vertex order within each chain. If a chain is adjacent to the apex, then it precedes all other chains; if a chain contains a terminal, then it follows all other chains. It remains to be shown how a CMS predicate can determine the order of the internal chains in each axis.

Definition 9.3. For any vertex u of R, $\mathit{Add}(u)$ denotes the unique bag in $\mathcal{X}_P$ such that u is an add vertex of $\mathit{Add}(u)$. If u is a nonterminal vertex of R, then $\mathit{Drop}(u)$ denotes the unique bag in $\mathcal{X}_P$ such that u is the drop vertex of $\mathit{Drop}(u)$.

For the rest of this section, we may refer to bags (say X_a, X_b for $a, b \in V(P)$) using the notation of Definition 9.3: We then say that a vertex $v \in V(R)$ is an add (drop) vertex of X_a to mean that v is an add (drop) vertex of a. We write $X_b \prec X_a$ to mean that a is an ancestor of b; and we write $X_b \preceq X_a$ to mean that either $a = b$ or a is an ancestor of b.

DEFINITION 9.4. Let $1 \le i \le 3$; and suppose H is a chain in A_i. The first vertex of H (with respect to "$\rightarrow_i$") is called the *head* of H; and the last vertex of H is called the *tail* of H. If u is the head of H, then the drop vertex of $Add(u)$ is called the *source* of H.

In the leftmost panel of Figure 6, u_1 is the tail of some chain; in the rightmost panel, u_3' is the head of some chain, and u_1 is its source.

PROPOSITION 9.5. *A binary CMS predicate β can be existentially defined such that*

- *if $\beta(h, v)$, then h is the head of some chain, and $Add(h) \preceq Add(v)$; and*
- *if h and t are the head and tail (respectively) of some internal chain, then $\beta(h, v)$ is true for some nondrop vertex v of $Drop(t)$.*

PROOF. Using Lemma 4.8, let α, λ, μ be binary CMS predicates encoding edge directions:

$$\begin{aligned}
\alpha(u, v) &\equiv \text{"}\{u, v\} \text{ is an axial edge with } u \rightarrow_i v\text{" (for } i = 1, 2 \text{ or } 3);\\
\lambda(u, v) &\equiv \text{"}u \text{ is the source of some chain whose head is } v\text{"};\\
\mu(u, v) &\equiv \text{"}\{u, v\} \text{ is a cross edge and } Add(h) \preceq Add(v), \text{ where } h \text{ is the head}\\
&\qquad\qquad \text{of the chain containing } u.\text{"}
\end{aligned}$$

By Lemma 4.2, the transitive closure (denoted α^+) and the reflexive-transitive closure (denoted α^*) of α are also encodable. We now define β as follows:

$$\beta(h, v) \equiv (\exists s)\big(\lambda(s, h) \wedge (\alpha^+(s, v) \vee (\exists x, y)(\alpha^*(h, x) \wedge \mu(x, y) \wedge \alpha^*(y, v)))\big). \tag{9.6}$$

It is clear that $\beta(h, v)$ is true only if $Add(h) \preceq Add(v)$, where h is the head of some chain. To complete the proof, we need only show that if h_1 and u_1 are the head and tail (respectively) of some internal chain (say H_1), then $\beta(h_1, v)$ is true for some nondrop vertex v of $Drop(u_1)$. So suppose $\beta(h_1, v)$ is false for each nondrop vertex v of $Drop(u_1)$. By Claim 9.1, $Drop(u_1)$ has no add vertex. Without loss of generality, assume $u_1 \in A_1$; let $u_2 \in A_2$ and $u_3 \in A_3$ be the nondrop vertices of $Drop(u_1)$. For $2 \le j \le 3$, let H_j be the maximal axial path containing u_j, and let h_j be the head of H_j (see Figure 7).

Since $\beta(h_1, u_2)$ is false and $\beta(h_1, u_3)$ is false (by supposition), the source (say s) of H_1 cannot belong to H_2 or H_3. Hence, either $Add(h_1) \prec Add(h_2)$ or $Add(h_1) \prec Add(h_3)$. Without loss of generality, we assume $Add(h_1) \prec Add(h_2)$ and $Add(h_3) \prec Add(h_2)$. It follows that the source (say s_3) of H_2 is a vertex of H_3. Now, $Drop(s_3)$ contains exactly two nonadd vertices: i.e., s_3 and some vertex (say x_1) of H_1. Because the underlying graph is 2-connected, there are two vertex-disjoint paths between $\{s_3, x_1\}$ and $\{u_2, u_3\}$. Hence, there is a cross edge between a vertex (say x) of H_1 and a vertex (say y) of either H_2 or H_3, where $x_1 \rightarrow_1^* x$ and either $h_2 \rightarrow_2^* y$ or $s_3 \rightarrow_3^+ y$. Thus $\mu(x, y)$ is true. Therefore, either $\beta(h_1, u_2)$ or $\beta(h_1, u_3)$ is true (a contradiction). □

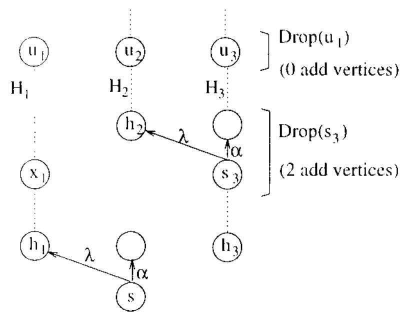

Fig. 7. Proof of Proposition 9.5.

PROPOSITION 9.7. *For $1 \le i \le 3$, a binary CMS predicate γ can be existentially defined such that*

- *if $\gamma(h, v)$, then h is the head of some chain in A_i, and $h \to_i^* v$; and*
- *if H and H' are internal chains in A_i such that H' immediately follows H, then $\gamma(h, v)$ is true for some vertex v of H', where h is the head of H.*

PROOF. Without loss of generality, we restrict our attention to the case of $i = 1$. We begin by using the predicate β (of Proposition 9.5) to encode a subset γ' of γ:

$$\gamma'(h, v) \equiv (\exists u)(\beta(h, u) \wedge \text{“}\{u, v\} \in E\text{”} \wedge (h, v \in A_1)).$$

It follows easily from Proposition 9.5 that the predicate γ' is consistent with the first statement itemized above. To complete the proof, it may be necessary to define additional ordered pairs in γ, so that the second itemized statement is also satisfied.

Suppose H_1 and H_1' are internal chains in A_1 such that H_1' immediately follows H_1. Let h_1, u_1, h_1', t_1' be the head of H_1, tail of H_1, head of H_1', tail of H_1', respectively (see Figure 8). Say $Drop(u_1) = \{u_1, u_2, u_3\}$, where $u_2 \in A_2$ and $u_3 \in A_3$; and let H_2 and

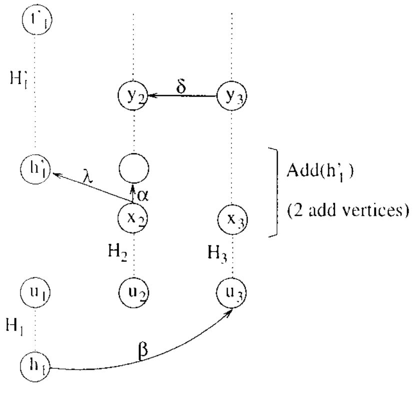

Fig. 8. Proof of Proposition 9.7.

H_3 be the chains containing u_2 and u_3 (respectively). Since the underlying graph is 2-connected, $Add(h'_1)$ contains two nonadd vertices, say $x_2 \in V(H_2)$ and $x_3 \in V(H_3)$. Without loss of generality (by Proposition 9.5), we assume $\beta(h_1, u_3)$ is true. By (9.6), if $u_3 \to_3^* x$, then $\beta(h_1, x)$ is also true. If any such vertex x is adjacent to a vertex (say v) of H_1, then $\gamma'(h_1, v)$ is true, and the proof is complete.

CLAIM. *If $\gamma'(h_1, v)$ is false for each $v \in V(H'_1)$, then there exist vertices $y_2 \in V(H_2)$ and $y_3 \in V(H_3)$ such that*

$$\alpha^+(x_2, y_2);\ \alpha^*(x_3, y_3);\ \{y_2, y_3\} \in E(R);\ \textit{and}\ Add(h'_1) \preceq Add(y_2) \preceq Drop(t'_1).$$

Whenever γ' is insufficient to satisfy the second itemized statement of the proposition, we choose some such edge $\{y_2, y_3\}$. Using Lemma 4.8, we encode a binary predicate δ to direct these edges $\delta(y_3, y_2)$. Now, we encode the following subset γ'' of the γ relation:

$$\gamma''(h_1, h'_1) \equiv (\exists y_3, y_2, x_2)\big(\beta(h_1, y_3) \wedge \delta(y_3, y_2) \wedge \alpha^+(x_2, y_2) \wedge \lambda(x_2, h'_1) \\ \wedge \neg(\exists s)(\text{“}s \text{ is between } x_2 \text{ and } y_2\text{”} \wedge \lambda(s, v) \wedge v \in A_1)\big).$$

The required predicate $\gamma(h, v)$ can be defined as $\gamma'(h, v) \vee \gamma''(h, v)$. To complete the proof, we need only prove the above claim. So suppose it is false, and yet $\beta(h_1, u_3)$ is true. Thus, there is no cross edge between H_3 and H'_1; and x_2 is the source of H'_1.

Case 1: *$Drop(t'_1)$ contains a vertex of H_2 and a vertex of H_3.* Since the underlying graph is 2-connected, it follows that t'_1 is adjacent to some vertex x of H_2 such that $x_2 \to_2^+ x$. Without loss of generality, assume that t'_1 is not adjacent to any vertex that follows x (in the order “$\to_2$”). Therefore, by Property 7.6, there is an edge between H_3 and some internal vertex of one of either $[x_2 \to_2^+ x, t'_1]$ or $[x_2, h'_1 \to_1^* t'_1]$ (a contradiction).

Case 2: *The tail* (*say t_2*) *of H_2 is not a terminal, and $Drop(t_2)$ contains a vertex of H'_1 and a vertex of H_3.* Since the claim is false, there is no edge between t_2 and H_3. It follows that t_2 is adjacent to some vertex (say v'_1) of H'_1. Hence, by Property 7.6, there is an edge between H_3 and some internal vertex of $[x_2 \to_2^+ t_2]$ or $[x_2, h'_1 \to_1^* v'_1, t_2]$ (a contradiction).

Case 3: *The tail* (*say t_3*) *of H_3 is not a terminal, and $Drop(t_3)$ contains a vertex of H'_1 and a vertex of H_2.* Since the claim is false, there is no edge between t_3 and any vertex y_2 such that $x_2 \to_2^+ y_2$. Hence, t_3 is adjacent to some vertex of H'_1 (a contradiction). □

We have now established the following:

LEMMA 9.8. *There is a pyramid in each trunk-graph in $\mathcal{R}$ for which each axis $(A_i, \to_i)$, $1 \le i \le 3$, is existentially encodable by a CMS predicate.*

10. 2-Connected Partial 3-Trees. In this section we develop CMS predicates to describe (Definition 4.5) a fixed tree decomposition of a 2-connected partial 3-tree G. To do this, we use the trunk hierarchy $\mathcal{R}$ constructed in Section 7. We have shown in Section 8 that CMS predicates can encode the structure of each trunk-graph in $\mathcal{R}$; and in Section 9 we showed how to encode a pyramid in each one. In this section we show how such a pyramid enables CMS predicates to describe a path decomposition of the corresponding trunk-graph. The collection of these path decompositions can then easily be assembled into a tree decomposition of G

Throughout this section $G = (V, E)$ is a 2-connected partial 3-tree with either two or three terminals; and $\mathcal{R}$ is a trunk hierarchy admitted by a width-3 simple tree decomposition of G. Without loss of generality (by Theorem 7.9), we assume each trunk-graph in $R \in \mathcal{R}$ has Properties 7.2, 7.4, and 7.6. In Section 9 we used Property 7.6 to show that the axes of a pyramid in R can be encoded in CMS logic over the universe $V(R) \cup E(R)$:

LEMMA 10.1. *Any CMS-encodable predicate over the universe $V(R) \cup E(R)$ can be expressed over the universe $V \cup E$.*

PROOF. Since $V(R) \subseteq V$, we need only show how to represent edges of $E(R) - E(G)$, and how to represent sets of edges. Then, any predicate over $V(R) \cup E(R)$ can be expressed as a disjunction of predicates over $V \cup E$.

A CMS predicate can encode that v is the apex of a child of R iff $V(R)$ contains each vertex t for $\mathbf{term}_j(v, t)$, $1 \leq j \leq 3$ (see Lemma 8.4). Each pair of these terminals are the endpoints of an edge in $E(R) - E(G)$. The apex v can be used to represent the (at most three) edges between its terminals; and each such edge can be distinguished by its index j.

To represent a set $E' \subseteq E(R)$, we use a subset of $E(G)$ and three vertex subsets: The edge subset contains the edges of $E' \cap E(G)$; and the vertex subsets contain apices representing the edges of $E' - E(G)$. Each such vertex subset corresponds to a distinct pair of indices j, j' $(1 \leq j < j' \leq 3)$. A vertex v belongs to this set iff v is the apex of some child of R, and $\mathbf{term}_j(v, t) \wedge \mathbf{term}_{j'}(v, t')$ for some edge $\{t, t'\} \in E' - E(G)$. $\square$

Each axis of a pyramid in R gives part of an elimination order on $V(R)$. By interleaving these orders in a fixed manner, we obtain a path decomposition of R for which the leaf bag contains the apex and the first vertex of each axis; each nonleaf bag contains the k maximal vertices of its child's bag, as well as the immediate successor of one of them. Property D5 (Definition 6.8) guarantees that the orders can be interleaved in this way. The bags are well-defined if we impose an order on the axes, and adopt the convention that we inductively advance along the first axis whenever possible, otherwise the second axis if possible, and otherwise the third.

LEMMA 10.2. *Each trunk-graph in $\mathcal{R}$ admits a width-*3 *path decomposition that can be described by existentially defined CMS predicates* **Bag** *and* **Parent**.

PROOF. By Lemma 9.8, the axes $(A_i, \rightarrow_i)$, $1 \leq i \leq 3$, can be encoded for some pyramid in $R \in \mathcal{R}$. By Lemma 4.2, the transitive closures "$\rightarrow_i^+$" can also be encoded. We assume that each A_i is nonempty—for otherwise, a simplification of the argument carries through. We explain how to define the **Bag** predicate for a path decomposition in which each bag has cardinality four, and adjacent bags intersect in exactly three vertices. So each nonroot bag contains a unique drop vertex—which becomes

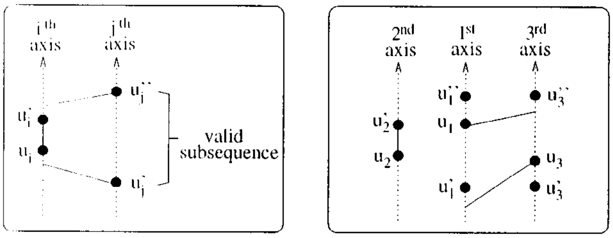

Fig. 9. Choosing a bag of the CMS-encoded path decomposition.

its witness. The witness of the root bag can be chosen arbitrarily from its (one or two) drop vertices.

To identify the leaf bag, we encode the fact that $\mathbf{Bag}(v_1, X)$ holds when X contains the designated apex v_1 as well as the first vertex in each axis. Each other bag X contains a unique pair of vertices $u_i, u_i' \in A_i$ $(1 \le i \le 3)$ such that $u_i \rightarrow_i u_i'$. The other two vertices $u_j \in A_j$ $(1 \le j \le 3;\ j \ne i)$ in X can be identified as follows: Let $u_j' \in A_j$ be the last vertex (in the order "$\rightarrow_j$") thatis adjacent to u_i or any vertex that precedes u_i in the ith axis; if there is no such adjacency, then we let u_j' be the first vertex of the jth axis by default. Let $u_j'' \in A_j$ be the first vertex that is adjacent to u_i' or any vertex that follows u_i' (or, by default, let u_j'' be the last vertex of the jth axis). It follows from property D5 (Definition 6.8) that $u_j' \rightarrow_j^* u_j''$; and we refer to the vertices between u_j' and u_j'' as the *valid subsequence* (see Figure 9). It is clear that the vertex $u_j \in X$ must belong to the valid subsequence: otherwise, there would be a cross edge incident to either u_j' or u_j'' without both endpoints in a common bag.

In the case that no vertex of the valid subsequence is incident to a cross edge, then *any* vertex in that subsequence can be chosen as u_j. To effect the precedence convention among the axes, we choose u_j as close as possible to u_j'', if $j < i$, and choose u_j as close as possible to u_j', if $j > i$. It is not hard to formalize this approach in CMS logic.

Now, it is easy to encode the **Parent** predicate: if $\mathbf{Bag}(c, X)$ and $\mathbf{Bag}(p, X')$ hold, where $X \ne X'$, then $\mathbf{Parent}(p, c)$ holds iff X' contains all three maximal vertices of X. □

The right-hand panel of Figure 9 illustrates the axes of a pyramid in a given partial 3-path. In this example, we have $u_2 \rightarrow_2 u_2'$. As explained in the proof of Lemma 10.2, the cross edges between the second and jth axis (for $j = 1, 3$) are used to identify the valid subsequence $[u_j', \ldots, u_j'']$. To identify the vertices u_1, u_3 for the bag $\{u_1, u_2, u_2', u_3\}$, a CMS formula need only consider the cross edges between the first and third axes. >From the valid subsequence of the first axis, u_1 is selected as close as possible to u_1'', under the constraint that there be no cross edge between a vertex preceding u_1 and a vertex following u_3'. From the valid subsequence of the third axis, u_3 is selected as close as possible to u_3', under the constraint that there be no cross edge between a vertex preceding u_1' and a vertex following u_3. By property D5 of a pyramid (Definition 6.8), there exists no cross edge between a vertex preceding u_1' and a vertex following u_3''.

LEMMA 10.3. *Any* 2*-connected partial* 3*-tree admits a width-*3 *tree decomposition that can be described by existentially defined CMS predicates* **Bag** *and* **Parent**.

PROOF. Suppose G is a 2-connected partial 3-tree; and let $\mathcal{R}$ be a trunk hierarchy given by Theorem 7.9. We use Lemma 10.2 to encode a path decomposition for each trunk-graph. To extend this to a tree decomposition of G, we need only encode parents for the roots of all but one of those path decompositions. By Lemma 8.4, a CMS predicate can determine the terminal set of each $R \in \mathcal{R}$. If $V_{\text{term}}(R) = V_{\text{term}}(G)$, then R is the root of the trunk hierarchy. Otherwise, the parent of R is the unique trunk-graph $R' \in \mathcal{R}$ for which $V_{\text{term}}(R) \subseteq V(R')$ and $V_{\text{term}}(R) \not\subseteq V_{\text{term}}(R')$. Thus $\mathbf{Parent}(p, c)$ can be encoded for c the witness of the root of the path decomposition of R, and p the witness of the closest node to the root of the path decomposition of R' such that $V_{\text{term}}(R) \subseteq X$ where $\mathbf{Bag}(p, X)$ is true. $\square$

11. Partial 3-Trees.

In Section 10 we established that CMS predicates can describe (Definition 4.5) a width-3 tree decomposition of any 2-connected partial 3-tree. Thus each block (Definition 2.4) of a partial 3-tree has a CMS-definable width-3 tree decomposition. In this section we use that result to draw the conclusion that definability equals recognizability of partial 3-trees.

LEMMA 11.1. *If each block of a graph has a CMS-definable width-k tree decomposition, then the graph itself has a CMS-definable width-k tree decomposition.*

PROOF. First, suppose G is a *connected* graph with ℓ distinct blocks, $G_1, G_2, \ldots, G_\ell$, each of which has a CMS-definable width-k tree decomposition. No pair of these blocks may intersect in more than one vertex—otherwise their union would also be 2-connected. Hence, the blocks can be arranged in a tree such that G_i is the parent of G_j only if $|V(G_i) \cap V(G_j)| = 1$. The root block of this tree can be existentially encoded in CMS logic (Definition 4.3) by providing a parameter to contain the vertex set of that block. Furthermore, it is easy for a CMS predicate to encode whether any vertex subset induces a block of G; hence, the CMS-definable width-k tree decomposition of each block can be determined independently. Without loss of generality, assume that if a block G_i is the parent of a block G_j, then the vertex in $V(G_i) \cap V(G_j)$ is a designated terminal in $V_{\text{term}}(G_j)$; this terminal will be contained in the root bag of the CMS-definable tree decomposition of G_j. Therefore, to encode a width-k tree decomposition of G, we need only encode a parent for the root of the tree decomposition of each nonroot block G_j. This parent can be encoded as the closest node to the root of the tree decomposition of G_i (i.e., the parent of G_j) such that the vertex of $V(G_i) \cap V(G_j)$ is contained in the corresponding bag.

Now, suppose G is a (possibly disconnected) graph for which each block has a CMS-definable width-k tree decomposition. A width-k tree decomposition can be encoded for each component (as above). We then choose the root (say r) of any one of these tree decompositions to be the root of a tree decomposition of G. CMS predicates can easily encode that each other root becomes a child of r. $\square$

Lemmas 10.3 and 11.1 provide the following:

COROLLARY 11.2. *Any partial* 3-*tree admits a width*-3 *rooted tree decomposition that can be described by existentially defined CMS predicates*.

Combining the above corollary with Lemmas 4.7 and 5.4 and Theorem 5.3, we can now conclude the following:

THEOREM 11.3. *Definability equals recognizability of partial* 3-*trees*.

12. *k*-Connected Partial *k*-Trees. We now generalize the results of this paper to show that definability equals recognizability of k-connected partial k-trees. To do this, we need only show that CMS predicates can describe a width-k rooted tree decomposition of any such graph. The results of Section 6 show that any partial k-tree can be decomposed into a trunk hierarchy (Definition 7.1). Some of the results of Sections 7, 8, and 10 need to be generalized.

Throughout this section G is a k-connected partial k-tree; and $\mathcal{R}$ is a trunk hierarchy admitted by a simple width-k tree decomposition of G. Since each trunk-graph has k terminals, Property 7.2 can be simplified to the following:

PROPERTY 12.1. The vertices of each trunk-graph $R \in \mathcal{R}$ can be ordered $v_1, v_2, \ldots, v_{|V(R)|}$ such that, for each $i = 2, 3, \ldots, |V(R)|$, there is a nonterminal vertex v_j of R (where $1 \leq j \leq i - 1$) for which at least one of the following two conditions is satisfied:

C1. v_i and v_j are adjacent (in G).
C3. R has a child $R' \in \mathcal{R}$ for which $v_i, v_j \in V_{\text{term}}(R')$, and $j' \leq i$ for each $v_{j'} \in V_{\text{term}}(R')$.

In Section 7 we enforced this property by choosing a *centered* trunk (Proposition 7.3) in a simple width-k tree decomposition (T, r, χ) of G. However, now, since G is k-connected, we have $|X_a \cap X_b| \geq k$ whenever a and b are adjacent nodes of T. It follows that every trunk in T is a centered trunk.

Property 7.4 can be enforced by a straightforward generalization of Proposition 7.5. Property 7.6 is not needed in the case of 3-*connected* partial 3-trees. Hence, a simplification of the proof of Theorem 7.9 provides the following:

THEOREM 12.2. *If G is a k-connected partial k-tree, then some width-k simple tree decomposition of G admits a trunk hierarchy $\mathcal{R}$ such that each trunk-graph $R \in \mathcal{R}$ satisfies Properties* 12.1 *and* 7.4.

The vertex set and edge set of each trunk-graph in $\mathcal{R}$ can now be encoded by CMS predicates (as in Section 8). Proposition 8.2 can easily be generalized as follows:

PROPOSITION 12.3. *$V(G)$ can be partitioned into* $4k + 1$ color classes *such that the nonterminal vertices of each trunk-graph $R \in \mathcal{R}$ belong to a common color class* (*say C*); *no terminal of R belongs to C; and if $t \in V_{\text{term}}(R) - V_{\text{term}}(G)$, then no vertex of $R(t)$ belongs to C.*

The proof of Lemma 8.3 remains valid in the case of k-connected partial k-trees (provided $k \geq 2$). The proof of Lemma 8.4 can be easily generalized, to give the following:

LEMMA 12.4. *Binary CMS predicates* **trunk**, **trunk-edge**, *and* $\mathbf{term}_j$ $(1 \leq j \leq k)$ *can be existentially defined such that there is a subset A of $V(G) - V_{\text{term}}(G)$ containing exactly one nonterminal vertex of each trunk-graph in $\mathcal{R}$; and*

- $\mathbf{trunk}(v_1, V')$ *holds iff $v_1 \in A$, and V' is the vertex set of $R(v_1)$; and*
- $\mathbf{trunk\text{-}edge}(u, v)$ *holds iff $\{u, v\}$ is an edge of some trunk-graph in $\mathcal{R}$; and*
- *for $v_1 \in A$: if t is a terminal of $R(v_1)$, then $\mathbf{term}_j(v_1, t)$ holds for a unique index j; and $\mathbf{term}_1(v_1, t) \vee \cdots \vee \mathbf{term}_k(v_1, t)$ holds only if t is a terminal of $R(v_1)$.*

We do not need the results of Section 9 to encode the axes of a pyramid in R, because each axis consists of a path between the apex and a distinct terminal: Lemma 9.8 becomes a trivial consequence of property D4 (Definition 6.8). Now, the proof of Lemma 10.2 can be easily generalized to prove the following:

LEMMA 12.5. *Any k-connected partial k-tree admits a width-k tree decomposition that can be described by existentially defined CMS predicates* **Bag** *and* **Parent**.

Using Lemmas 4.7 and 5.4 and Theorem 5.3, we can draw the following conclusion:

THEOREM 12.6. *Definability equals recognizability of k-connected partial k-trees.*

13. Conclusion.

This paper has established that CMS-definability is a necessary and sufficient condition for a subclass of the partial 3-trees (or k-connected partial k-trees) to be recognized by a finite-state tree automaton. It was known that CMS-definability is sufficient for a subclass of the partial k-trees (for any k) to be recognized in this way, but it was an open question whether CMS-definability is necessary (for $k \geq 4$). Very recently, this question was resolved by Lapoire [21] who showed the equivalence of definability and recognizability for *any* subclass of the partial k-trees. In this paper we used a general strategy which may independently provide another proof of necessity—by showing CMS logic can encode a tree decomposition of any partial k-tree G. We showed how this can be done in the case of $k = 3$, and in the case that G is k-connected.

The general strategy to encode a tree decomposition is first to decompose the partial k-tree G into a collection of *simple* partial k-paths. These partial k-paths cover the vertex set of G; and the union over their edge sets is a superset of the edge set of G. Moreover, we may assume without loss of generality (by Lemma 11.1) that G is 2-connected. Encoding a tree decomposition of G is thus reduced to the following two tasks:

1. Determine which vertices belong to each of the (2-connected) simple partial k-paths.
2. Encode a path decomposition for each of the simple partial k-paths. (These path decompositions can then be easily assembled into a tree decomposition of G.)

We have shown how the first task can be implemented (if $k \leq 3$ or G is k-connected) by inductively identifying the vertices of each partial k-path (say R). In these cases, $V(R)$ can be ordered $v_1, v_2, \ldots, v_{|V(R)|}$ such that each v_i ($2 \leq i \leq |V(R)|$) satisfies some logical condition relative to the set $\{v_1, v_2, \ldots, v_{i-1}\}$: For some v_j ($1 \leq j \leq i-1$) in this set, v_i can be uniquely identified by a CMS predicate $\Phi(v_j, v_i)$. Furthermore, the first vertex v_1 of the order does not belong to any other partial k-path in the decomposition. Thus $V(R)$ is encoded as the minimal set containing v_1 and any vertex v_i such that $\Phi^*(v_1, v_i)$, where Φ^* is the transitive closure Φ. If this approach is generalized in a straightforward way, then instead of identifying the vertices one at a time, we would need to identify up to $\min\{\lfloor k \rfloor/2, k - \ell + 1\}$ vertices at a time—for an ℓ-connected partial k-tree. For $k \geq 4$ and $\ell \leq k - 1$, this quantity is greater than 1; so it becomes much more difficult for a CMS formula to determine the vertices of such a group.

We have shown how the second task can be implemented for 2-connected partial 3-trees (this could be generalized to $(k-1)$-connected partial k-trees). This completes the proof that recognizability implies CMS-definability for partial 3-trees, because the 2-connected blocks of an arbitrary partial 3-tree G can be handled separately in this way, and then the resulting collection of tree decompositions can be assembled together. Recently, Kabanets [18] showed that CMS logic can encode a fixed path decomposition of *any* partial k-path: i.e., the second task enumerated above can be done for any k.

References

[1] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. *BIT*, 25:2–33, 1985.

[2] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8:277–284, 1987.

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

[4] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23:11–24, 1989.

[5] M.W. Bern, E.L. Lawler, and A.L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[6] H.L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Proc.* 15*th ICALP*, pages 105–119. Lecture Notes in Computer Science, volume 317. Springer-Verlag, Berlin, 1988.

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

[8] B. Bollobás. *Extremal Graph Theory*. Academic Press, London, 1978.

[9] R.B. Borie, R.G. Parker, and C.A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.

[10] B. Courcelle. On context-free sets of graphs and their monadic second-order theory. In *Proc*. *International Workshop on Graph Grammars*, pages 133–146. Lecture Notes in Computer Science, volume 291. Springer-Verlag, Berlin, 1987.

[11] B. Courcelle. Graph rewriting: an algebraic and logic approach. In J. van Leeuwen, editor, *Handbood of Theoretical Computer Science*, volume B, pages 193–242. Elsevier, Amsterdam, 1990.

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

[13] B. Courcelle. The monadic second-order logic of graphs. V. On closing the gap between definability and recognizability. *Theoret*. *Comput*. *Sci*., 80:153–202, 1991.

[14] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoret*. *Comput*. *Sci*., 109:49–82, 1993.

[15] M.R. Garey and D.S. Johnson. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[16] A. Gupta, D. Kaller, and T. Shermer. Linear-time algorithms for partial k-tree complements. *Algorithmica*, this issue.

[17] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory*, *Languages*, *and Computation*. Addison-Wesley, Reading, MA, 1979.

[18] V. Kabanets. Recognizability equals definability for partial k-paths. In *Proc*. 24*th ICALP*, pages 805–815. Lecture Notes in Computer Science, volume 1256. Springer-Verlag, Berlin, 1997.

[19] D. Kaller. Monadic Second-Order Logic and Linear-Time Algorithms for Graphs of Bounded Treewidth. Ph.D. thesis, Simon Fraser University, Burnaby, B.C., Canada, 1996.

[20] D. Kaller. Definability equals recognizability of partial 3-trees. In *Proc*. 22*nd WG*, pages 239–253. Lecture Notes in Computer Science, volume 1197. Springer-Verlag, Berlin, 1997.

[21] D. Lapoire. Recognizability equals monadic second-order definability for sets of graphs of bounded tree-width. In *Proc*. 15*th STACS*, pages 618–629. Lecture Notes in Computer Science, volume 1373. Springer-Verlag, Berlin, 1998.

[22] S. Mahajan and J.G. Peters. Regularity and locality in k-terminal graphs. *Discrete Appl*. *Math*., 54:229–250, 1994.

[23] K. Menger. Zur allgemeinen kurventheorie. *Fund*. *Math*., 10:96–115, 1927.

[24] D. Perrin. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 1–57. Elsevier, Amsterdam, 1990.

[25] N. Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[26] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. Assoc*. *Comput*. *Math*., 29:623–641, 1982.

[27] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbood of Theoretical Computer Science*, volume B, pages 133–191. Elsevier, Amsterdam, 1990.

[28] J. van Leeuwen. Graph algorithms. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 527–631. Elsevier, Amsterdam, 1990.