MONADIC SECOND-ORDER LOGIC AND LINEAR-TIME ALGORITHMS FOR GRAPHS OF BOUNDED TREEWIDTH

by

Damon Kaller

B.Sc. University of British Columbia 1986

M.Sc. Simon Fraser University 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

APPROVAL

Name:	Damon Kaller
Degree:	Doctor of Philosophy
Title of thesis:	Monadic Second-Order Logic and Linear-Time Algorithms for Graphs of Bounded Treewidth

Examining Committee: Dr. Slawomir Pilarski, Associate Professor
School of Computing Science, SFU
Chair

Continued next page

APPROVAL (CONTINUED)

Dr. Arvind Gupta, Assistant Professor
School of Computing Science, SFU
Co-Senior Supervisor

Dr. Tom Shermer, Associate Professor
School of Computing Science, SFU
Co-Senior Supervisor

Dr. Luis Goddyn, Assistant Professor
Department of Mathematics and Statistics, SFU
Supervisor

Dr. Pavol Hell, Professor
School of Computing Science, SFU
SFU Examiner

Dr. Detlef Seese, Professor
Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe
External Examiner

Date Approved: August 2, 1996

Abstract

The *treewidth* of a graph is a nonnegative integer that measures how closely the graph resembles a tree. For constant k, the class of graphs with treewidth at most k is also known as the class of *partial k-trees.* Dynamic-programming techniques can be used to solve many different problems in linear time over partial k-trees. For a decision problem of this sort, the corresponding linear-time algorithm can be modeled by a *tree automaton*—which is finite-state machine that *recognizes* the subclass of partial k-trees that are yes-instances of the decision problem. It is known that such a tree automaton exists to recognize any subclass that can be defined by a statement of the *Counting Monadic Second-order* (or CMS) logic: *i.e.* CMS-definability implies recognizability. It remains an open question whether, conversely, recognizability implies CMS-definability. This converse implication was previously known to hold only over partial 1-trees and partial 2-trees. In this thesis, we show it also holds over partial 3-trees and k-connected partial k-trees. Hence, a subclass of the partial 3-trees and k-connected partial k-trees can be recognized by a tree automaton if and only if it can be defined by a statement of CMS logic.

For many commonly-studied graph decision problems, the class (say Π) of yes-instances is CMS-definable. Thus, for any constant k, the intersection of Π with the class $\mathcal{G}_k$ of partial k-trees can be recognized by a tree automaton. We define the *complement-problem* of Π to be the class $\overline{\Pi}$ that contains the graph-theoretic complement of each graph in Π. For a CMS-definable class Π, it is often the case that $\overline{\Pi}$ is also CMS-definable; in other cases, $\overline{\Pi}$ is not CMS-definable, but $\overline{\Pi} \cap \mathcal{G}_k$ is CMS-definable. Either way, $\overline{\Pi} \cap \mathcal{G}_k$ is recognizable by a tree automaton: This not only provides a linear-time decision algorithm for $\overline{\Pi}$ over the class of partial k-trees, but it also provides a linear-time algorithm for Π over the class of partial k-tree complements. We will show, however, that $\overline{\Pi} \cap \mathcal{G}_k$ is not always CMS-definable when Π is CMS-definable. To obtain this result, we develop a *pumping lemma*—which can be

applied to an arbitrary subclass of the partial k-trees to (possibly) show that it cannot be recognized by a tree automaton, and hence, is not CMS-definable.

To the moon ...

  Art thou pale for weariness
Of climbing heaven and gazing on the earth,
  Wandering companionless
Among the stars that have a different birth,—
And ever changing, like a joyless eye
That finds no object worth its constancy?

  Thou chosen sister of the Spirit,
That gazes on thee till in thee it pities ...

"To The Moon"
by Percy Bysshe Shelley, 1820

Acknowledgements

I would like to thank my Senior Supervisors, Arvind Gupta and Tom Shermer, for collaborating on these results, for the many constructive critical comments, and for all of the support and encouragement provided over the last several years. I would like to thank Sanjeev Mahajan for introducing me to this area of research, for contributing to these results, and for the many helpful insights provided during the early phases of this work. I would also like to express my gratitude to Luis Goddyn, Pavol Hell and Detlef Seese for their helpful comments and suggestions, and to Dieter Kratsch for bringing to my attention several relevant results of which I had been previously unaware.

Contents

List of Figures

Chapter 1

Introduction

The *treewidth* of a graph is a nonnegative integer that measures how closely the graph resembles a tree. For constant k, the class of graphs with treewidth at most k is also known as the class of *partial k-trees.* These classes encompass many important graph families including trees and forests ($k = 1$), series-parallel and outerplanar graphs ($k = 2$), Halin graphs ($k = 3$) and k-terminal recursive graphs. A partial k-tree on n vertices has at most $kn - \frac{k^2+k}{2}$ edges; k is generally taken to be a constant. Dynamic programming techniques can be used to solve many different problems over these graphs with time complexity that is linear in the number of vertices (but is often super-exponential in k). In fact, most of the standard NP-complete problems [GJ79] can be solved in linear time over partial k-trees in this way. See Arnborg [Arn85] for a survey of early results in this area.

Takamizawa, Nishizeki and Saito [TNS82] described a general technique for constructing linear-time algorithms for combinatorial problems over series-parallel graphs (a subclass of the partial 2-trees). A number of more general paradigms were later developed [AP89, BLW87, Bod88, MP94], including several based on the *Monadic Second-order* (or MS) logic [ALS91, BPT92, Cou90b]: These provide linear-time algorithms over partial k-trees (for any constant k). Often, it is very easy to define a graph decision problem with a statement of MS logic; and any such statement can be automatically translated into an algorithm to solve the problem over the class of partial k-trees [ALS91, BPT92, Cou90b].

The MS logic is a predicate calculus over a universe comprised of the vertices and edges of a graph. Logical statements are written without using constants to represent elements of the universe; hence, a statement can be evaluated over an arbitrary graph. A logical variable may represent either an individual vertex or edge, or a set of them; predicates can be defined with any constant number of these variables as arguments. This is not, strictly speaking, a "second-order" predicate calculus—because variables may not be used to represent the predicates. However, a variable representing a set is effectively a predicate with exactly one argument—hence the name *Monadic Second-order* logic. Arnborg, Lagergren and Seese [ALS91] used this "ordinary" MS logic to develop a paradigm for solving many different graph decision problems in linear time over partial k-trees. Courcelle [Cou90b] and Borie, Parker and Tovie [BPT92] strengthened the MS logic by including predicates to determine the cardinality (modulo any fixed constant) of a set variable; this strengthened form of MS logic is called *Counting Monadic Second-order* (or CMS) logic. A CMS statement can be evaluated in linear time over any partial k-tree. The "counting" predicates do not come at any additional expense, and they do provide additional expressive power [Cou90b]. So we will not concern ourselves further with the "ordinary" MS logic.

Arnborg, Lagergren and Seese [ALS91] developed a much stronger version of MS logic, called *Extended* MS logic. Any statement written in this formalism can be evaluated in polynomial time over partial k-trees. Within the more general setting of weighted graphs, a statement can be evaluated in pseudopolynomial time over partial k-trees. This formalism strengthens MS logic by allowing logical statements to compare the (weighted) cardinalities of set variables; but this comes at the expense of sacrificing the linear time complexity for evaluating statements over partial k-trees. This thesis is concerned with using MS logic to obtain linear-time algorithms over partial k-trees, so we will not be using the Extended MS logical formalism.

CMS logic is well-suited to capture decision problems for which an instance consists of just a single graph. Any graph is either a *yes-instance* or a *no-instance* of such a problem; hence, any class of graphs is the class of yes-instances of some such problem. A CMS statement is always interpreted over a given graph; and it evaluates to either true or false. Hence, a CMS statement (say Φ) defines the decision problem for which a graph G is a yes-instance iff Φ is true when evaluated over G. If a decision problem can be defined in this way, then it can be solved over partial k-trees by evaluating the CMS statement. This evaluation can be performed in linear time by a finite-state machine called a *tree automaton* [GS84, Tho90]; the specifications of such a machine are inherent in the corresponding CMS statement [ALS91, BPT92, Cou90b]. Thus, a CMS statement can be translated automatically into a tree automaton that solves the corresponding decision problem in linear time over partial k-trees.

Any partial k-tree can be decomposed into a tree-like hierarchy of *basic* graphs—each with $k+1$ or fewer vertices. Arnborg, Corneil and Proskurowski [ACP87] showed how to construct such a *tree decomposition* in polynomial time; later, Bodlaender [Bod93] gave a linear-time algorithm to do this. A tree decomposition can be represented by a rooted tree (say T) for which each node has one of a constant number of labels—where the label of a node is determined by the structure of the corresponding basic graph. The tree T then forms the input to a tree automaton—which processes T in much the same way as a string would be processed by a conventional finite-state automaton: Each node of T becomes assigned to one of a constant number of states, computed as a function of the node's label and the states of its children; T 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 given tree decomposition.

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 [Cou90b] has shown that if a CMS statement defines a subclass (say Π) of the partial k-trees, then there exists a tree automaton that recognizes Π. Similar results were obtained independently by Arnborg, Lagergren and Seese [ALS91], and by Borie, Parker and Tovie [BPT92]. This fundamental result can be restated as "CMS-definability implies recognizability of partial k-trees". It remains an open question whether, conversely, recognizability implies CMS-definability of partial k-trees. This converse was previously known to hold only for partial 1-trees [Cou90b] and partial 2-trees [Cou91]. In this thesis, we prove the converse for partial 3-trees and k-connected partial k-trees (these results are also available in [Kal96]). We use general techniques which may lead to a proof of the conjectured equivalence—CMS-definability equals recognizability—over all partial k-trees.

While proving that recognizability implies CMS-definability of partial 1-trees, Courcelle [Cou90b] showed that this result would also hold for partial k-trees—if one could only encode (in CMS logic) 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. The class of partial 2-trees can be characterized by one forbidden *minor*: namely, the clique on four vertices. (A minor of a graph G is a graph that can be obtained from a subgraph of G by a series of edge contractions.) This characterization imposes a regular structure on partial 2-trees; and Courcelle [Cou91] used that structure to show that CMS logic can encode a fixed tree decomposition for any partial 2-tree. The work of Robertson and Seymour [RS] has established that partial k-trees (for any k) can be characterized with some finite number of forbidden minors. Unfortunately this number explodes with increasing k; and the forbidden minors are known only for $k \leq 3$ [APC90]. So it is difficult to use such a characterization in general to encode fixed tree decompositions of partial k-trees.

In Chapter 2 we define tree decompositions and tree automata more precisely, and explain how CMS predicates encoding a tree decomposition can be used to write a CMS statement that evaluates to true on a partial k-tree iff it belongs to the class that some tree automaton recognizes. We then show, in Chapter 3, that CMS predicates can indeed encode the structure of a fixed tree decomposition of any partial 3-tree (or k-connected partial k-tree). To do this, we begin with an arbitrary tree decomposition of a *2-connected* partial 3-tree G, and then modify it to satisfy several special properties. The modified tree decomposition can be used to represent G with a collection of *partial 3-paths* (such a graph is a restricted type of partial k-tree). We show that CMS logic can independently encode a fixed tree decomposition for each of these partial k-paths, and the corresponding collection of tree decompositions can then be assembled into a fixed tree decomposition of G. For an arbitrary partial 3-tree, each of its 2-connected *blocks* can be decomposed separately in this way; and the resulting collection of tree decompositions can then be assembled together. Thus we conclude that CMS-definability equals recognizability of partial 3-trees. The proof generalizes to also give this result for k-connected partial k-trees.

Under the conjecture that CMS-definability equals recognizability of partial k-trees, CMS logic would elegantly characterize which classes of partial k-trees can be recognized by tree automata—just as *regular expressions* [HU79] characterize the *regular* sets (of finite strings) that conventional finite-state automata can recognize. This characterization would provide a useful description of the decision problems that are amenable to the dynamic programming technique modeled by tree automata. Many commonly-studied decisions problem can be (almost automatically) expressed as CMS statements, thus providing linear-time decision algorithms over partial k-trees. Furthermore, such a decision problem is often associated with a corresponding search problem and optimization problem [GJ79]. Generally, with only minor modifications to the tree automaton (that solves the decision problem), one can obtain a linear-time algorithm for the corresponding search or optimization problem. Refer to Arnborg, Lagergren and Seese [ALS91], or Courcelle and Mosbah [CM93] for a further discussion of this.

Instead of focusing on decision problems within the confines of a logical formalism, Bern, Lawler and Wong [BLW87] studied problems that can be defined by a class of graph-subgraph pairs: For each pair (G, S) in such a class, S is a subgraph of G that satisfies certain problem-specific conditions. For example, the HAMILTONIAN CIRCUIT problem can be defined, in this way, by the class consisting of every pair (G, S) for which G is a graph, S is a subgraph of G, and S consists of a single cycle on all the vertices of G. Bern *et al.* describe a general technique to obtain linear-time algorithms for such problems over partial k-trees. Such an algorithm recognizes a particular class (say C) of graph-subgraph pairs in which G is a partial k-tree for each $(G, S) \in C$. Provided C satisfies a certain notion of *regularity*, the technique of Bern *et al.* can be applied to obtain a linear-time algorithm to decide whether an input pair (G, S) belongs to C. Such an algorithm can be modeled by a tree automaton whose input is formed from a tree decomposition (of G) that has been labeled to indicate the structure of the subgraph S. For many CMS-definable decision problems over partial k-trees, the corresponding search problem can be captured by a regular class of graph-subgraph pairs; hence, the technique of Bern *et al.* can be used to recognize (or to construct) any solution S for a partial k-tree G. There are, however, CMS-definable decision problems for which the corresponding search problem cannot be captured in this way: For example, it may be impossible to represent an arbitrary solution with a single subgraph (or even with any constant number of them). Also, it may be the case that a class C' of graph-subgraph pairs is not regular, despite the existence of a regular subclass C of C' such that $\{G \mid (G, S) \in C\} = \{G \mid (G, S) \in C'\}$. In this case, the yes-instances of the corresponding decision problem have a regular structure; "irregularity" is present only in some of the solutions: Although not every graph-subgraph pair in C' can be recognized by the technique of Bern *et al.*, the corresponding decision problem can be solved by recognizing the graph-subgraph pairs in C.

As mentioned above, a tree decomposition of a partial k-tree G is essentially a rooted tree T for which each node corresponds to a "basic subgraph" (with at most $k + 1$ vertices). For each node b of T, the subtree (say T_b) rooted at b corresponds to a collection of basic subgraphs—taken together, these basic graphs give a particular subgraph of G: This subgraph (say G^b) has k or fewer *terminals*—which are the only vertices of G^b that may belong to the basic subgraph corresponding to a node (of T) that is not in the subtree T_b. Hence, the tree T provides a way to divide a problem on G into subproblems on subgraphs of G. After solving the subproblems recursively, the solutions can be combined quite easily (provided the notion of regularity is satisfied) because the solutions of distinct subproblems can have at most k vertices (*i.e.* their terminals) in common. Thus T provides the structure for a "divide-and-conquer" (or dynamic-programming) algorithm to construct a solution of some problem on G. The MS logical formalism de-emphasizes the requirement that a solution on G need be obtained by combining solutions of subproblems on subgraphs of G. A tree automaton (obtained from an MS statement) assigns each node b of T to one of a constant number of states: The state of b reflects whether or not the corresponding subgraph G^b has a solution for any of the subproblems; so any irregular solutions can be safely ignored—provided there also exists one or more regular solutions.

Although many commonly-studied decision problems can be defined quite easily in CMS logic, there are others that cannot be readily expressed in this way. Some such problems can still be defined over partial k-trees by exploiting certain properties of those graphs (we give several examples of this in Chapter 4). Other problems provably cannot be encoded in CMS logic: A powerful tool for proving these negative results is given by Courcelle [Cou90b]: If Π is a CMS-definable class of graphs, then it is a decidable problem to ascertain whether Π contains a graph that can be generated using any specified "context-free" graph grammar. Hence, if an undecidable problem (such as POST'S CORRESPONDENCE PROBLEM) can be reduced to the question of whether a class Π contains a graph generated by some "context-free" graph grammar, then Π is not CMS-definable. This approach was used by Wanke and Wiegers [WW89] to show that BANDWIDTH (Problem GT40 in [GJ79]) is not CMS-definable: *i.e.* a CMS statement cannot define the class of graphs with bandwidth at most c (for any $c \geq 3$).

In this thesis, we develop a *pumping lemma* (analogous to the pumping lemma for regular sets [HU79]) to show that certain classes of graphs cannot be recognized by a tree automaton (and hence, cannot be defined by a CMS statement). The idea of applying a pumping lemma to graph problems is not new: Using the notion of regularity (as mentioned above [BLW87]), Mahajan and Peters [MP94] developed a pumping lemma to show that a certain "locality" condition must be satisfied by the graph-subgraph pairs constituting any regular class: Their pumping lemma says that if a certain graph-subgraph pair belongs to a given regular class, then certain other graph-subgraph pairs must also belong to it. In contrast, our pumping lemma says that if a certain graph belongs to a given CMS-definable class, then certain other graphs must also belong to it.

Any class (say Π) of graphs corresponds to the decision problem for which a graph is a yes-instance iff it belongs to Π. We use the same symbol (Π) to refer to either the graph class or the decision problem. We then define the *complement-problem* of Π to be the class (denoted by $\overline{\Pi}$) comprised of the (graph-theoretic) complement of each graph in Π. It is often the case that each of Π and $\overline{\Pi}$ is an interesting problem on its own. For example, if Π is the CLIQUE problem (*i.e.* the class of graphs containing a clique on, say, c vertices), then $\overline{\Pi}$ is the INDEPENDENT SET problem (*i.e.* the class of graphs containing an independent set of c vertices). For this example (and many others), both Π and $\overline{\Pi}$ can be defined by CMS statements, thus providing a linear-time algorithm for each of these problems over partial k-trees. From another perspective, a CMS statement defining Π (or $\overline{\Pi}$) provides an algorithm to solve $\overline{\Pi}$ (respectively Π) over the class of partial k-tree complements—with time complexity that is linear in the number of vertices. A partial k-tree complement (say G) on n vertices has $\Theta(n^2)$ edges, but it can be represented in $O(n)$ space with a list of its "non-edges": *i.e.* instead of storing the structure of G, we store the structure of its graph-theoretic complement $\overline{G}$ (which is a partial k-tree). Hence, if a tree automaton recognizes whether $\overline{G}$ belongs to Π, then this can be interpreted as the decision of whether G belongs to $\overline{\Pi}$.

In Chapter 4 we consider the question of whether CMS-definability is preserved under graph-theoretic complementation: *i.e.* if a decision problem Π is CMS-definable, then is its complement-problem $\overline{\Pi}$ CMS-definable? We are interested in defining $\overline{\Pi}$ in this way in order to obtain a linear-time algorithm over the class (denoted by $\mathcal{G}_k$) of partial k-trees. So for practical purposes, we need only define the intersection $\overline{\Pi} \cap \mathcal{G}_k$. We will see, however, that $\overline{\Pi} \cap \mathcal{G}_k$ is not always CMS-definable when Π is CMS-definable. In the case that Π is itself a subclass of $\mathcal{G}_k$, then $\overline{\Pi} \cap \mathcal{G}_k$ is trivially CMS-definable: Such a class would contain only a constant number of graphs—each being a partial k-tree as well as as the complement of some (other) partial k-tree. Hence, each graph in $\overline{\Pi} \cap \mathcal{G}_k$ would have $O(k)$ vertices; a CMS statement could then explicitly encode the structure of each such graph. We will give an example, in Chapter 4, of a problem Π that is CMS-definable (over all graphs); but, using the pumping lemma developed in Chapter 2, we will show that $\overline{\Pi} \cap \mathcal{G}_k$ is not CMS-definable. We will also examine several problems that are themselves CMS-definable for any graph, but whose complement-problems can only be defined over $\mathcal{G}_k$ by exploiting the structure of partial k-trees. We use two different approaches, in Chapter 4, to define the complement of a problem Π over the class $\mathcal{G}_k$: The first approach is to directly define the class $\overline{\Pi} \cap \mathcal{G}_k$ with a CMS statement. The second approach is to define $\Pi \cap \overline{\mathcal{G}_k}$ with a statement of the *Complement* CMS logic (where $\overline{\mathcal{G}_k}$ is the class of partial k-tree complements).

The Complement CMS (or CCMS) logic is virtually identical to the CMS logic, except that when a CCMS statement is evaluated over a graph G, the logical universe is comprised of the vertices and "non-edges" of G (*i.e.* the edges of the graph-theoretic complement $\overline{G}$ of G). If a problem Π is CCMS-definable over the class $\overline{\mathcal{G}_k}$ of partial k-tree complements, then a tree automaton can be used to solve Π in linear time over $\overline{\mathcal{G}_k}$, and (equivalently) to solve $\overline{\Pi}$ in linear time over $\mathcal{G}_k$. So CCMS logic captures decision problems with linear time complexity over $\overline{\mathcal{G}_k}$, in the same way that CMS logic captures decision problems with linear time complexity over $\mathcal{G}_k$. In fact, a CCMS statement being evaluated over a graph $G \in \overline{\mathcal{G}_k}$ is nothing but a disguised CMS statement being evaluated over $\overline{G} \in \mathcal{G}_k$.

The rest of this thesis is organized as follows: In Chapter 2 we review notation and terminology; we develop some preliminary results concerning tree decompositions, CMS logic and tree automata; we show that the conjecture "recognizability implies CMS-definability" can be proved by encoding in CMS logic a fixed tree decomposition of any partial k-tree; and we also present the pumping lemma in this chapter. Chapter 3 contains a proof that CMS logic can encode a fixed tree decomposition of any partial 3-tree or k-connected partial k-tree; thus we draw the conclusion that CMS-definability equals recognizability over those classes of graphs. The results of Chapter 3 can also be found in [Kal96]. In Chapter 4 we discuss the idea of defining complement-problems with CMS logic, and we present the Complement CMS logic. We consider several different problems that are CMS-definable over all graphs; we show that their complement-problems are, in some cases, CMS-definable over partial k-trees, and in other cases, not CMS-definable over partial k-trees. Most of the results of Chapter 4 are contained in [KGS95a, KGS95b, GKMS96]. Finally, in Chapter 5 we make concluding remarks and discuss open problems.

Chapter 2

Preliminaries

In this chapter we review the background material needed to develop the results of this thesis. We also present a few preliminary lemmas, so that their proofs need not disturb the flow of later chapters. In Sections 2.1 and 2.2 we describe our notation and give a few elementary definitions; in Sections 2.3 to 2.5 we briefly review some graph-theoretic results which will be needed later on; refer to Berge [Ber76] or Bollobas [Bol78] for a more complete treatment of graph theory. In Section 2.6 we discuss *tree decompositions*—which provide a useful representation of the graphs called *partial k-trees* (for $k \in \mathbb{N}$). In Section 2.7 we describe the *Counting Monadic Second-order* logic, and explain (with examples) how graph decision problems can be defined by logical statements in this language. Such a statement can be automatically translated into a linear-time dynamic-programming algorithm to solve the problem over the class of partial k-trees; and in Section 2.8 we discuss *tree automata* which implement these algorithms by deciding whether or not to accept any given tree decomposition of the input graph. Finally, in Section 2.9 we present a "pumping lemma" that can be used to show certain problems cannot be solved over partial k-trees by a tree automaton.

2.1 Graph-theoretic Notation

This thesis is concerned only with graphs that are finite and simple; we will assume implicitly that any graph satisfies these properties. Furthermore, unless specifically stated otherwise, we assume that any graph is undirected. We will use directed graphs only as a tool for developing certain results over undirected graphs.

A graph is a pair (V, E) where V is a finite set of vertices, and E is a set of edges—each edge being a subset of V with cardinality two. The vertex set of a graph G may be denoted by $V(G)$, and its edge set may be denoted by $E(G)$. The *endpoints* of an edge e are the vertices belonging to the set $e = \{u, v\}$: We say that e is *incident* to u (and to v). Two vertices of G are said to be *adjacent* if they are the endpoints of a common edge. Similarly, if H and H' are vertex-disjoint subgraphs of G, and some edge of G has one endpoint in $V(H)$ and the other endpoint in $V(H')$, then we say that H and H' are adjacent. The *degree* of a vertex $v \in V(G)$ is the number (denoted by $\delta_G(v)$) of edges in $E(G)$ that have v as an endpoint. The *complement graph* of G (denoted by $\overline{G}$) is the graph with vertex set $V(\overline{G}) = V(G)$ such that two distinct vertices are adjacent in $\overline{G}$ iff they are not adjacent in G.

Suppose G and G' are graphs. We write $G' \cong G$ to indicate that they are isomorphic. We write $G' \sqsubseteq G$ to indicate that G' is a subgraph of G. If G' is a subgraph of G such that $V(G) = V(G')$, then G' is called a *factor* of G. The *union* of G and G' (denoted by $G \sqcup G'$) is the graph with vertex set $V(G) \cup V(G')$ and edge set $E(G) \cup E(G')$; it is not necessarily the case here that G and G' are disjoint graphs.

The subgraph of G *induced* by $V' \subseteq V(G)$ is denoted by $G_{[V']}$: this is the graph with vertex set V' such that two vertices are adjacent in $G_{[V']}$ iff they are adjacent in G (and both belong to V'). If V' is a maximal subset of $V(G)$ such that $G_{[V']}$ is connected, then $G_{[V']}$ is called a *component* of G.

We use the symbol "\" to represent deletion of vertices or edges from a graph: If G' is a subgraph of G, and V' is a subset of $V(G)$, then $G' \backslash V'$ is the subgraph of G' induced by $V(G') - V'$; it is not necessarily the case here that V' is a subset of $V(G')$. 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 $G \backslash V(G')$. If E' is a subset of $E(G)$, then $G \backslash E'$ is the factor of G with edge set $E(G) - E'$. We will *not* write $G \backslash e$ in place of $G \backslash \{e\}$, thus avoiding any possible confusion with the subgraph induced by $V(G) - e$.

If G' is an induced subgraph of G, and V' is a subset of $V(G)$, then $G' + V'$ is the subgraph of G induced by $V(G') \cup V'$.

2.2 Trees and Paths

A *tree* is an acyclic connected graph; and a *path* is a tree in which no vertex is incident to more than two edges. If a vertex of a path P is incident to fewer that two edges, then it is an *endpoint* of P; otherwise, it is an *internal* vertex of P. If P is a subgraph of a graph G, such that P is a path with endpoints u and v (where $u = v$ iff P has exactly one vertex), then we say that P is a path in G *between* u and v.

Suppose P and P' are paths in a graph G: 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.

A *rooted* tree is a tree T with a specially-designated vertex, called the *root*: Each non-root vertex v of T then has a unique *parent*—which is the vertex adjacent to v on the unique path in T between v and the root. The terms "child", "ancestor", "descendant" *etc.* are defined analogously. 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. For a vertex v of a rooted tree T, the *subtree* of T rooted at v (denoted by T_v) is the connected subgraph of T whose vertex set is comprised of v and all descendants of v. We will use the term "subtree" only to mean this special type of subgraph of a rooted tree.

Definition 2.2.1. 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.

2.3 Cut-sets and Separators

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 $G \backslash V'$.

We will adopt the following definition for the connectivity of a graph.

Definition 2.3.1. An *ℓ-connected* graph (for $\ell \in \mathbb{Z}^+$) is a graph for which there are ℓ internally-vertex-disjoint paths between each pair of non-adjacent 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 [Bol78]. Lemma 2.3.2 (first given by Menger [Men27]) shows that these two characterizations are equivalent for graphs with more than ℓ vertices.

Lemma 2.3.2. *Let $\ell \in \mathbb{Z}^+$, and suppose G is a 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 2.3.1, a graph on ℓ or fewer vertices is ℓ-connected iff it is a clique; by the alternative definition, no such graph would be ℓ-connected. Note that Definition 2.3.1 implies that a clique has infinite connectivity.

2.4 Partial k-Trees

A *k-tree* is either a 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 (*i.e.* a collection of trees). Series-parallel graphs and outerplanar graphs are subclasses of the partial 2-trees; Halin graphs [Hal71] form a subclass of the partial 3-trees.

There are several other equivalent ways to characterize the class of partial k-trees (see *e.g.* Arnborg [Arn85]). For example, partial k-trees are subgraphs of those chordal graphs for which no minimal cut-set has cardinality exceeding k. The class of partial k-trees can also be characterized with a finite *obstruction* set (as explained in Section 2.5). Furthermore, a graph is a partial k-tree iff it admits a *width-k tree decomposition* (to be defined in Section 2.6).

2.5 Graph Minors

A *minor* of a graph G is a graph that can be obtained from a subgraph of G by a sequence of zero or more *edge contractions*:

Definition 2.5.1. Suppose u and v are adjacent vertices of a graph H. Let H' be the graph obtained from $H \backslash \{u, v\}$ by adding a new vertex (say w) with an edge between w and each vertex $x \in V(H) - \{u, v\}$ for which $\{x, u\} \in E(H)$ or $\{x, v\} \in E(H)$. We say that H' is obtained by *contracting* $\{u, v\}$ into w.

A class $\mathcal{C}$ of graphs is said to be *minor-closed* if $G \in \mathcal{C}$ implies that every minor of G is also in $\mathcal{C}$. Robertson and Seymour [RS] have shown that any minor-closed class of graphs has a finite set of forbidden minors; and this set—called the *obstruction* set—can be used to characterize the class of graphs: Thus G belongs to a minor-closed class of graphs iff no minor of G belongs to the corresponding obstruction set. Although the obstruction set is always finite, it is often quite large, and its graphs may be very difficult to identify. The following lemma is not hard to prove (see *e.g.* [APC90]):

Lemma 2.5.2. *For $k \in \mathbb{N}$, the class of partial k-trees is minor-closed.*

The obstruction sets for partial 0-trees, partial 1-trees and partial 2-trees each consist of a single graph—the clique on two, three or four vertices (respectively). The obstruction set for partial 3-trees consists of four graphs, the largest of which has ten vertices [APC90]. For $k \geq 4$, the obstruction set for partial k-trees is not known.

In Section 2.7 we will describe how a graph class can be defined by a statement of the Monadic Second-order (or MS) logic. If such a graph class (say $\mathcal{C}$) is minor-closed, then the MS statement need only encode the requirement that no minor of a graph in $\mathcal{C}$ belong to the (finite) obstruction set. Thus, from the obstruction set of a minor-closed graph class, one can easily derive a MS statement defining it. Arnborg *et al.* [APS90] have also established the converse of this implication: *i.e.* from an MS definition of a minor-closed graph class, one can determine the graphs in its obstruction set.

2.6 Tree Decompositions

Definition 2.6.1. 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:

T1: $\bigcup_{a \in V(T)} X_a = V(G)$.

T2: Each edge of G has both endpoints in some set $X_a \in \mathcal{X}$.

T3: 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 the elements of $V(T)$ as *nodes*, so as not to confuse them with the 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 $(T, \mathcal{X})$ 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 nodes of T'; and $X_{T'}$ is the the union of those bags:

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

We refer to the subgraph of G that is induced by $X_{T'}$ as the subgraph *underlying* T'.

Fact 2.6.2. *If $(T, \mathcal{X})$ is a width-k tree decomposition of a graph G, then $V(G)$ can be partitioned into sets $V_1, V_2, \ldots, V_{k+1}$ such that $|V_i \cap X| \leq 1$ for each $X \in \mathcal{X}$, $1 \leq i \leq k+1$.*

Proof. Since no bag of a width-k tree decomposition contains more than $k+1$ vertices, this fact follows easily from property **T3** (Def. 2.6.1). □

It is clear that any graph on n vertices admits a width-$(n-1)$ tree decomposition. The *treewidth* of a graph G is the minimum k such that G admits a width-k tree decomposition. It is not hard to show (see *e.g.* [vL90]) that G admits a width-k tree decomposition iff G is a subgraph of a k-tree (as defined in Section 2.4). Hence, the class of partial k-trees is the class of graphs with treewidth bounded by k. It will be useful to designate k or fewer vertices of such a graph as *terminals*:

Definition 2.6.3. 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 2.6.4. 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; $(T, \mathcal{X})$ is a tree decomposition of G; and $V_{\text{term}}(G) \subseteq X_r$.

If $(T, r, \mathcal{X})$ 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 2.6.5. Suppose $(T, r, \mathcal{X})$ 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 *non-drop* 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 2.6.6. 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 "non-drop" vertex. Two basic graphs B and B' are *equivalent* if $B \cong B'$ where an isomorphism respects the labels and designations of each vertex. The collection of these equivalence classes is called the *k-derivation alphabet.*

In Section 2.8 we will explain how graph problems can be solved over partial k-trees using finite-state machines called *tree automata.* The input to such a machine is a rooted tree for which each node is labeled with a symbol from the k-derivation alphabet.

Proposition 2.6.7. *Let Σ_k be the k-derivation alphabet; and suppose $(T, r, \mathcal{X})$ is a width-k rooted tree decomposition of a graph G. There exists a function $\sigma : V(T) \to \Sigma_k$, 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 "drop" vertex iff it is a drop vertex of b.*

We say that σ is a derivation function *for G on T.*

Proof. Let each vertex of G be labeled with an integer between 1 and $k+1$. Without loss of generality (by Fact 2.6.2), assume the vertices of X_b are labeled distinctly, for each $b \in V(T)$. The proposition follows easily. □

In Chapter 3 we will modify a given rooted tree decomposition by *contracting* edges and *splitting* nodes. We have already defined the operation that contracts an edge of a graph (Def. 2.5.1): We apply this operation to a tree decomposition as follows:

Definition 2.6.8. Suppose $(T, r, \mathcal{X})$ 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 by contracting $\{b, c\}$ into a new node (say b');
- let $r' = \begin{cases} b' & \text{if } r \in \{b, c\} \\ r & \text{otherwise;} \end{cases}$
- 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 $(T, r, \mathcal{X})$ by *contracting* $\{b, c\}$.

We will apply this operation to a width-k tree decomposition only if $|X_b \cup X_c| \leq k + 1$, so the width of the resulting tree decomposition shall also be k.

Definition 2.6.9. Suppose $(T, r, \mathcal{X})$ is a rooted tree decomposition; and let $b \in V(T)$.

- Let T' be the tree obtained from $T \backslash b$ by adding two new nodes (say b' and b'') and the edge $\{b', b''\}$ and also one edge (either $\{a, b'\}$ or $\{a, b''\}$) whenever $\{a, b\} \in E(T)$;
- let $r' = \begin{cases} b' & \text{if } r = b \\ r & \text{otherwise;} \end{cases}$
- let $\mathcal{X}' = \mathcal{X} - \{X_b\} \cup \{X_{b'}, X_{b''}\}$, where $X_{b'}, X_{b''} \subseteq X_b$.

We say that $(T', r', \mathcal{X}')$ is obtained from $(T, r, \mathcal{X})$ by *splitting* b.

To use this operation, we must specify which of b' or b'' becomes the parent of each child of b, and which becomes the child of the parent of b; and we must choose the bags corresponding to b' and b'' so as not to violate Definition 2.6.1. In particular, the non-drop vertices of b cannot become drop vertices of either b' or b''.

2.7 Counting Monadic Second-Order Logic

A graph $G = (V, E)$ can be interpreted as a logical structure over the universe $V \cup E$. The structure of G is described by a predicate $Edge(e, v)$ which holds whenever $v \in V$ is an endpoint of $e \in E$. Many different properties of graphs can then be expressed in any of several variations of the *Monadic Second-order* (or MS) logic [ALS91, BPT92, Cou90b]. We follow Courcelle's *Counting* MS (or CMS) logic [Cou90b] which uses 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") $\Leftrightarrow$ ("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 ℓ (mod c). The inclusion of the $\mathbf{card}_{\ell,c}$ predicates is what distinguishes CMS logic from the "ordinary" MS logic. Courcelle [Cou90b] has shown that these predicates do, in fact, give the logic additional expressive power.

By a CMS "formula", we mean a string of the symbols listed above, constructed such that the usual syntactic rules of logic are observed; quantification is allowed over both individual and set variables. Such a formula is always interpreted on a given graph—called the *evaluation graph.* If every variable is quantified in a CMS formula, then it is called a *CMS statement*—which is either true or false on the evaluation graph. We write $G \models \Phi$ to indicate that a CMS statement Φ is true on a graph G. This statement defines a certain class (say Π) of graphs: that is, $G \models \Phi$ iff $G \in \Pi$. 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 statements—we give two examples below, and further examples in Chapter 4. Any CMS statement can be evaluated in linear time on a partial k-tree by a *tree automaton*—as discussed in Section 2.8. Thus, there is a linear-time algorithm to solve any CMS-definable decision problem over the class of partial k-trees.

If Φ is a CMS formula, then some number (say ℓ) of its unquantified variables may be designated as *arguments*: This defines a *CMS predicate*—which we denote by the symbol "Φ" followed by an ordered sequence 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. The relation $\mathcal{L}$ is a set containing sequences of ℓ elements—each of which is either a vertex, edge, vertex set or edge set (of the evaluation graph). We now generalize this notion by allowing Φ to be defined with unquantified variables that are not arguments: These are called *free* variables.

Definition 2.7.1. Suppose Φ is a CMS predicate defined with ℓ arguments (and some number of free variables). Let each free variable assume some fixed value; then 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 a relation $\mathcal{L}$ is existentially encoded by a CMS predicate Φ, then the free variables (say $x_1, x_2, \ldots$ and x_d) of Φ can be existentially quantified at the outermost level of a CMS statement as follows:

$$(\exists x_1, x_2, \ldots, x_d)(\Phi') \tag{2.7.2}$$

Here, Φ' is a CMS formula that does not contain any unquantified variable other than $x_1, x_2, \ldots$ and x_d; so the predicate Φ can be used to writing the CMS statement (2.7.2). It is clear that two (or more) existentially-defined predicates may be used together, in statements of this form, by quantifying all of their free variables at the outermost level. In Chapter 3 we will use this approach to encode the structure of a rooted tree decomposition of the evaluation graph G. To do this, a subset of $V(G)$ will be used as *witnesses* representing the bags (or nodes) of the tree decomposition.

Definition 2.7.3. Suppose **Bag** and **Parent** are binary CMS predicates. These predicates are said to *describe* a rooted tree decomposition $(T, r, \mathcal{X})$ of a graph G if there exists a one-to-one function $f : \mathcal{X} \to V(G)$, such that

- $\mathbf{Bag}(v, X)$ holds iff $v = f(X)$, and
- $\mathbf{Parent}(p, c)$ 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).

In Chapter 3 we will develop existentially-defined CMS predicates (**Bag** and **Parent**) that describe a rooted tree decomposition $(T, r, \mathcal{X})$ of the evaluation graph G—provided it is a 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 subset of $V(G)$ comprised of the witnesses:

$$V(T) = \{v \in V(G) \mid (\exists X)\mathbf{Bag}(v, X)\} \tag{2.7.4}$$

In Section 2.8 we will show how the **Bag** and **Parent** predicates can be used to write a CMS statement (2.7.2), where Φ' encodes whether or not the described tree decomposition is accepted by a particular tree automaton. The validity of this CMS statement is, of course, dependent upon "correct" values having been chosen for the free variables. At the end of this section, we will show how Φ' can be written to ensure that the "correct" values are indeed chosen.

We now give a few examples showing how CMS predicates can be defined, beginning with a predicate (denoted by the infix operator "$\sim$") to encode the (symmetric and irreflexive) edge relation:

$$u \sim v \equiv \neg(u = v) \wedge (\exists e)(Edge(e,u) \wedge Edge(e,v)) \tag{2.7.5}$$

We write "$\sim^+$" for the predicate encoding the transitive closure of the edge relation. So if u and v are distinct vertices of a graph G, then $u \sim^+ v$ holds iff G contains a path between u and v. The following lemma (see also [Cou90b, Lemma 3.7]) shows that "$\sim^+$" is a CMS predicate.

Lemma 2.7.6. *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. Suppose Φ is a CMS predicate encoding a binary relation on the vertex set V. The transitive closure of this relation is encoded by a CMS predicate Φ^+ such that, for $u, v \in V$, $\Phi^+(u,v)$ holds iff v belongs to every set (say V') that contains u and is closed under Φ:

$$\begin{aligned}\Phi^+(u,v) \quad \equiv \quad & (u \in V) \wedge (v \in V) \wedge \\ & (\forall V')(\ (\quad (u \in V') \wedge (\forall x,y)(\ ((x \in V') \wedge \Phi(x,y)) \Rightarrow (y \in V')\ ) \\ & \qquad\quad)\ \Rightarrow (v \in V')\ )\end{aligned}$$

□

Our next two examples show how graph problems can be defined by CMS statements. Our first problem—whether a graph $G = (V,E)$ is connected—can be encoded using the transitive closure of the edge relation (2.7.5). Alternatively, it can be encoded with a statement requiring there be an edge between any pair of nonempty vertex subsets (say X and Y) that cover the vertex set V.

Example 2.7.7. The following statements are equivalent:

- $G = (V,E)$ is a connected graph.
- $(\forall u, v \in V)((u = v) \vee (u \sim^+ v))$.
- $(\forall X,Y)(\quad (\ (\forall v)((v \in V) \Leftrightarrow ((v \in X) \vee (v \in Y))) \wedge (\exists x)(x \in X) \wedge (\exists y)(y \in Y)\ )$
  $\Rightarrow (\exists x,y)((x \in X) \wedge (y \in Y) \wedge (x \sim y))\ \ )$

In our next example, we use a CMS predicate for connectedness to encode the existence of a Hamiltonian circuit: *i.e.* a connected factor of the input graph in which exactly two edges are incident to each vertex. We also use the symbol "$\subseteq$", which can be translated into CMS logic as follows: $X \subseteq Y \equiv (\forall x)(x \in X \Rightarrow x \in Y)$.

Example 2.7.8. A graph $G = (V, E)$ has a Hamiltonian circuit iff the following CMS statement is satisfied:

$$\begin{aligned}(\exists E')(\quad & (E' \subseteq E) \wedge \text{“}(V, E') \text{ is connected”} \wedge \\ & (\forall v \in V)(\exists e, e' \in E')(\quad \neg(e = e') \wedge Edge(e, v) \wedge Edge(e', v) \wedge \\ & \qquad\qquad\qquad\qquad \neg(\exists e'' \in E')(\neg(e'' = e) \wedge \neg(e'' = e') \wedge Edge(e'', v)) \\ ) & \qquad\qquad\qquad)\end{aligned}$$

When writing CMS formulae in this thesis, we will often use high-level expressions (such as "G is connected") rather than providing a detailed translation into the low-level logical symbols of CMS. We will use such high-level expressions if we have already shown how to encode them; and we will also use them (without formality) if they are trivial to encode. Refer to Courcelle [Cou90a, Cou90b] or Borie *et al.* [BPT92] for a further discussion of encoding such expressions.

We will need the following lemmas in Chapter 3:

Lemma 2.7.9. *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' with a binary CMS predicate Φ that is defined with $k + 2$ free set variables $V_1, V_2, \ldots, V_{k+1}$ and E''.

$$\begin{aligned}\Phi(u, v) \quad \equiv \quad (\exists e \in E')(\quad & Edge(e, u) \wedge Edge(e, v) \wedge \neg(u = v) \wedge \\ & ((e \in E'') \Leftrightarrow \textstyle\bigvee_{i=1}^{k} \bigvee_{j=i+1}^{k+1} (u \in V_i \wedge v \in V_j))\ )\end{aligned}$$

Suppose $V_1, V_2, \ldots, V_{k+1}$ are independent sets that partition $V(G)$. It follows from Fact 2.6.2 and property **T2** (Def. 2.6.1) that such a partition exists. Thus, for any edge $e \in E'$, its endpoints (say u and v) belong to distinct sets: Without loss of generality, assume $u \in V_i$ 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 (if e does not belong to E''), $\Phi(u, v)$ is false and $\Phi(v, u)$ is true. Therefore, Φ encodes a unique direction for each edge in E': 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 existentially encoded, by choosing a vertex partition $(V_1, V_2, \ldots, V_{k+1})$ and an appropriate subset E'' of E'. □

Lemma 2.7.10. *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 free set variables $X_1, X_2, \ldots, X_\ell$ to represent a string of ℓ bits for each vertex or edge (say x): The i^{th} bit $(1 \leq i \leq \ell)$ is turned on iff $x \in X_i$. □

The preceding lemmas will be used in Chapter 3 to obtain existentially-defined CMS predicates that describe (Def. 2.7.3) a rooted tree decomposition. Using the following lemma, we can then ensure that the free variables of those predicates are always instantiated "correctly".

Lemma 2.7.11. *Suppose* **Bag** *and* **Parent** *are existentially-defined CMS predicates with free variables $x_1, x_2, \ldots, x_d$ (in some fixed order). There exists a CMS predicate Φ with d arguments and zero free variables 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 free variable x_i assumes the value c_i $(1 \leq i \leq d)$.*

Proof. Let $G = (V, E)$ be the evaluation graph. Within the scope of the arguments $x_1, x_2, \ldots, x_d$, the predicate Φ can identify a subset $V(T)$ of $V(G)$ comprised of the witnesses, as shown in equation (2.7.4). Now, Φ can verify the following:

- $\mathbf{Bag}(v, X) \wedge \mathbf{Bag}(v, X') \Rightarrow X = X'$; and
- $\mathbf{Bag}(v, X) \wedge \mathbf{Bag}(v', X) \Rightarrow v = v'$; and
- there is exactly one witness $r \in V(T)$ such that $(\forall p \in V(T))\neg\mathbf{Parent}(r, p)$; and
- for $v \in V(T) - \{r\}$, there is a unique witness $p \in V(T)$ such that $\mathbf{Parent}(v, p)$.

By Lemma 2.7.6, a CMS predicate (say **Ancestor**) can encode the transitive closure of **Parent**. Then Φ can test whether the **Ancestor** relation is irreflexive, antisymmetric and transitive. If so, the **Parent** predicate encodes the edges of a tree, on $V(T) \subseteq V(G)$, that is rooted at r. Now, Φ can verify that a rooted tree decomposition $(T, r, \mathcal{X})$ is described by

Bag and **Parent**, as follows:

- $(\forall v \in V)(\exists x, X)(\mathbf{Bag}(x, X) \wedge (v \in X))$
- $(\forall e \in E)(\exists X, x, v, u)(\mathbf{Bag}(x, X) \wedge Edge(e, v) \wedge Edge(e, u) \wedge (u, v \in X) \wedge \neg(u = v))$
- $(\forall X, Y, Z, x, y, z, v)\left(\begin{pmatrix} \mathbf{Bag}(x, X) \wedge \mathbf{Bag}(y, Y) \wedge \mathbf{Bag}(z, Z) \wedge \\ \mathbf{Ancestor}(x, y) \wedge \mathbf{Ancestor}(y, z) \wedge \\ (v \in X) \wedge (v \in Z) \end{pmatrix} \Rightarrow (v \in Y) \right)$
- $(\forall X, Y, Z, x, y, z, v)\left(\begin{pmatrix} \mathbf{Bag}(x, X) \wedge \mathbf{Bag}(y, Y) \wedge \mathbf{Bag}(z, Z) \wedge \\ \mathbf{Ancestor}(x, y) \wedge \mathbf{Ancestor}(x, z) \wedge \\ (v \in Y) \wedge (v \in Z) \end{pmatrix} \Rightarrow (v \in X) \right)$

The first two logical statements encode properties **T1** and **T2** (Def. 2.6.1), respectively. Property **T3** is encoded by the conjunction of the last two logical statements. Now, to verify that a *width-k* tree decomposition is described, Φ need only check that $\mathbf{Bag}(v, X) \Rightarrow$ "$|X| \leq k + 1$". □

2.8 Tree Automata

We assume the reader is familiar with conventional finite-state automata [HU79, Per90]. Such a machine executes finite strings over some alphabet. An input string can be interpreted as a path, rooted at one of its endpoints, for which each node is labeled with a symbol from the alphabet. The automaton then assigns each node (in order from the leaf to the root) to one of a constant number of states: The state is computed as a binary function of the node's label and the state of its child. The string (path) is accepted iff the root is thus assigned to a designated accepting state. A *tree automaton* is defined by generalizing this conventional finite-state automaton to execute trees, instead of just paths [GS84, Tho90].

Definition 2.8.1. A *tree automaton* over an alphabet Σ is a quadruple $\mathcal{A} = (\mathcal{S}, S_0, \mathcal{S}_A, f)$ where $\mathcal{S}$ is a finite set of *states*; $S_0 \in \mathcal{S}$ is the *initial* state; $\mathcal{S}_A \subseteq \mathcal{S}$ is the set of *accepting* states; and $f : \mathcal{S} \times \mathcal{S} \times \Sigma \rightarrow \mathcal{S}$ is the *transition function.*

The *input* to $\mathcal{A}$ is a rooted binary tree T, with a labeling function $\sigma : V(T) \rightarrow \Sigma$. Each leaf b of T is then assigned to the state $f(S_0, S_0, \sigma(b))$; and each other node b of T is assigned to the state $f(S, S', \sigma(b))$, where $S, S' \in \mathcal{S}$ are the states to which the children of

b are (recursively) assigned. The tree T is *accepted* by $\mathcal{A}$ iff its root is thus assigned to an accepting state.

For our purposes, a tree automaton is used to decide whether a partial k-tree G belongs to a certain class of graphs. The input is the tree T of a width-k rooted tree decomposition of G. The alphabet Σ is the k-derivation alphabet (Def. 2.6.6); and the labeling function σ is nothing other than a derivation function for G on T (as defined by Proposition 2.6.7). We assume that any such input tree T is binary: This is no loss of generality, because any width-k tree decomposition can be quite easily modified into one with a binary tree—a construction is given in Definition 2.8.4. If we say that a tree automaton accepts a rooted tree decomposition $(T, r, \mathcal{X})$, we mean that it accepts the tree T with its vertices labeled by an appropriate derivation function.

Suppose $(T, r, \mathcal{X})$ is a width-k rooted tree decomposition of a graph G: If T forms the input to a tree automaton $\mathcal{A}$ (with, say, s states), then the nodes of T are effectively partitioned into s sets. Thus, for each $b \in V(T)$, the subgraph (say G^b) underlying T_b belongs to one of s equivalence classes—each corresponding to some state of $\mathcal{A}$. If T_b would be accepted by $\mathcal{A}$, then G^b belongs to an equivalence class corresponding to an accepting state. So the tree automaton computes the equivalence class of G^b as a function of the equivalence classes of the subgraphs underlying T_c and $T_{c'}$ (where c and c' are the two children of b). This provides a model for a "divide-and-conquer" (or dynamic-programming) algorithm to solve a certain decision problem over partial k-trees. We are only interested in those tree automata that accept either all of the (binary) tree decompositions of a given partial k-tree, or none of them. Any such tree automaton, then, *recognizes* a subclass of the partial k-trees.

Definition 2.8.2. Let Π be a subclass of the partial k-trees; and suppose $\mathcal{A}$ is a tree automaton over the k-derivation alphabet Σ_k. We say that $\mathcal{A}$ *recognizes* Π if the following statements are equivalent for any rooted binary tree T:

- $\sigma : V(T) \to \Sigma_k$ is a derivation function for some graph in Π.
- $\mathcal{A}$ accepts the tree T labeled with σ.

Courcelle [Cou90b] has shown that any CMS statement Φ can be automatically translated into a tree automaton that recognizes the subclass of partial k-tree over which Φ

evaluates to true. Similar results were obtained independently by Arnborg *et al.* [ALS91], and by Borie *et al.* [BPT92]. These results provide the following:

Theorem 2.8.3. *If Φ is a CMS statement, 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.*

This theorem can be restated more tersely as follows: CMS-definability implies recognizability over partial k-trees. So a tree automaton exists to solve any CMS-definable decision problem over the class of partial k-trees. Every problem that is known to be amenable to such an algorithm is also known to be CMS-definable; so it has been conjectured [Cou91] that recognizability implies CMS-definability over partial k-trees. This conjectured converse of Theorem 2.8.3 is known to hold over partial 1-trees [Cou90b] and partial 2-trees [Cou91]; in Chapter 3, we extend the result to partial 3-trees and k-connected partial k-trees. To obtain this result, we will develop CMS predicates that describe (Def. 2.7.3) a rooted tree decomposition of any such graph. We can then write a CMS formula to encode whether or not a tree automaton accepts a *binary representation* of that tree decomposition.

Definition 2.8.4. Suppose $(T, r, \mathcal{X})$ is a rooted tree decomposition; and let T' be obtained from T as follows: If $b \in V(T)$ has exactly one child, then add a new node (say c) and the edge $\{b, c\}$. If $b \in V(T)$ has $n \geq 3$ children (say $c_1, c_2, \ldots$ and c_n), then

- add $n-2$ new nodes (say $p_1, p_2, \ldots$ and p_{n-2}), and
- delete the edges between b and $n-1$ of its children (say $c_1, c_2, \ldots$ and c_{n-1}), and
- add the following $2n-3$ edges: $\{b, p_1\}$, $\{p_i, p_{i+1}\}_{i=1}^{n-3}$, $\{p_i, c_i\}_{i=1}^{n-2}$ and $\{p_{n-2}, c_{n-1}\}$.

We say that $(T', r, \mathcal{X}')$ is a *binary representation* of $(T, r, \mathcal{X})$, where $\mathcal{X}'$ contains each bag $X_b \in \mathcal{X}$ (for $b \in V(T)$), as well as a bag X_p (for each node p of $T' \backslash T$) that is identical to the bag indexed by the parent of p.

Figure 2.1 illustrates how a node b of T with $n \geq 3$ children is converted into a path (say P_b) in T' that consists of $n-1$ nodes: We say that this path P_b *represents* the node b. All the new nodes of $P_b \backslash b$ are given the same label by any derivation function $\sigma : V(T') \to \Sigma_k$. This label is different from $\sigma(b)$ if b has one or more drop vertices (because the new nodes do not have any drop vertices). We now describe how a CMS formula can represent the state of each node in $V(T) \subseteq V(T')$, when T' is input to a tree automaton.

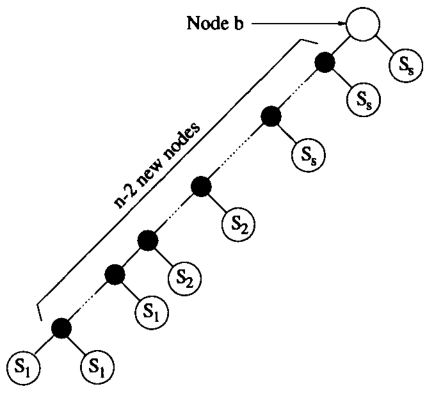

Figure 2.1: A binary representation of a tree decomposition

Lemma 2.8.5. *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 partial* k*-tree. It follows that a CMS statement* Φ *can be written such that* $G \models \Phi$ *iff* $G \in \Pi$.

Proof. Suppose $\mathcal{A}$ is a tree automaton over Σ_k that recognizes a subclass Π of the partial k-trees. Suppose further that **Bag** and **Parent** are CMS predicates that describe a width-k rooted tree decomposition of the evaluation graph—provided it is a partial k-tree. Let $(T, r, \mathcal{X})$ be the tree decomposition described by these predicates for some partial k-tree G; and let $(T', r, \mathcal{X}')$ be a binary representation of $(T, r, \mathcal{X})$. Hence, $\mathcal{A}$ accepts $(T', r, \mathcal{X}')$ iff $G \in \Pi$. We now describe how a CMS statement Φ can be written such that $G \models \Phi$ iff $(T', r, \mathcal{X}')$ is accepted.

A set $V(T) \subseteq V(G)$ can be identified within the CMS statement Φ, as shown in equation (2.7.4). Then Φ can say that $V(T)$ is partitioned into sets $S_1, S_2, \ldots, S_s$, each representing one of the (say s) states of $\mathcal{A}$. To encode whether $(T', r, \mathcal{X}')$ is accepted, then, Φ need only verify that this state assignment is consistent with the transition function of $\mathcal{A}$; and then test whether the witness of the root is thus assigned to a designated accepting state.

To complete the proof, we need only show how Φ can verify that the state of $b \in V(T)$ is consistent with the states of $C \subseteq V(T)$, where $(c \in C) \Leftrightarrow \textbf{Parent}(b, c)$. Let P_b be the path in T' that represents b. Without loss of generality, assume that T' was constructed such that the nodes corresponding to each set $C \cap S_i$ $(1 \leq i \leq s)$ are grouped together: *i.e.* their parents induce a connected subgraph (say P_b^i) of P_b (see Figure 2.1). Since every node of $P_b \backslash b$ is given the same label by any derivation function, it follows that the nodes of each P_b^i $(1 \leq i \leq s)$ will be assigned to states in a cyclic manner—whose period is no greater than the number s of states. Of course, for $2 \leq i \leq s$, the states of $V(P_b^i)$ will exhibit a different cyclic pattern depending upon the state to which the root of P_b^{i-1} is assigned. Thus, the state of b can be computed as a function of its label and (for $1 \leq i \leq s$) the residue of $|S_i \cap C|$, modulo some constant (which is at most s). This can be encoded using the **card** predicates of CMS logic—see Courcelle [Cou90b, Proposition 5.4] for details. □

It is clear that the transition function of a tree automaton can be implemented by looking up values in a finite-size table. Therefore, only constant work is required to compute the state of each node. Since tree decompositions can be generated in linear time [Bod93], it follows that a tree automaton solves any CMS-definable graph decision problem in linear time over partial k-trees. If the converse of Theorem 2.8.3 holds, then CMS logic would elegantly characterize the class of problems that can be solved in linear time over partial k-trees with tree automata. To prove this converse, our work is now very clearly cut out: By Lemmas 2.7.11 and 2.8.5, we need only obtain existentially-defined CMS predicates that describe a tree decomposition of the evaluation graph.

2.9 The Pumping Lemma

In this section we develop a *pumping lemma* to show that tree automata cannot recognize certain subclasses of the partial k-trees; it also follows (by Theorem 2.8.3) that such a subclass cannot be defined by a CMS statement. This pumping lemma is a relatively straightforward generalization of the pumping lemma for conventional finite-state automata [HU79]. Although the author is unaware of a pumping lemma having previously been published in this particular form, the idea of applying a pumping lemma to graph problems is not new: Mahajan and Peters [MP94] used a pumping lemma to show that graph-subgraph pairs (G, S) must satisfy a certain "locality" condition in order to be recognized by a tree automaton. In this context, the input to a tree automaton is formed from a partial k-tree G and a putative solution $S \sqsubseteq G$ to some search problem. Under our interpretation, however, a tree automaton is used directly to solve a decision problem—which may be the question of whether G has a subgraph S satisfying some condition, but a putative solution does not form part of the input. So we modify the pumping lemma accordingly.

We assume the reader is familiar with the pumping lemma for conventional finite-state automata [HU79]. The input to such machine can be interpreted as a rooted path P (as discussed at the beginning of Section 2.8) with a labeling function $\sigma : V(P) \rightarrow \Sigma$, for some finite alphabet Σ. The conventional pumping lemma is a consequence of the observation that—provided P is sufficiently long—two of its nodes (say b and d) must become assigned to the same state. Say b is an ancestor of d; let c be the parent of d; and let r be the root of P. The input string is thus $[\ldots\sigma(d), \sigma(c), \ldots, \sigma(b), \ldots, \sigma(r)]$. Since the state of c is computed as a function of $\sigma(c)$ and the state of d (which is the same as the state of b), it follows that r will still be assigned to the same state if a second copy of the path between c and b is inserted, giving

$$[\ldots\sigma(d), \sigma(c), \ldots, \sigma(b), \sigma(c), \ldots, \sigma(b), \ldots, \sigma(r)].$$

We wish to apply this idea in the situation where σ is a derivation function (Prop. 2.6.7) on P for some graph G. It is not always permissible to have a node labeled $\sigma(c)$ become the parent of a node labeled $\sigma(b)$. So we require that, in addition to having the same state, the nodes b and d have the same label $\sigma(b) = \sigma(d)$. In this situation, there is a rooted tree decomposition $(P, r, \mathcal{X})$ of G; each non-drop vertex of d belongs to X_c; and $G_{[X_b]} \cong G_{[X_d]}$. So we can "pump" G into a new graph by *fusing* the non-drop vertices of b (in the subgraph underlying P_b) with the copies of the non-drop vertices of d (in a copy of $G \backslash X_{P_d}$).

Recall that a terminal set (Def. 2.6.3) of a partial k-tree G is a set of k or fewer vertices that belong to a common bag of some tree decomposition of G; and $V_{\text{term}}(G)$ is a specially-designated terminal set—which must be a subset of the root bag of any rooted tree decomposition of G.

Proposition 2.9.1. *Suppose G_1 and G_2 are disjoint partial k-trees. Suppose further that V_1 and V_2 are terminal sets of G_1 and G_2 (respectively) such that $G_{1[V_1]} \cong G_{2[V_2]}$; and let $f : V_1 \rightarrow V_2$ be a corresponding isomorphism. A partial k-tree (say G) is obtained from $G_1 \sqcup G_2$ by adding an edge $\{v, f(v)\}$ for each $v \in V_1$, and then contracting each such edge. We say that G is obtained by* fusing *V_1 with V_2.*

Proof. The fact that G is a partial k-tree follows easily from Lemma 2.5.2. $\square$

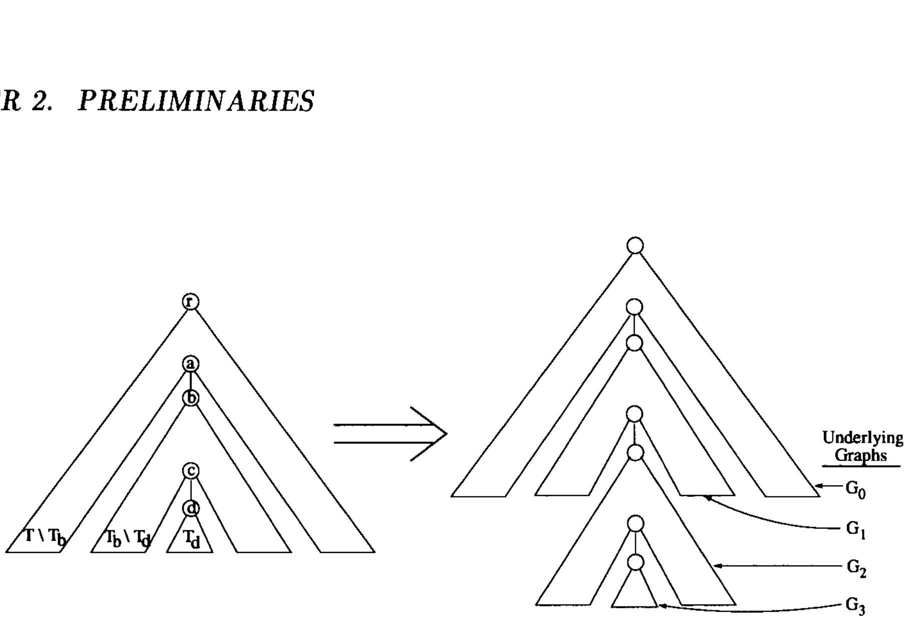

Figure 2.2: Pumping a decomposition tree

Definition 2.9.2. Suppose $(T, r, \mathcal{X})$ is a width-k rooted tree decomposition of a graph G; and let $\sigma : V(T) \to \Sigma_k$ be a derivation function for G (where Σ_k is the k-derivation alphabet). Suppose further that $b \in V(T)$ is an ancestor of $d \in V(T)$ such that $\sigma(b) = \sigma(d)$; and let $\ell \in \mathbb{N}$.

- If $b = r$, then let G_0 be a copy of the subgraph induced by $V_{\text{term}}(G)$; otherwise, let G_0 be a copy of the subgraph underlying $T \backslash T_b$. Let $V_0 \subseteq V(G_0)$ be comprised of the copies of the non-drop vertices of b.

- For $1 \leq i \leq \ell$, let G_i be a copy of the subgraph underlying $T_b \backslash T_d$. Let $V_i \subseteq V(G_i)$ be comprised of the copies of the non-drop vertices of d. Let $V_{\text{term}}(G_i)$ be comprised of the copies of the non-drop vertices of b.

- Let $G_{\ell+1}$ be a copy of the subgraph underlying T_d. Let $V_{\text{term}}(G_{\ell+1})$ be comprised of the copies of non-drop vertices of d.

We define $G[T \backslash T_b \cdot (T_b \backslash T_d)^\ell \cdot T_d]$ to be the graph obtained from $G_0 \sqcup G_1 \sqcup \ldots \sqcup G_{\ell+1}$ by fusing V_i with $V_{\text{term}}(G_{i+1})$, for $0 \leq i \leq \ell$.

Figure 2.2 illustrates the idea behind Definition 2.9.2: Here, a and c are the parents of b and d (respectively). The diagram shows a tree decomposition of G being "pumped" into a tree decomposition of $G[T \backslash T_b \cdot (T_b \backslash T_d)^2 \cdot T_d]$. Note that if the subgraph underlying $T_b \backslash T_d$ is the null graph, then the "pumped" graph is identical to G. We are now ready to present the pumping lemma, using the notion of a trunk (Def. 2.2.1) in T:

Pumping Lemma 2.9.3. *If a subclass* Π *of the partial k-trees can be recognized by a tree automaton, then there exists a constant* m *such that the following is true: If* $(T, r, \mathcal{X})$ *is a rooted tree decomposition of* $G \in \Pi$, *and* P *is a trunk in* T *of length* m *(or more), then* P *contains nodes* b *and* d *(where* b *is an ancestor of* d*) such that* $G[T \backslash T_b \cdot (T_b \backslash T_d)^\ell \cdot T_d]$ *belongs to* Π, *for any* $\ell \in \mathbb{N}$.

Proof. Suppose Π is recognized by a tree automaton $\mathcal{A}$ over the k-derivation alphabet Σ_k. Suppose further that $(T, r, \mathcal{X})$ is a width-k rooted tree decomposition of a graph $G \in \Pi$. Let $(T', r, \mathcal{X}')$ be a binary representation of $(T, r, \mathcal{X})$, as constructed by Definition 2.8.4; and let σ be a derivation function for G on T'. If T contains a trunk (say P) with more than $s|\Sigma_k|$ nodes, then all these nodes also belong to some trunk in T'; hence, there exist two nodes $b, d \in V(P)$ (say b is an ancestor of d) such that $\sigma(b) = \sigma(d)$, and both b and d become assigned to the same state when the binary tree T' is input to $\mathcal{A}$. It follows easily from Definition 2.9.2 that, for any $\ell \in \mathbb{N}$,

$$G[T' \backslash T'_b \cdot (T'_b \backslash T'_d)^\ell \cdot T'_d] = G[T \backslash T_b \cdot (T_b \backslash T_d)^\ell \cdot T_d].$$

Furthermore, a binary tree decomposition of this graph can easily be constructed from $(T', r, \mathcal{X}')$ by splicing in $\ell - 1$ extra copies of $T'_b \backslash T'_d$. After doing this, each node will still be assigned to the same state as before; so this tree decomposition will also be accepted by $\mathcal{A}$. Therefore, the constant required by the lemma is $m = s|\Sigma_k| + 1$. $\square$

Chapter 3

Definability Equals Recognizability

In this chapter we prove that a graph decision problem can be *defined* in CMS logic if the partial 3-trees (and k-connected partial k-trees) that are yes-instances can be *recognized* by a finite-state tree automaton. The converse—definability implies recognizability—is given by Theorem 2.8.3 over all partial k-trees. It has been conjectured [Cou91] that recognizability implies definability over partial k-trees; but a proof was previously known only for $k \leq 2$. In this chapter we prove the conjecture—and hence the equivalence of definability and recognizability—over partial 3-trees and k-connected partial k-trees. These results are also available in [Kal96].

To prove recognizability implies definability of partial 3-trees, we suppose there exists a tree automaton $\mathcal{A}$ that accepts a width-3 tree decomposition iff the underlying graph belongs to some subclass (say Π) of the partial 3-trees. Our goal is to derive a CMS statement defining Π. We proceed by showing that any partial 3-tree admits a particular tree decomposition that can be described (Def. 2.7.3) by existentially-defined CMS predicates **Bag** and **Parent**. Then, we apply Lemmas 2.7.11 and 2.8.5 to conclude that a CMS statement can encode whether or not such a tree decomposition is accepted by $\mathcal{A}$

This chapter is organized as follows: In Section 3.1 we show that any connected partial k-tree admits a *simple* tree decomposition; and we describe how *trunk-graphs* are derived from such a tree decomposition. In Section 3.2 we decompose a 2-connected partial 3-tree G into a tree-like hierarchy of these trunk-graphs; and we develop several important properties of the hierarchy. In Section 3.3 we show how those properties enable CMS predicates to encode the vertex set and edge set of each trunk-graph; and in Section 3.4 we show how they enable CMS predicates to encode a particular structure—called a *pyramid*—in each trunk-graph. These results are combined in Section 3.5 to show that CMS predicates can describe a fixed tree decomposition for any 2-connected partial 3-tree. In Section 3.6 we generalize that result to all partial 3-trees, and draw the conclusion that recognizability implies definability of partial 3-trees. Finally, in Section 3.7 we explain how the proof is generalized to k-connected partial k-trees.

3.1 Simple Tree Decompositions

To develop a canonical tree decomposition, we begin with a *simple* tree decomposition, and then modify it. In this section we show that any connected partial k-tree admits a simple tree decomposition—which is a rooted tree decomposition $(T, r, \mathcal{X})$ in which each node of T satisfies three special properties. We then discuss the structure of a *trunk-graph*—which is the graph obtained by adding certain edges to the subgraph underlying a trunk (Def. 2.2.1) in T. We show that a trunk-graph can be represented by a *pyramid* consisting of k vertex sequences.

Definition 3.1.1. A *simple* tree decomposition is a rooted tree decomposition $(T, r, \mathcal{X})$ 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 non-drop vertices of b, then V' is not a cut-set of the subgraph underlying T_b.

We will need the following consequence of the above properties:

Lemma 3.1.2. *Suppose $(T, r, \mathcal{X})$ is a simple tree decomposition. If $a \in V(T)$ is the parent of $b \in V(T)$, then X_b contains the drop vertex of a.*

Proof. Suppose not, and let v be the drop vertex of a. So $X_a \cap X_b$ is a cut-set that separates v from the drop vertex of b (contradicting property **P3** for node a). □

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 (Def. 2.6.3). 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 3.1.3. *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 $(T, r, \mathcal{X})$ 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 $(T, \mathcal{X})$ 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 2.6.8). 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 2.6.9) 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, $(T, r, \mathcal{X})$ 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 3.1.4. *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 3.1.3, G admits a width-k rooted tree decomposition $(T, r, \mathcal{X})$ 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 $H + X_c$. 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 non-drop vertices of c is a cut-set of G^c. We can now assume, without loss of generality (by Lemma 3.1.3), 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. □

A tree decomposition $(P, \mathcal{X})$ for which P is a path is sometimes called a *path decomposition.* For this to be a *simple* path decomposition, the path P must be rooted at one of its endpoints.

Definition 3.1.5. 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.

By Lemma 3.1.4, any connected partial k-tree admits a simple tree decomposition. The analogous statement, however, does not hold for connected partial k-paths: *i.e.* not every connected partial k-path is a simple partial k-path. In Section 3.2, we will decompose a partial k-tree into a collection of simple partial k-paths by recursively choosing trunks (Def. 2.2.1) in the tree of a simple tree decomposition.

Definition 3.1.6. Suppose $(T, r, \mathcal{X})$ 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 non-drop vertices of the root of P.

If P 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 3.1.7. *Suppose $(T, r, \mathcal{X})$ 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. To complete the proof, we need only show that each node of P satisfies properties **P1**, **P2** and **P3** relative to the trunk-graph R.

Since each node of T satisfies **P1** relative to the graph G, each node of P has exactly one drop vertex; hence, **P1** is satisfied by each node relative to R. If b is the leaf of P, then the subgraph (of R) underlying P_b is identical to the subgraph (of G) underlying T_b; so **P2** and **P3** are also 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 Lemma 3.1.2, 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 3.1.6, 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. □

Corollary 3.1.8. *R is a simple partial k-path iff R is the trunk-graph of some trunk in the tree of some width-k simple tree decomposition.*

Proof. Suppose R is a simple partial k-path. Let $(P, r, \mathcal{X})$ be a width-k simple path decomposition of R. So P is a trunk in P, and R is the trunk-graph of P. The converse has been established by Lemma 3.1.7. □

We now define a structure called a *pyramid*, and show that any simple partial k-path (*i.e.* any trunk-graph) contains such a structure. A pyramid consists 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 3.1.9. 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 $(A_1, \rightarrow_1)$, $(A_2, \rightarrow_2), \ldots, (A_k, \rightarrow_k)$ with the following properties:

D1: $\{A_1, A_2, \ldots, A_k\}$ is a partition of $V(R) - \{v_1\}$.

D2: For $1 \leq i \leq k$: $v \in A_i$ is adjacent to v_1 only if v is the first vertex of $(A_i, \rightarrow_i)$.

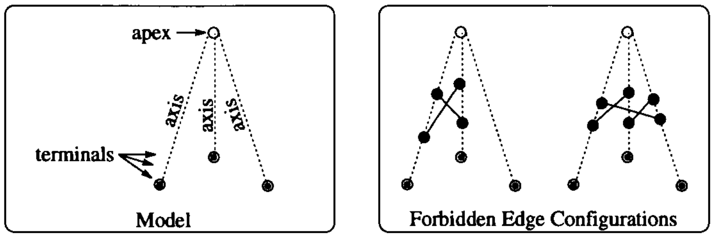

Figure 3.1: A pyramid in a simple partial 3-path

D3: For $1 \leq i \leq k$: v belongs to $A_i \cap V_{\text{term}}(R)$ only if v is the last vertex of $(A_i, \rightarrow_i)$.

D4: For $1 \leq i \leq k$: two vertices $u, v \in A_i$ are adjacent (*i.e.* $\{u, v\} \in E(R)$) only if $u \rightarrow_i v$.

D5: For $2 \leq \ell \leq k$: if $i_1, i_2, \ldots, i_\ell$ are distinct indices between 1 and k, and each A_{i_j} $(1 \leq j \leq \ell)$ 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.

Figure 3.1 illustrates the structure of 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 in any simple partial k-path, we will visit the bags of a simple path decomposition in order from the leaf to the root. Each bag contains at most $k+1$ vertices: A vertex being seen for the last time is called a *drop* vertex (Def. 2.6.5); and a vertex being seen for the first time is called an *add* vertex:

Definition 3.1.10. 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 *non-add* vertex of b.

Lemma 3.1.11. *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 non-drop vertex of the leaf become the first element of a distinct axis. This establishes the basis of the following

Inductive Hypothesis. For $b \in V(P)$, there is a pyramid in the trunk-graph (say R^b) of P_b, where $V_{\text{term}}(R^b)$ is comprised of the non-drop vertices of b.

Suppose a is the parent of b. Since b has a drop vertex, let $h \leq k$ such that $h = |X_a \cap X_b|$. Inductively (by property **D3**), each vertex in $X_a \cap X_b$ is the last element of a distinct axis, say $(A_i, \rightarrow_i)$ for $1 \leq i \leq h$. Without loss of generality, assume the last element of $(A_h, \rightarrow_h)$ is the drop vertex of a (see Lemma 3.1.2). There are no more than $k+1-h$ add vertices of a; so a pyramid in R^a can be formed by putting each of these add vertices at the end of a distinct axis $(A_i, \rightarrow_i)$, for $h \leq i \leq k$. It is clear that this pyramid satisfies properties **D1**, **D2** and **D3**. Furthermore, R^a has at most one axial edge that does not also belong to R^b: that is, a (possible) edge between the last two vertices of A_h. So property **D4** is inductively satisfied.

Suppose the pyramid in R^a does not satisfy property **D5**. 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 $u_{i_j} \rightarrow^+_{i_j} u'_{i_j}$ $(1 \leq j \leq \ell)$. Inductively, not all of these edges exist in R^b; so assume without loss of generality that u'_{i_1} is an add vertex of a. Now, u'_{i_1} belongs to no bag other than X_a, and u'_{i_1} is adjacent to u_{i_ℓ}. Therefore, u_{i_ℓ} belongs to X_a. Since $u_{i_\ell} \rightarrow^+_{i_\ell} u'_{i_\ell}$, it follows that u'_{i_ℓ} also belongs to X_a. Hence, $(A_{i_\ell}, \rightarrow_{i_\ell})$ is the unique axis containing more than one vertex of X_a (*i.e.* the index i_ℓ equals the index h from the previous paragraph). Now, u'_{i_ℓ} is an add vertex of a, but it is adjacent a vertex $u_{i_{\ell-1}} \notin X_a$ (a contradiction). □

Each of the axes in a pyramid gives part of an elimination order for the vertices of the corresponding partial k-path. By interleaving these orders in a fixed manner, we can obtain a rooted path decomposition in which each bag contains four vertices: The leaf bag contains the apex as well as the first vertex of each axis; and each other bag contains, inductively, the k maximal vertices of the preceding (child) bag, and one additional vertex that is an immediate successor along one of the axes. In Section 3.5, we will show how this can be encoded in CMS logic.

3.2 A Trunk Hierarchy

In this section we decompose a 2-connected partial 3-tree G into a tree-like hierarchy of trunk-graphs (Def. 3.1.6). We begin with a width-3 simple tree decomposition of G, and recursively partition its nodes into trunks, so that each corresponding trunk-graph satisfies three special properties. Later in this chapter we will use these properties to show that CMS predicates can describe (Def. 2.7.3) a tree decomposition of each trunk-graph.

Definition 3.2.1. Suppose $(T, r, \mathcal{X})$ 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 $(T, r, \mathcal{X})$.

Suppose $\mathcal{R}$ is the trunk hierarchy obtained by partitioning the nodes of T into a collection $\mathcal{P}$ of trunks. Let T' be the tree obtained by contracting (Def. 2.5.1) each trunk in $\mathcal{P}$ into a single node. The trunk-graphs in $\mathcal{R}$ have an obvious one-to-one correspondence with the nodes of T'. We will use the terms "root", "child", "parent" *etc.*, with implied reference to T', when speaking of these trunk-graphs.

Remark. The pair $(T', \mathcal{R})$ describes a structure 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 the terminal set $V_{\text{term}}(G)$ is a cut-set of G; hence, by Lemma 3.1.4, 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 3.2.2, 3.2.4 and 3.2.6. Our discussion describes how to obtain a simple tree decomposition $(T, r, \mathcal{X})$ with a trunk P between r and a leaf of T; 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 terminal set $X_p \cap X_c$ of each such subgraph has cardinality two or three; and furthermore, the subgraph is a *2-connected* partial 3-tree (under the assumption that $X_p \cap X_c$ induces a clique). All the results of this section would still carry through should we renounce the assumption that G is 2-connected, and assume instead that a 2-connected graph be obtained from G by adding edges between each pair of non-adjacent terminals. Therefore, we can apply the techniques of this section recursively to obtain all of the trunk-graphs comprising $\mathcal{R}$.

The first property gives an order on the vertex set of each trunk-graph. In Section 3.3 we will define a CMS predicate to identify each such vertex set, inductively, in this order:

Property 3.2.2. *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 non-terminal 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 $V_{\text{term}}(G)$.*

In the proof of Theorem 3.2.9, we show that Property 3.2.2 is enforced by having each trunk-graph correspond to a *centered* trunk of a simple tree decomposition $(T, r, \mathcal{X})$. This is a trunk P, rooted at r, for which each node satisfies two other properties in addition to **P1**, **P2** and **P3** (Def. 3.1.1). When we refer to a vertex v as an *add* vertex of $p \in V(P)$, we mean that Definition 3.1.10 is to be interpreted relative to the path decomposition $(P, r, \mathcal{X}_P)$: that is, v does not belong to the bag X_c indexed by the child $c \in V(P)$ of p, but v may belong to the bag $X_{c'}$ indexed by any child $c' \in V(T \backslash P)$.

Proposition 3.2.3. *G admits a width-3 simple tree decomposition $(T, r, \mathcal{X})$ such that T contains a* centered *trunk P: this is defined as a trunk 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 $(T, r, \mathcal{X})$ is a simple tree decomposition of G. For any node b of T, each bag indexed by a child of b contains the drop vertex of b (see Lemma 3.1.2); and by property

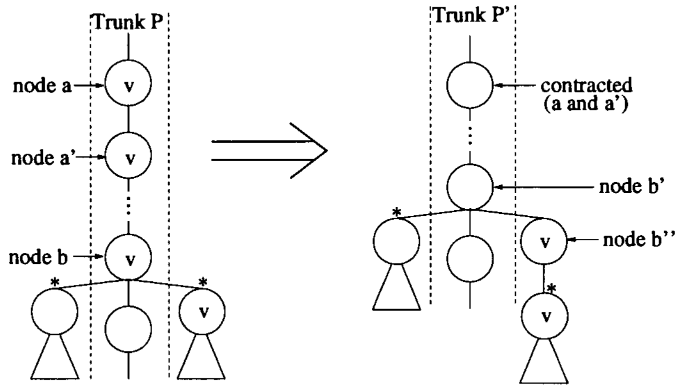

Figure 3.2: Enforcing **P5**
("$*$" indicates there may be multiple similar subtrees)

P1 (Def. 3.1.1), each of these bags contains at most three vertices of X_b. It follows 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.

Suppose a node b of 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 $\mathcal{X}_{T_b}$. By Lemma 3.1.2, v is a non-drop vertex of b; so let $a \in V(P)$ be the ancestor of b such that v is the drop vertex of a. Let $a' \in V(P)$ be the child of a (possibly $a' = b$). The situation is illustrated in the left-hand-side of Figure 3.2. It follows from property **P3** (Def. 3.1.1) that a has no add vertex. Therefore $X_a \subseteq X_{a'}$, and we can contract the edge $\{a, a'\}$ (see Definition 2.6.8) without violating **P2** or **P3**. The contracted node now has two drop vertices (including v); so we delete v from each bag indexed by $T \backslash T_b$. At this point, b is the only node with two drop vertices.

Now, we split b (see Definition 2.6.9) into two nodes b', b'' (with b' the parent of b''). Let $X_{b'} \leftarrow X_b - \{v\}$; let $X_{b''} \leftarrow X_b$; and for each child c of b, let its parent become b'' if $v \in X_c$, and b' otherwise. Thus b' becomes the parent of the unique child belonging to P. It is not difficult to verify that each node of T continues to satisfy **P1**, **P2** and **P3** after this construction. 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**. $\square$

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. These modifications will be carried out such that **P4** and **P5** continue to hold for each node of the centered trunk, and the terminals of G are retained in the root bag. The first operation (Proposition 3.2.5) results in one additional vertex in the trunk-graph. This is used to enforce the following property:

Property 3.2.4. *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 $u, v \in V(H)$ such that $\{u, v\} \in E(H) - E(G)$, then G contains two internally-vertex-disjoint paths between u and v for which each internal vertex is a non-terminal of some descendant of R.*

Suppose $(P, r, \mathcal{X}_P)$ is a simple path decomposition of R. By Definition 3.1.1, there is a path $H \sqsubseteq R$ between any pair y, z of vertices of R such that no internal vertex of H is a terminal (*i.e.* condition **H1** is satisfied). If condition **H2** is not satisfied, then there is a node $b \in V(P)$ such that X_b contains the endpoints (say u and v) of some edge in $E(H) - E(G)$, and there are not two internally-vertex-disjoint paths between u and v in the subgraph induced by $V(G) - X_P \cup \{u, v\}$. By Lemma 2.3.2, then, there is a cut-vertex x that separates u from v in that subgraph. The following proposition can be used to "promote" x to a centered trunk of a perturbed simple tree decomposition (see Figure 3.3). It turns out that we need only consider those cases where v is an add vertex of b and u is the drop vertex of b.

Proposition 3.2.5. *Suppose P is a centered trunk of a width-3 simple tree decomposition $(T, r, \mathcal{X})$ 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** (Def. 3.1.1), b has a child $c \in V(T \backslash P)$ such that $u, v \in X_c$. If there is a cut-vertex x of G' that separates u from v, then this child c is unique (since G' contains

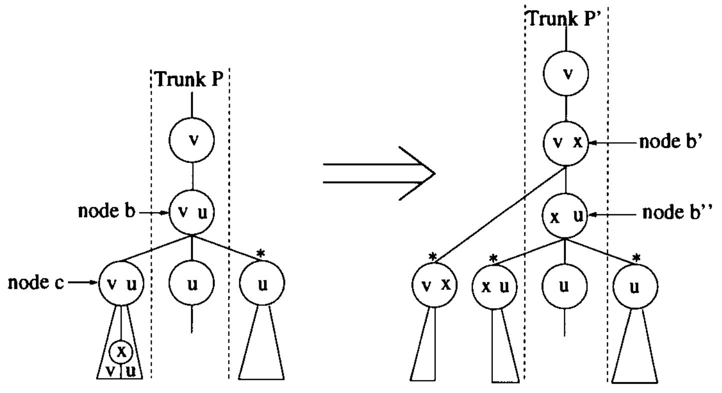

Figure 3.3: Promoting a vertex

a distinct path between u and v for each such child). Hence, the cut-vertex 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 $1 \le i \le \ell$, let $G_i = G'_i + V'_i$ where V'_i is the largest subset of $X_b \cup \{x\}$ in which each vertex is adjacent to one or more vertices of G'_i. Now a tree decomposition of G_i $(1 \le i \le \ell)$ is obtained from $(T_c, \mathcal{X}_{T_c})$ as follows: First, we delete from each bag all vertices not in $V(G_i)$; and then we add x to each bag indexed by a node on the path between c and some node whose bag originally contained x (this is possible because at least one vertex, not in $V(G_i)$, was deleted from each of those bags). It follows from Lemma 3.1.4 that there exists a width-3 simple tree decomposition $(T_i, r_i, \mathcal{X}_i)$ of G_i, where $V_{\text{term}}(G_i) = V'_i$. (Note that V'_i does not contain both u and v).

To construct $(T', r', \mathcal{X}')$, we split b into two nodes b', b'', where b' becomes the parent of b'' (see Def. 2.6.9). Let P' be the trunk (in T') so derived from P. Let $X_{b'} = X_b \cup \{x\} - \{u\}$; and let $X_{b''} = X_b \cup \{x\} - \{v\}$. For each child $c' \in V(T \backslash T_c)$ of b, the bag $X_{c'}$ contains u but not v: so let c' become a child of b''. To complete the construction of $(T', r', \mathcal{X}')$, we let each r_i $(1 \le i \le \ell)$ become a child of either b' or b'', depending upon whether v or u is contained in X_{r_i}. This construction performs only localized modifications to $(T, r, \mathcal{X})$: For each node $a \in V(T \backslash T_c) - \{b\}$, the bag X_a is unchanged, the subgraph underlying T_a is unchanged, and a has the same drop vertex. So a continues to satisfy properties **P1** to **P3** (Def. 3.1.1); and if a was a node of P, it continues to satisfy properties **P4** and **P5** (Prop. 3.2.3). By Lemma 3.1.4, the nodes of each T_i $(1 \le i \le \ell)$ satisfy **P1** to **P3**. It is not difficult to verify that the new nodes b' and b'' also satisfy **P1** to **P5**. □

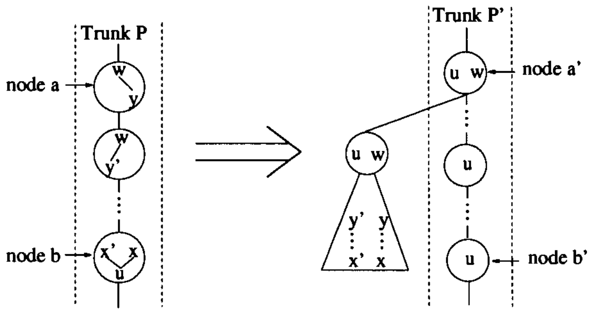

Figure 3.4: Demoting vertices

One other property is needed for a CMS formula to encode the axes of a fixed 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 Def. 3.1.9). The following property ensures that there are enough cross-edges between maximal path in different axes, so that a CMS formula can determine their order within each axis.

Property 3.2.6. *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 either*

- *some internal vertex (of H or H') is a non-drop vertex of a, or*
- *some internal vertex is a non-add vertex of b, or*
- *some internal vertex 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 $(T, r, \mathcal{X})$, as described in the following proposition. This operation yields a new centered trunk P' for which $X_{P'}$ has fewer vertices than X_P. Hence, the operation can be applied repeatedly until Property 3.2.6 is satisfied.

Proposition 3.2.7. *Suppose P is a centered trunk of a width-3 simple tree decomposition $(T, r, \mathcal{X})$. Suppose further that Proposition 3.2.5 cannot be applied to P. If $(P, r, \mathcal{X}_P)$ does not satisfy the statement of Property 3.2.6, then there exists a centered trunk P' of some* *width-3 simple tree decomposition of G, such that Proposition 3.2.5 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 3.2.6. So P has nodes a and b (where a is an ancestor of b) such that there are internally-vertex-disjoint paths H and H' between the drop vertex (say u) of b and some vertex (say w) in X_a such that:

- no internal vertex (of H or H') is adjacent to any vertex in $V(R) - V(H) - V(H')$,
- no internal vertex belongs to $V_{\text{term}}(R)$, and
- no internal vertex belongs to a bag indexed by a descendant of b or an ancestor of a.

Figure 3.4 shows the paths H and H' with vertex sequences $(u, x \dots y, w)$ and $(u, x' \dots y', w)$. Possibly one of these paths has no internal vertices (in which case $w \in X_b$).

Let G' be the subgraph consisting of those components of $G \backslash \{u, w\}$ containing one or more vertices of $H \sqcup H'$. Let the (say ℓ) components of G' be enumerated $G'_1, G'_2, \dots, G'_\ell$. Using an argument similar to that used in the proof of Proposition 3.2.5, we obtain a width-3 simple tree decomposition $(T_i, r_i, \mathcal{X}_i)$ of $G'_i + \{u, w\}$ such that $\{u, w\}$ is the terminal set.

We can construct the tree decomposition $(T', r', \mathcal{X}')$ from $(T, r, \mathcal{X})$ as follows: We delete those subtrees $T_c \sqsubseteq T$ whose bags contain only vertices of $G' + \{u, w\}$; each such subtree is rooted at a child c of a node on the path (in T) between a and b. We then delete any vertices of G' from each bag of that path. Since at least one vertex is deleted from each of these bags, we can then add u to each of them, and let each r_i $(1 \leq i \leq \ell)$ become a child of a. To complete the construction, we need only contract (Def. 2.6.8) 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** to **P5**. Furthermore, this operation does not create any new instances where Proposition 3.2.5 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 modified trunk P'. $\square$

Lemma 3.2.8. *G admits a width-3 simple tree decomposition for which the trunk-graph corresponding to some centered trunk satisfies Properties 3.2.4 and 3.2.6.*

Proof. Let $(T, r, \mathcal{X})$ be a simple tree decomposition of G such that $P \sqsubseteq T$ is a centered trunk (Prop. 3.2.3). Without loss of generality, assume neither Proposition 3.2.5 nor 3.2.7 can be applied to P. It follows immediately that Property 3.2.6 is satisfied by the corresponding trunk-graph (say R). Let v_1 be the drop vertex of the leaf of P.

Claim. 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 3.2.4).

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 Lemma 3.1.2, the drop vertex (say u) of b is not an add vertex of b; so u is a non-terminal vertex of R that does satisfy the claim. Now v would also satisfy the claim if there were an edge of G between u and v. By property **P3** (Def. 3.1.1), $X_b - \{u, v\}$ does not separate u from v in the subgraph underlying T_b. So b has a child $c \in V(T \backslash P)$ for which $u, v \in X_c$. If two such children exist, then v would satisfy the claim—there being a path between u and v whose internal vertices belong to $X_{T_c} - X_b$ for each child c. Hence, b has a unique child c whose bag contains both u and v. Since Proposition 3.2.5 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 2.3.2, 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 3.2.9. *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 3.2.2, 3.2.4 and 3.2.6.*

Proof. Let $(T, r, \mathcal{X})$ be a width-3 simple tree decomposition of G. Without loss of generality (by Lemma 3.2.8), assume that T contains a centered trunk whose trunk-graph satisfies Properties 3.2.4 and 3.2.6. 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 3.2.4 and 3.2.6. 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 3.2.2. So suppose $R \in \mathcal{R}$ is the trunk-graph of $P \sqsubseteq T$. We choose the first vertex v_1 of the sequence to be the drop vertex of the leaf of P. By property **P3** (Def. 3.1.1), v_1 is adjacent to each other vertex of the leaf bag; hence, these vertices satisfy condition **C1** (of Property 3.2.2) if they follow v_1 in any permutation. This establishes the basis of the following

Inductive Hypothesis. For $c \in V(P)$, there is a sequence on X_{P_c} in which each vertex satisfies condition **C1**, **C2** or **C3**.

Suppose $b \in V(P)$ is the parent of c; and let v_j be the drop vertex of b. So v_j is not a terminal of R, and (by Lemma 3.1.2) v_j belongs to X_c. Since G is 2-connected, there are at most two vertices in $X_b - X_c$. We will show that the vertex sequence on X_{P_b} can be formed by placing these add vertices of b (in either order if there are two of them) at the end of the inductive sequence on X_{P_c}.

If v_i is an add vertex (of b) that is adjacent to the drop vertex v_j, then v_i satisfies **C1** when placed at the end of the sequence on X_{P_c}. Suppose, then, that v_i is an add vertex that is not adjacent to v_j. By property **P3**, $X_b - \{v_j, v_i\}$ does not separate v_j from v_i in the subgraph underlying P_b. It follows 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. □

3.3 Encoding a Trunk Hierarchy

In this section we develop CMS predicates to existentially encode the vertex set and edge set of each trunk-graph in a trunk hierarchy of a 2-connected partial 3-tree G. In Section 3.4, we will show that the structure of a pyramid in each trunk-graph can also be encoded; and in Section 3.5, we will 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 3.2.9), we assume each trunk-graph $R \in \mathcal{R}$ has Properties 3.2.2 and 3.2.4. We will show that these properties enable CMS predicates to identify the sets $V(R), V_{\text{term}}(R)$ and $E(R)$. Note that each vertex in $V(G) - V_{\text{term}}(G)$ is a non-terminal vertex of exactly one trunk-graph in $\mathcal{R}$.

Definition 3.3.1. If v is a non-terminal vertex of G, then $R(v)$ denotes the unique trunk-graph in $\mathcal{R}$ such that v is a non-terminal vertex of $R(v)$.

By Lemma 2.7.10, a CMS predicate can existentially encode any constant amount of information pertaining to the role of each vertex v in the trunk-graph $R(v)$. This allows a non-terminal vertex (say v_1) to be designated for each trunk-graph. The other vertices of $R(v_1)$ will then be identified inductively, in the order given by Property 3.2.2. We will use a (non-proper) vertex coloring to help identify the vertices in this way:

Proposition 3.3.2. *$V(G)$ can be partitioned into thirteen* color classes *such that the non-terminal 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 assign each terminal of G to any color class. Then, we repeatedly find some trunk-graph $R \in \mathcal{R}$ whose terminals are already colored, but whose non-terminal vertices are not colored. For each terminal $t \in V_{\text{term}}(R)$, there are at most four color classes that cannot be used to color the non-terminal vertices of R: that is, if $t \notin V_{\text{term}}(G)$, then the non-terminal vertices of $R(t)$ all belong to one color class, and $R(t)$ has at most three terminals. It follows that the non-terminal vertices of R can be legally assigned to one of the thirteen color classes. □

Suppose $R \in \mathcal{R}$; and let $v_1, v_2, \ldots v_{|V(R)|}$ be an order on $V(R)$ given by Property 3.2.2. We will describe how each vertex v_i $(2 \leq i \leq |V(R)|)$ is inductively identified, with the help of a unique edge incident to an already-identified non-terminal vertex v_j (where $j \leq i - 1$). Property 3.2.2 gives three different conditions by which the vertex v_i may be identified: In each case, the non-terminal vertex v_j interacts with v_i in some prescribed way: If condition **C1** is satisfied, then $\{v_i, v_j\}$ is an edge of G; so (by Lemma 2.7.10) it can be existentially encoded that the endpoints of this edge belong to the same trunk-graph. Otherwise, v_i and v_j are both terminals of some child (say R') of $R(v_1)$; and v_i will be identified with the help

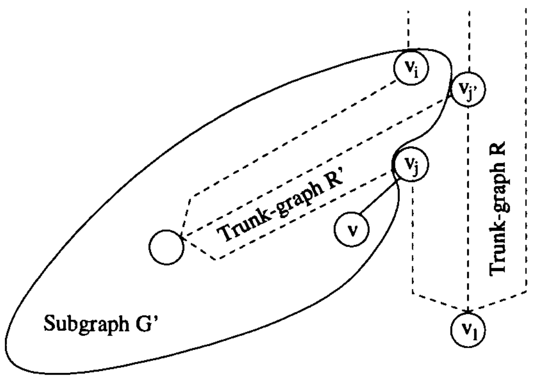

Figure 3.5: G' is not cut by any other vertex with the color of v_i

of a vertex (say v) adjacent to v_j, where v is a non-terminal vertex of either R' or some descendant of R' (see Figure 3.5). We will need the following lemma to show that v_i is the "first" vertex of the correct color that separates v from the terminals of G.

Lemma 3.3.3. *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 non-terminal 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.*

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 $2 \leq \ell \leq d$) $t_\ell \in V_{\text{term}}(R_\ell) - V_{\text{term}}(R_{\ell-1})$. Such vertices exist because $\mathcal{R}$ was admitted by a simple tree decomposition; so each trunk-graph R_ℓ has a terminal that is not a vertex of its parent $R_{\ell-1}$.

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 3.3.2, the color class containing v_i is distinct from the color class containing the non-terminal vertices of each R_ℓ $(1 \leq \ell \leq d)$. Suppose now 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 3.2.4, 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 in $G' \backslash u$. $\square$

We now show how CMS predicates can determine the structure of $R(v_1)$, for a designated vertex v_1 in each trunk-graph. The vertex set of $R(v_1)$ is the minimal set (say V') containing v_1 such that if $v_j \in V'$ belongs to the same color class as v_1, and there is an edge (say e) incident to v_j, then certain other "correctly-colored" vertices are also in V', as specified by Property 3.2.2. The opposite endpoint of e is either another vertex v_i of $R(v_1)$, or some non-terminal vertex v of a descendant of $R(v_1)$. In the latter case, another vertex v_i of $R(v_1)$ is found with the help of Lemma 3.3.3.

Lemma 3.3.4. *Binary CMS predicates* **trunk**, **trunk-edge** *and* $\mathbf{term}_j$ *($1 \le j \le 3$) can be existentially defined such that there is a subset A of $V(G) - V_{\text{term}}(G)$ containing exactly one non-terminal 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* $\mathbf{term}_1(v_1, t) \vee \mathbf{term}_2(v_1, t) \vee \mathbf{term}_3(v_1, t)$ *holds only if t is a terminal of $R(v_1)$.*

Note. If $R(v_1)$ has only two terminals, then say $\mathbf{term}_3(v_1, t)$ is never satisfied.

Proof. Let A be comprised of the the first vertex v_1 of each trunk-graph in an order given by Property 3.2.2. Suppose V' is the vertex set of $R(v_1)$, for some $v_1 \in A$. To encode $\mathbf{trunk}(v_1, V')$, we first identify a superset V'' of V', such that V'' does not contain any non-terminal vertex of any descendant of $R(v_1)$. Each vertex of V'' is identified inductively, using one of the three conditions of Property 3.2.2. Throughout this proof, $v_j \in V''$ is a vertex belonging to the same color class as v_1. Hence, either v_j is a non-terminal vertex of $R(v_1)$, or v_j is an extra vertex in $V'' - V'$ that can be weeded out later. In either case, the CMS formula can force any vertex v_i to belong to V'' if it interacts with v_j to satisfy one of the conditions of Property 3.2.2.

For each edge incident to v_j, we encode (by Lemma 2.7.9) whether its opposite endpoint also belongs to $R(v_j)$. So if condition **C1** is satisfied for a vertex v_i, then v_i can easily be identified for membership in V''. To implement conditions **C2** and **C3**, we will use edges $\{v, v_j\}$ such that v_j is a terminal of $R(v)$. Suppose $R_1 \in \mathcal{R}$ is the child of $R(v_j)$ such that either $R_1 = R(v)$, or R_1 is an ancestor of $R(v)$; hence, v_j is a terminal of R_1. By Lemma 2.7.10, we 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, and require that $v_i \in V''$. Otherwise, since G is 2-connected, it follows that v_i is a cut-vertex of $G\backslash v_j$ that separates v from $V_{\text{term}}(G)$. Suppose $u \in V$ is a vertex distinct from v_i that belongs to the same color class as v_i. It follows from Lemma 3.3.3 that if u is a cut-vertex of $G\backslash v_j$ separating v from $V_{\text{term}}(G)$, then u is not a non-terminal vertex of any descendant of $R(v_j)$; hence, 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. This fact allows a CMS formula to identify v_i as a cut-vertex that places v into a minimal-sized component.

Condition C3: $V_{\text{term}}(R_1) = \{v_i, v_j, v_{j'}\}$ where $j' \leq i - 1$. This case is similar to **C2**. A CMS formula can encode the following: If $v_{j'} \in V''$, and v_i is a cut-vertex of $G\backslash\{v_j, v_{j'}\}$ that separates v from $V_{\text{term}}(G)$, and v belongs to a minimal-sized 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 3.2.2 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 3.2.4) v_i cannot be a non-terminal 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. So V'' does not contain any non-terminal vertex of any descendant of $R(v_1)$.

Now, to identify the set V', we use the same approach (as used to identify V''), except we require that each inductively-identified vertex v_i belongs to V''; and for condition **C3**, we require that v_i be the cut-vertex such that v is in a component of $G\backslash\{v_j, v_{j'}\}$ with minimal size over all choices of $v_{j'} \in V''$. It follows from Lemma 3.3.3 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 simply note that each vertex of $R(v_1)$ is identified with a unique edge. Hence, that edge can encode (by Lemma 2.7.10) the index of any terminal that it is used to identify. Then we can encode that **trunk-edge**(u, v) holds iff either

- $\{u, v\}$ is an edge of G (as encoded by statement (2.7.5)), or

- both u and v are terminals of a common trunk-graph. □

3.4 Encoding a Pyramid

In this section we restrict our attention to a single trunk-graph R belonging to a trunk hierarchy of a 2-connected partial 3-tree G. We develop existentially-defined CMS predicates to encode the axial orders "$\rightarrow_i$ ($1 \leq i \leq 3$) of some pyramid (Def. 3.1.9) in R. These predicates shall be defined over the universe $V(R) \cup E(R)$; and in Section 3.5, we will use these predicates 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 will show (in Section 3.5) how such edges can be represented within a CMS formula. So there is no loss of generality in assuming that $V(R) \cup E(R) \subseteq V(G) \cup E(G)$.

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 3.2.6, 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 3.4.1. *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. As in the proof of Lemma 3.1.11, we construct a pyramid, using $(P, r, \mathcal{X}_P)$, by visiting each node b of P (in order from the leaf to the root), and assigning each add vertex of b to a distinct axis that does not already contain a non-drop vertex of b. To enforce the claim, we merely put an add vertex into the axis containing the drop vertex, before putting one into any other axis. □

By Lemma 3.3.4, a CMS predicate can determine the vertex set $V(R)$, and can associate each terminal $t \in V_{\text{term}}(R)$ with a distinct index j ($1 \leq j \leq 3$). If $v \in V(R) - V_{\text{term}}(R)$, then there is no other trunk-graph R' in the hierarchy such that v is a non-terminal vertex of R'. So by Lemma 2.7.10, we can encode which axial set A_i ($1 \leq i \leq 3$) contains each non-terminal vertex of R. Similarly, using the designated apex of R, we can encode which axis contains the terminal that is associated with each index (see Lemma 3.3.4). For the rest of this section, we assume that the sets $V_{\text{term}}(R), A_1, A_2$ and A_3 are free set variables; and we use these sets to (existentially) encode the order "$\rightarrow_i$" of the vertices in each set A_i.

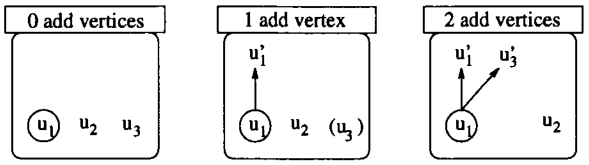

Figure 3.6: The bags of a simple path decomposition
(u_1 is the drop vertex; u'_1 and u'_3, when shown, are add vertices)

Each non-leaf node of P has at most two add vertices (because the underlying graph is 2-connected). By property **P3** (Def. 3.1.1), 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 3.4.1, some such edge is axial—unless b has no add vertex. Figure 3.6 illustrates the three possible situations, depending on how many add vertices there are:

0 add vertices: Hence, there are exactly two non-add, non-drop vertices.

1 add vertex: There is at least one non-add, non-drop vertex—and possibly a second.

2 add vertices: There is exactly one non-add, non-drop vertex.

Each vertex in Figure 3.6 is named u_i or u'_i $(1 \leq i \leq 3)$, where the subscript i 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 (depicted by arrows) directed from u_1 to each add vertex.

By property **D4** (Def. 3.1.9), the subgraph of R induced by each axial set consists of a collection of paths (which we will call *chains*):

Definition 3.4.2. For $1 \leq i \leq 3$, 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 directing the axial edges (see Lemma 2.7.9), a CMS predicate can encode the vertex order within each chain. Therefore, we need only show how CMS can encode the order of the chains in each axial set. If a chain H is adjacent to the apex, then H precedes all other chains; if H contains a terminal, then H follows all other chains. It remains to be shown how a CMS predicate can determine the order of distinct internal chains.

Definition 3.4.3. 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 non-terminal 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 3.4.3: 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 3.4.4. Let $1 \leq i \leq 3$; and suppose H is a chain in A_i. The first vertex of H (with respect to the order "$\rightarrow_i$") 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 $\mathit{Add}(u)$ is called the *source* of H.

In the leftmost panel of Figure 3.6, 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 3.4.5. *A binary CMS predicate β can be existentially defined such that*

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

Proof. Using Lemma 2.7.9, let α, λ, μ be binary CMS predicates that existentially encode edge directions as follows:

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

By Lemma 2.7.6, the transitive closure (denoted α^+) and the reflexive-transitive closure (denoted α^*) of α are also encodable. We now define β as follows:

$$\begin{aligned}\beta(h, v) \equiv (\exists s)(\lambda(s, h) \wedge (\quad & \alpha^+(s, v) \vee \\ & (\exists x, y)(\alpha^*(h, x) \wedge \mu(x, y) \wedge \alpha^*(y, v)) \;\;))\end{aligned} \tag{3.4.6}$$

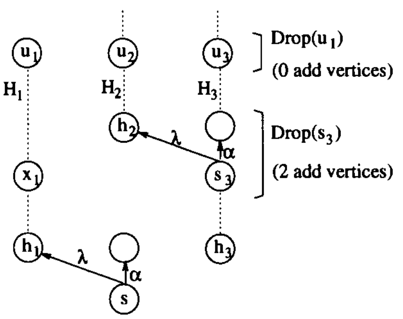

Figure 3.7: Proof of Proposition 3.4.5

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 non-drop vertex v of $Drop(u_1)$. So suppose $\beta(h_1, v)$ is false for each non-drop vertex v of $Drop(u_1)$. By Claim 3.4.1, $Drop(u_1)$ 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 non-drop vertices of $Drop(u_1)$. This is the type of bag illustrated by the leftmost panel of Figure 3.6. For $2 \leq j \leq 3$, let H_j be the maximal axial path containing u_j, and let h_j be the head of H_j (see Figure 3.7).

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 $Add(h_3) \prec Add(h_2)$. Therefore, $Add(h_2)$ contains a vertex of H_1 and a vertex of H_3. It follows that the source (say s_3) of H_2 is a vertex of H_3: s_3 could not be a vertex of H_1, because otherwise $\alpha^*(h_1, s_3) \wedge \mu(s_3, h_2) \wedge \alpha^*(h_2, u_2)$ would be true, contradicting the supposition that $\beta(h_1, u_2)$ is false.

Now $Drop(s_3)$ contains exactly two non-add vertices, namely 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 $\{u_2, u_3\}$. It follows that 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). □

Proposition 3.4.7. *For $1 \leq i \leq 3$, a binary CMS predicate γ can be existentially defined such that*

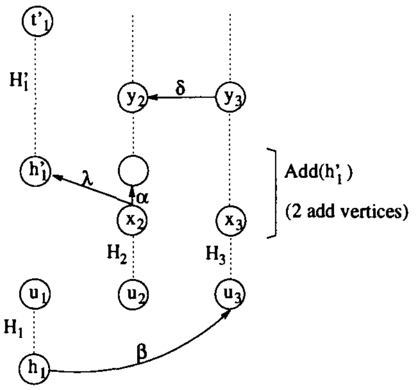

Figure 3.8: Proof of Proposition 3.4.7

- *if $\gamma(h, v)$, then h is the head of some chain in A_i, and $h \rightarrow_i^* v$; 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 3.4.5) to define a subset γ' of γ:

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

By Proposition 3.4.5, $\beta(h, u)$ is true only if h is the head of some chain, and *Add*$(h) \preceq$ *Add*(u). Hence, if u is adjacent to v (where $h, v \in A_1$), then $h \rightarrow_1^* v$. Therefore, the predicate γ' is consistent with the first itemized statement of the proposition. 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 3.8). 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 H_3 be the chains containing u_2 and u_3 (respectively). Since the underlying graph is 2-connected, it follows that *Add*(h_1') contains two non-add vertices, say $x_2 \in V(H_2)$ and $x_3 \in V(H_3)$.

Without loss of generality (by Proposition 3.4.5), we assume $\beta(h_1, u_3)$ is true. By formula (3.4.6), if $u_3 \rightarrow_3^* x$, 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:

$$\alpha^+(x_2, y_2);\ \alpha^*(x_3, y_3);\ \{y_2, y_3\} \in E(R);\ \text{and } Add(h_1') \preceq Add(y_2) \preceq Drop(t_1')$$

So whenever γ' is insufficient to satisfy the second itemized statement of the proposition, we choose some such edge $\{y_2, y_3\}$. Using Lemma 2.7.9, CMS can encode a binary predicate δ to direct these edges: *i.e.* $\delta(y_3, y_2)$. Now, CMS can encode the following subset γ'' of the γ relation:

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

It follows that 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 the claim 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'. We consider three cases:

Case 1: $Drop(t_1')$ contains a vertex of H_2 and a vertex of H_3. Since the underlying graph is 2-connected, and there is no edge between H_1' and H_3, it follows that t_1' is adjacent to some vertex x of H_2 such that $x_2 \rightarrow_2^+ x$. Without loss of generality, assume that t_1' is not adjacent to any vertex that follows x in the axis $(A_2, \rightarrow_2)$. It then follows from Property 3.2.6 that there is an edge between H_3 and some internal vertex of one of the paths $[x_2 \rightarrow_2^+ x, t_1']$ and $[x_2, h_1' \rightarrow_1^* t_1']$ (a contradiction).

Case 2: The tail (say t_2) of H_2 is a non-terminal vertex, 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. Since the underlying graph is 2-connected, it follows that t_2 is adjacent to some vertex (say v_1') of H_1'. But by Property 3.2.6, then, there is an edge between H_3 and some internal vertex of either the path $[x_2 \rightarrow_2^+ t_2]$ or the path $[x_2, h_1' \rightarrow_1^* v_1', t_2]$ (a contradiction).

Case 3: The tail (say t_3) of H_3 is a non-terminal vertex, 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 $x_2 \rightarrow_2^+ y_2$. Since the underlying graph is 2-connected, it follows that t_3 is adjacent to some vertex of H_1' (a contradiction). □

Lemma 3.4.8. *There is a pyramid in each trunk-graph in $\mathcal{R}$ for which each axis $(A_i, \rightarrow_i)$, $1 \leq i \leq 3$, is existentially encodable by a CMS predicate.*

Proof. The vertex order within each chain in A_i is encoded by the α predicate (from the proof of Proposition 3.4.5). A CMS formula can easily determine if a given chain is adjacent to the apex (and hence, it precedes all other chains in A_i) or if it contains a terminal (and hence it follows all other chains in A_i). By Proposition 3.4.7, the CMS formula can determine the order of internal chains in A_i. □

3.5 2-Connected Partial 3-Trees

In this section we develop CMS predicates to describe (Def. 2.7.3) a fixed tree decomposition of a 2-connected partial 3-tree G. To do this, we use a trunk hierarchy $\mathcal{R}$, as constructed in Section 3.2. We have shown in Section 3.3 that CMS predicates can encode the structure of each trunk-graph in $\mathcal{R}$; and in Section 3.4 we showed how to encode the structure of a fixed pyramid in each trunk-graph. In this section we will 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 3.2.9), we assume each trunk-graph in $R \in \mathcal{R}$ has Properties 3.2.2, 3.2.4 and 3.2.6. In Section 3.4 we used Property 3.2.6 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 3.5.1. *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. After this is done, it is not difficult to express any predicate over $V(R) \cup E(R)$ as a disjunction of predicates over $V \cup E$.

Each edge in $E(R) - E(G)$ has endpoints that are terminals of a common child of R. By Lemma 3.3.4, a CMS predicate can determine that a vertex (say v) is the apex of a child of R: this is true iff $V(R)$ contains each vertex t for which $\mathbf{term}_j(v, t)$, $1 \leq j \leq 3$. Each pair of these terminals are the endpoints of an edge of R. The apex v can be used to represent each of the (at most three) edges between its terminals; and these edges can be distinguished by the corresponding indices.

To represent an edge subset $E' \subseteq E(R)$, 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 that represent the edges of $E' - E(G)$. Each of these vertex subsets corresponds to a unique pair of distinct indices j, j' $(1 \leq j < j' \leq 3)$. A vertex v belongs to the corresponding 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 $\{t,t'\} \in E' - E(G)$. □

As explained at the end of Section 3.1, each axis of a pyramid in R gives part of an elimination order on $V(R)$. By interleaving these elimination orders in a fixed manner, we can obtain a path decomposition of R for which the leaf bag contains the apex and the first vertex of each axis; each other bag contains, inductively, the k maximal vertices of the preceding (child) bag, as well as the immediate successor of one of them. Property **D5** (Def. 3.1.9) guarantees that at least one of the maximal vertices has no cross edges extending beyond the others; so we can "advance" along the corresponding axis. These bags are well-defined if we impose an order on the axes, and adopt the convention that we inductively advance along 1st axis whenever possible, otherwise the 2nd axis if possible, and otherwise the 3rd.

Lemma 3.5.2. *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. Let $R \in \mathcal{R}$. By Lemma 3.4.8, the axes $(A_i, \rightarrow_i)$, $1 \leq i \leq 3$, can be encoded for some pyramid in R. By Lemma 2.7.6, the transitive closures "$\rightarrow_i^+$" can also be encoded. We assume that each A_i is non-empty—for otherwise, a simplification of the following argument carries through. We describe how to define the **Bag** predicate for a path decomposition in which each bag contains exactly four vertices; and the bags indexed by adjacent nodes intersect in exactly three vertices. So each non-root bag contains a unique drop vertex—which becomes its witness. The witness of the root bag can be chosen arbitrarily from among its (one or two) drop vertices.

To identify the leaf bag, CMS encodes 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 $u_i, u_i' \in A_i$ $(1 \leq i \leq 3)$ such that $u_i \rightarrow_i u_i'$. The other two vertices $u_j \in A_j$ $(1 \leq j \leq 3;\ j \neq i)$ in X can be identified as follows: Let $u_j' \in A_j$ be the last vertex (in the order "$\rightarrow_j$") that is adjacent to u_i or any vertex that precedes u_i in the i^{th} axis; if

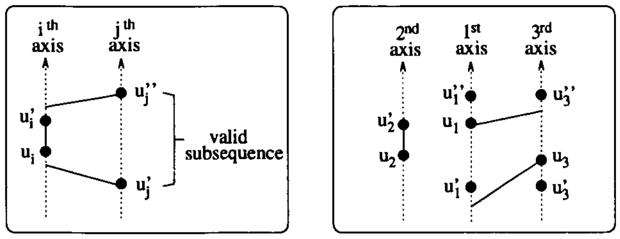

Figure 3.9: Choosing a bag of the CMS-encoded path decomposition

there is no such adjacency, then we let u'_j be the first vertex of the j^{th} 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, u''_j is the last vertex of the j^{th} axis). It is clear that a CMS formula can identify u'_j and u''_j using the "$\rightarrow^+_j$" predicates. It follows from property **D5** (Def. 3.1.9) that $u'_j \rightarrow^*_j u''_j$; and we refer to the subsequence of the j^{th} axis between u'_j and u''_j as the *valid subsequence* (see the left-hand panel of Figure 3.9). 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 $u_j = u''_j$ if $j < i$, and choose $u_j = u'_j$ if $j > i$. In the general case, 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 $j > i$. This choice can be encoded in CMS by considering only the cross edges that are not incident to A_i. Note that, by property **D5** of Definition 3.1.9, there can be no cross edge between a vertex preceding one valid subsequence and a vertex following the other.

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

The right-hand panel of Figure 3.9 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 3.5.2, the cross edges between the second and j^{th} 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. Recall that these cross edges cannot "criss-cross" (as shown in Figure 3.1). 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 (see Definition 3.1.9), there exists no cross edge between a vertex preceding u_1' and a vertex following u_3'': therefore u_1 and u_3 can be chosen to satisfy these constraints.

Lemma 3.5.3. *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 3.2.9. We use Lemma 3.5.2 to encode a path decomposition for each trunk-graph $R \in \mathcal{R}$. 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 3.3.4, a CMS predicate can determine each terminal set $V_{\mathrm{term}}(R)$. If $V_{\mathrm{term}}(R) = V_{\mathrm{term}}(G)$, 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 $V_{\mathrm{term}}(R) \subseteq V(R')$ and $V_{\mathrm{term}}(R) \not\subseteq V_{\mathrm{term}}(R')$. CMS can encode this, and then $\mathbf{Parent}(p, c)$ can be defined 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 $V_{\mathrm{term}}(R) \subseteq X$ when $\mathbf{Bag}(p, X)$ is true. □

3.6 Partial 3-Trees

In Section 3.5, we showed that existentially-defined CMS predicates **Bag** and **Parent** can describe (Def. 2.7.3) a tree decomposition of any 2-connected partial 3-tree G. In this section we generalize this result by removing the requirement that G be 2-connected.

Lemma 3.6.1. *Any connected partial 3-tree admits a width-k rooted tree decomposition that can be described by existentially-defined CMS predicates.*

Proof. Let G be a connected partial 3-tree. A *block* of G is defined as a maximal 2-connected subgraph of G. By Definition 2.3.1, such a block may consist of a single vertex or a pair of adjacent vertices. A CMS predicate can easily be written to encode whether a subgraph of G is a block.

Suppose G has ℓ distinct blocks $G_1, G_2, \ldots, G_\ell$. No pair of these blocks may intersect in more than one vertex—otherwise their union would also be 2-connected (and they would not be *maximal* 2-connected subgraphs). It follows that the ℓ blocks can be arranged in a tree-like hierarchy such that G_i is the parent of G_j only if $|V(G_i) \cap V(G_j)| = 1$. Without loss of generality, we assume that if G_i is the parent of G_j, then the vertex in $V(G_i) \cap V(G_j)$ is a designated terminal in $V_{\text{term}}(G_j)$.

By Lemma 3.5.3, existentially-defined CMS predicates (**Bag** and **Parent**) can describe a tree decomposition of each block G_i. Reviewing the proof of Lemma 3.5.3, we see that no designated terminal (in $V_{\text{term}}(G_i)$) need be assigned to a free variable to encode these predicates. Hence, there is no conflict in simultaneously describing a tree decomposition of each block. The collection of these tree decompositions can easily be assembled into a tree decomposition of G. □

Theorem 3.6.2. *Definability equals recognizability of partial 3-trees.*

Proof. Suppose G is a partial 3-tree. By Lemma 3.6.1, a width-k tree decomposition can be described by CMS predicates for each component of G. We 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 then encode that each other root becomes a child of r. The theorem now follows from Lemmas 2.7.11 and 2.8.5. □

Theorem 3.6.2 gives further support to the following conjecture (see also [Cou91, Conjecture 1]).

Conjecture 3.6.3. *Definability equals recognizability of partial k-trees.*

In Section 3.7 we make further progress towards proving this conjecture, by showing that the proof of Theorem 3.6.2 generalizes to the case of k-connected partial k-trees.

3.7 k-Connected Partial k-Trees

We now generalize the results of this chapter to show that CMS-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 3.1 show that any partial k-tree can be decomposed into a trunk hierarchy (Def. 3.2.1) which is a collection of simple partial k-paths. The results of Sections 3.2 to 3.6 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 3.2.2 can be simplified to the following:

Property 3.7.1. *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 non-terminal 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 3.2 we enforced this property by choosing a *centered* trunk (Prop 3.2.3) in a simple width-k tree decomposition $(T, r, \mathcal{X})$ of G. But 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 any trunk in T is a centered trunk.

Property 3.2.4 can be enforced by a straightforward generalization of Proposition 3.2.5. Property 3.2.6 is always satisfied, because each axis of any pyramid in a trunk-graph $R \in \mathcal{R}$ is a path between the apex and some terminal of G. Hence, a simplification of the proof of Theorem 3.2.9 provides the following:

Theorem 3.7.2. *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 3.7.1 and 3.2.4.*

The vertex set and edge set of each trunk-graph in $\mathcal{R}$ can now be encoded by CMS predicates (as in Section 3.3). Proposition 3.3.2 can easily be generalized as follows:

Proposition 3.7.3. *$V(G)$ can be partitioned into* $4k+1$ color classes *such that the non-terminal 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 3.3.3 remains valid in the case of k-connected partial k-trees (provided $k \geq 2$). Using Lemma 3.3.3, the proof of Lemma 3.3.4 can be easily generalized, to give the following:

Lemma 3.7.4. *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 non-terminal 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 $\mathbf{term}_1(v_1, t) \vee \mathbf{term}_2(v_1, t) \vee \ldots \vee \mathbf{term}_k(v_1, t)$ holds only if t is a terminal of $R(v_1)$.*

We do not need the results of Section 3.4 to encode the axes of a pyramid in R, because each axis consists of a path between the apex and a distinct terminal: Lemma 3.4.8 becomes a trivial consequence of property **D4** (Def 3.1.9). Now, the proof of Lemma 3.5.2 can be easily generalized to prove the following:

Lemma 3.7.5. *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 2.7.11 and 2.8.5, we can draw the following conclusion:

Theorem 3.7.6. *Definability equals recognizability of k-connected partial k-trees.*

Chapter 4

The Complements of CMS-Definable Problems

This chapter is concerned with decision problems for which an instance consists of just a single graph—which is either a yes-instance or a no-instance of the problem at hand. Most of the standard NP-complete graph problems [GJ79] can be captured in this way, after perhaps fixing some other parameters as part of the problem description (we give several examples of this in Section 4.2). A decision problem, then, can be represented by the class of graphs that are yes-instances. In fact, we will use the class of yes-instances to name the corresponding decision problem.

Definition 4.0.1. A decision problem is a class of graphs. If Π is a decision problem, then any graph in Π is called a yes-instance of Π.

A decision problem can be defined, in many cases, by a CMS statement (as explained in Section 2.7). Such a statement can be automatically translated into tree automaton that solves the problem in linear time over the class of partial k-trees (see Section 2.8).

Definition 4.0.2. The class of partial k-trees is denoted by $\mathcal{G}_k$.

So if Π is a CMS-definable decision problem, then there exists a linear-time membership test (*i.e.* a tree automaton) to decide whether a given graph (the instance) belongs to $\Pi \cap \mathcal{G}_k$. Tree automata are a generalization of the conventional finite-state automata which recognize *regular* sets; and the collection of regular sets has a number of closure properties [HU79]. Similar closure properties also exist for the collection of CMS-definable problems:

It is easy to see that this collection is closed under set-theoretic union, intersection and complementation: *i.e.* if Π and Π' are defined by CMS statements Φ and Φ', then

- $\Pi \cup \Pi'$ is CMS-definable, because a graph G belongs to this class iff $G \models \Phi \vee \Phi'$; and
- $\Pi \cap \Pi'$ is CMS-definable, because a graph G belongs to this class iff $G \models \Phi \wedge \Phi'$; and
- $\{G \mid G \notin \Pi\}$ is CMS-definable, because a graph G belongs to this class iff $G \models \neg\Phi$.

The class $\{G|G \notin \Pi\}$ is the set-theoretic complement of Π. This chapter is mainly concerned with the graph-theoretic complement of a decision problem.

Definition 4.0.3. Suppose Π is a decision problem (*i.e.* a class of graphs). The *complement-problem* of Π is the class $\overline{\Pi} = \{\overline{G} \mid G \in \Pi\}$.

This conflicting terminology is perhaps unfortunate, but the term "complement" is used pervasively in both set-theoretic and graph-theoretic settings. We shall not concern ourselves further with problems of the form $\{G \mid G \notin \Pi\}$; and the term "complement-problem" shall be used only in the sense of Definition 4.0.3. If we wish to stress this fact, we will write "graph-theoretic complement-problem". If the meaning is clear from the context, we will write simply "complement" to mean either a complement-graph or a (graph-theoretic) complement-problem.

Later in this chapter, we will see that the collection of CMS-definable problems is *not* closed under graph-theoretic complementation. If fact, we will give a negative answer to the following weaker notion of "closure" under complementation:

Question 4.0.4. *Suppose Π is a CMS-definable decision problem. Is $\overline{\Pi} \cap \mathcal{G}_k$ CMS-definable?*

Should we weaken this question even further by requiring Π itself be a subclass of $\mathcal{G}_k$, then we could always give an affirmative answer: In this case, $\overline{\Pi}$ would be a subclass of $\overline{\mathcal{G}_k}$ (the class of partial k-tree complements). Hence, each yes-instance of $\overline{\Pi} \cap \mathcal{G}_k$ would belong to the class $\mathcal{G}_k \cap \overline{\mathcal{G}_k}$. The number of graphs in $\mathcal{G}_k \cap \overline{\mathcal{G}_k}$ is dependent only on k; and each such graph has only $O(k)$ vertices. A CMS statement, then, could explicitly encode the structure of each yes-instance of $\overline{\Pi} \cap \mathcal{G}_k$.

In Section 2.5, we mentioned the work of Robertson and Seymour [RS], showing that the class $\mathcal{G}_k$ can be characterized by a finite number (dependent only on k) of forbidden minors. Since a CMS statement can encode whether or not the evaluation graph has a particular minor, it follows that the class $\mathcal{G}_k$ is CMS-definable. So, under Conjecture 3.6.3, a decision problem Π can be solved over $\mathcal{G}_k$ in linear time by a tree automaton iff $\Pi \cap \mathcal{G}_k$ is CMS-definable. Such a tree automaton is automatically derived from a CMS statement defining Π. Sometimes, Π is not CMS-definable, but $\Pi \cap \mathcal{G}_k$ can be defined by a CMS statement that takes advantage of the structure of partial k-trees—we will give several examples of this later in the chapter.

We will consider several examples where a problem Π can be defined quite easily by a CMS statement Φ; thus $\Pi = \{G \mid G \models \Phi\}$. The complement-problem (Def. 4.0.3) is then $\overline{\Pi} = \{G \mid \overline{G} \models \Phi\}$. To define $\overline{\Pi}$, therefore, a CMS statement (say Φ') must encode whether or not Φ would evaluate to true on $\overline{G}$, given an evaluation graph G. Such a statement Φ' is to be evaluated over the universe $V(G) \cup E(G)$, so it must be written without making reference to any edge of $\overline{G}$. Thus the complement-problem becomes defined $\overline{\Pi} = \{G \mid G \models \Phi'\}$. In some cases, such a statement Φ' can be written by making only small modifications to the statement Φ that defines Π. For other problems, however, it is difficult (or even impossible) to find such a statement Φ'. We discuss this in more detail in Section 4.1.

The rest of this chapter is organized as follows: In Section 4.1 we introduce the complement of CMS logic—which is defined over the universe $V(G) \cup E(\overline{G})$ for an evaluation graph G. This will be useful for defining $\overline{\Pi} \cap \mathcal{G}_k$ in certain cases where Π is CMS-definable. In Section 4.2 we consider two pairs of well-known decision problems—the complements of one another—to illustrate these ideas. Sections 4.3 to 4.5 deal with χ_t-COLORING—which is the problem of partitioning the vertices of a graph into independent sets of cardinality at most t (the results are also available in [KGS95a]). Sections 4.6 to 4.9 deal with the complement of the f-FACTOR problem—where a graph G is a yes-instance iff there exists a factor of $\overline{G}$ in which each vertex has degree f (the results are also available in [KGS95b]). For fixed $t, f \in \mathbb{N}$, we show that χ_t-COLORING and the complement of f-FACTOR are CMS-definable over $\mathcal{G}_k$. These are the complements of problems that are CMS-definable over all graphs; hence Question 4.0.4 is answered affirmatively in these cases. In Section 4.10, however, we will give a negative answer to that question: A yes-instance of PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS is a graph whose vertex set can be partitioned into sets that each induce a subgraph isomorphic to some *pattern* graph H. The complement of this problem is simply the version of PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS in which $\overline{H}$ is the pattern graph. For any *connected* pattern graph, the problem is CMS-definable over all graphs; but for some disconnected graphs, the problem is not CMS-definable over partial

k-trees. Since any graph H is connected if its complement-graph $\overline{H}$ is disconnected, this provides a negative answer to Question 4.0.4.

4.1 Complement CMS

For a CMS-definable problem Π, we are interested (Question 4.0.4) in whether $\overline{\Pi} \cap \mathcal{G}_k$ is CMS-definable (where $\overline{\mathcal{G}_k}$ is the class of partial k-tree complements). By Definition 4.0.3, a graph G belongs to $\overline{\Pi} \cap \mathcal{G}_k$ iff its complement graph $\overline{G}$ belongs to $\Pi \cap \overline{\mathcal{G}_k}$. The problem Π may be more "natural" than the problem Π; hence, it may be more convenient to consider the question of whether $\Pi \cap \overline{\mathcal{G}_k}$ can be defined in the *complement CMS* (or CCMS) logic. This is actually equivalent to defining $\overline{\Pi} \cap \mathcal{G}_k$ in CMS logic, but it may allow us the view the problem in a more intuitive way.

Definition 4.1.1. The *Complement CMS* (or CCMS) logic is identical to the CMS logic, with the following exceptions:

- For an evaluation graph G, the universe is $V(G) \cup E(\overline{G})$, instead of $V(G) \cup E(G)$.
- Instead of the *Edge* predicate, there is a *Nonedge*(e, v) predicate—which holds iff $\{v, u\} \in E(\overline{G})$ for some $u \in V(G)$.
- An individual variable represents an element of $V(G)$ or $E(\overline{G})$; a set variable represents a subset of $V(G)$ or $E(\overline{G})$.

Equation (2.7.5) gives a CMS predicate "$\sim$" encoding the edge relation of the evaluation graph. This relation can also be encoded by a CCMS predicate:

$$u \sim v \equiv \neg(u = v) \wedge \neg(\exists e)(\mathit{Nonedge}(e, u) \wedge \mathit{Nonedge}(e, v)) \tag{4.1.2}$$

The CCMS predicate $u \sim v$ indicates that $\{u, v\}$ does not belong to the universe $E(\overline{G})$ of non-edges. Any CMS statement that is written without using edge or edge-set variables (but possibly using the "$\sim$" predicate as a "macro") can, therefore, be interpreted as a CCMS statement.

Lemma 4.1.3. *Suppose* Π *is a decision problem. If* $\Pi \cap \overline{\mathcal{G}_k}$ *is CCMS-definable, then* $\overline{\Pi} \cap \mathcal{G}_k$ *is CMS-definable.*

Proof. Suppose Φ is a CCMS statement defining $\Pi \cap \overline{\mathcal{G}_k}$:

$$G \models \Phi \quad \equiv \quad G \in \Pi \cap \overline{\mathcal{G}_k}$$

By Definition 4.0.3, $G \in \Pi \cap \overline{\mathcal{G}_k}$ iff $\overline{G} \in \overline{\Pi} \cap \mathcal{G}_k$. Therefore, $\overline{\Pi} \cap \mathcal{G}_k$ is defined by the CMS statement obtained from Φ by replacing each occurrence of "*Nonedge*(e, v)" with "*Edge*(e, v)", and interpreting this over the universe $V(G) \cup E(G)$, rather than the universe $V(G) \cup E(\overline{G})$, for any evaluation graph G. □

Corollary 4.1.4. *If a decision problem* Π *is CCMS-definable over partial* k*-tree complements (on* n *vertices), then there is an* $O(n)$*-time algorithm to solve* Π *for partial* k*-tree complements, and to solve* $\overline{\Pi}$ *for partial* k*-trees.*

Proof. If Π is CCMS-definable over partial k-tree complements, then (by Lemma 4.1.3) $\overline{\Pi} \cap \mathcal{G}_k$ is CMS-definable; hence (by Theorem 2.8.3), there is a tree automaton that recognizes $\overline{\Pi} \cap \mathcal{G}_k$. This tree automaton is a linear-time algorithm to decide if a partial k-tree is a yes-instance of $\overline{\Pi}$. This algorithm can decide if the complement G of a partial k-tree is a yes-instance of Π, by letting the partial k-tree $\overline{G}$ be the input. To obtain the $O(n)$ time bound, we assume that G is given by representing the $O(n)$ edges of $\overline{G}$, rather than explicitly representing the $\Theta(n^2)$ edges of G. □

4.2 For Example: Cliques and Independent Sets

In this section we consider four well-known graph problems—two pairs of problems that are complements of one another. We adopt the numbering scheme of Garey and Johnson [GJ79] for these problems.

Problem GT19. CLIQUE: For $t \in \mathbb{Z}^+$, a graph G is a yes-instance iff there is a subset V' of $V(G)$ with cardinality $|V'| = t$ such that each pair of vertices in V' are adjacent.

Problem GT20. INDEPENDENT SET: For $t \in \mathbb{Z}^+$, a graph G is a yes-instance iff there is a subset V' of $V(G)$ with cardinality $|V'| = t$ such that no pair of vertices in V' are adjacent.

CLIQUE can be defined for any graph $G = (V, E)$ as follows:

$$(\exists V')(\ (V' \subseteq V)\ \wedge\ (\forall u, v \in V')((u = v) \vee (u \sim v))\ \wedge\ \text{"}|V'| \geq t\text{"}\ ) \tag{4.2.1}$$

For any fixed $t \in \mathbb{N}$, the expression "$|V'| \geq t$" can be translated into either CMS or CCMS logic—by providing t individual variables, and encoding that they are pairwise distinct elements of V'. Furthermore, "$u \sim v$" is either a CMS predicate (2.7.5) or a CCMS predicate (4.1.2). So the above statement (4.2.1) can be interpreted as either a CMS statement or a CCMS statement defining CLIQUE. It follows from Lemma 4.1.3 that INDEPENDENT SET (which is the complement-problem of CLIQUE) is CMS-definable over $\mathcal{G}_k$. In fact, a CMS statement defining the latter problem over all graphs is obtained from statement (4.2.1) by simply replacing "$u \sim v$" with "$\neg(u \sim v)$".

Arnborg *et al.* [ALS91] discuss the idea of leaving some of the variables unquantified in a CMS formula, thus obtaining a "modified" tree automaton to maximize or minimize any polynomial function of the cardinalities of those variables. Courcelle and Mosbah [CM93] show how more general optimization functions can be implemented in this way. This idea provides linear-time algorithms over partial k-trees for the optimization problems corresponding to many CMS-definable decision problems. For example, the size of a maximum clique in a graph $G = (V, E)$ is encoded by the following variation of statement (4.2.1)

$$\max|V'| : \ (\ (V' \subseteq V) \wedge (\forall u, v \in V')((u = v) \vee (u \sim v))\ ) \qquad (4.2.2)$$

By replacing the subexpression "$u \sim v$" with "$\neg(u \sim v)$", we can also encode the size of the largest independent set in this way. The important feature of formula (4.2.2) is that it does not refer to the parameter t fixed in the description of the CLIQUE problem. The formula can be evaluated by a "modified" tree automaton over any partial k-tree, thus computing the size of a maximum clique in linear time.

We now shift our attention to the following pair of problems:

Problem GT15. PARTITION INTO CLIQUES: For $r \in \mathbb{Z}^+$, a graph G is a yes-instance iff $V(G)$ can be partitioned into r sets that each induce a clique in G.

Problem GT4. CHROMATIC NUMBER: For $r \in \mathbb{Z}^+$, a graph G is a yes-instance iff $V(G)$ can be partitioned into r independent sets.

Example 4.2.3. PARTITION INTO CLIQUES is defined for any graph $G = (V, E)$ by the following CMS statement:

$$\begin{aligned}(\exists V', E')(\ \ & (E' \subseteq E) \ \wedge \ \text{"each component of } (V, E') \text{ is a clique"} \ \wedge \\ & \text{"}V' \text{ contains exactly one vertex of each such component"} \wedge \\ & \text{"}|V'| \leq r\text{"}\ )\end{aligned}$$

If V' is left unquantified in the above statement, then a "modified" tree automaton is obtained to minimize $|V'|$, thus determining the minimum number of cliques required to partition the vertices of a partial k-tree. The above statement, however, cannot be interpreted in CCMS logic; so Lemma 4.1.3 is not helpful to obtain a CMS statement defining the complement-problem, CHROMATIC NUMBER. However, a CMS formula can be written to optimize CHROMATIC NUMBER for a partial k-tree $G = (V, E)$ as follows:

$$\begin{aligned}\min |V'| : (\exists V_1, V_2, \ldots V_{k+1})(\; & \text{"}\{V_1, V_2, \ldots V_{k+1}\} \text{ is a partition of } V(G)\text{"} \wedge \\ & \text{"each } V_i \text{ is an independent set"} \wedge \\ & (\forall v)((v \in V_i) \Rightarrow \text{"}|V' \cap V_i| = 1\text{"}) \;) \end{aligned} \tag{4.2.4}$$

The validity of this formula is a consequence of the fact that the vertices of a partial k-tree G can always be partitioned into $k+1$ independent sets. These independent sets can, furthermore, be chosen so that a forest is induced in G by the union of any pair of them—we will make use of this fact later in this chapter.

Proposition 4.2.5. *If G is a partial k-tree, then $V(G)$ can be partitioned into $k+1$ independent sets such that the subgraph of G induced by the union of any $\ell+1$ of these sets is a partial ℓ-tree (for $0 \le \ell \le k$). Such a collection of independent sets is called a* standard partition *of $V(G)$.*

Proof. Suppose G is a partial k-tree; and let $(T, \mathcal{X})$ be a width-k tree decomposition of G. By Fact 2.6.2, $V(G)$ can be partitioned into $k+1$ sets $V_1, V_2, \ldots V_{k+1}$ such that no bag in $\mathcal{X}$ contains more than one vertex of any set V_i $(1 \le i \le k+1)$. It follows from property **T2** (Def. 2.6.1) that each set V_i is independent. Let V' be the union of any $\ell+1$ of these independent sets. Hence, $(T, \mathcal{X}')$ is a tree decomposition of $G_{[V']}$, where $\mathcal{X}' = \{X_a \cap V' | X_a \in \mathcal{X}\}$. Each bag of $\mathcal{X}'$ has at most $\ell+1$ vertices. Therefore, $G_{[V']}$ is a partial ℓ-tree. □

Corollary 4.2.6. *If $\{I_1, I_2, \ldots I_{k+1}\}$ is a standard partition of the vertex set of a partial k-tree G, then the subgraph of G induced by $I_i \cup I_j$ (for $1 \le i, j \le k+1$) is acyclic.*

Proof. By Proposition 4.2.5, the subgraph is a partial 1-tree, which is a forest. □

4.3 The χ_t-COLORING Problem

A *χ_t-coloring* of a graph G is a partition of $V(G)$ into independent sets (or *color classes*) with cardinality at most t. This is said to be an *optimal* χ_t-coloring if the minimum number (denoted by $\chi_t(G)$) of color classes are used. Using this notation, the chromatic number of G can be written $\chi(G) = \chi_n(G)$, where $|V(G)| = n$. We will only be concerned with χ_t-colorings for fixed (constant) t. The problem of finding an optimal χ_t-coloring is known to have polynomial time complexity for several classes of graphs, including cographs [BJ93], bipartite graphs [BJ93], and split graphs [Lon91]. For interval graphs, the problem is NP-hard when $t \geq 4$ [BJ93]; and its complexity remains open when $t = 3$. We will use CMS logic to encode the decision version of this problem (for fixed $t \in \mathbb{N}$) over partial k-trees, thereby proving the problem has linear time complexity for partial k-trees.

The decision problem, χ_t-COLORING, can be defined by a CMS statement similar to the statement (4.2.4) defining CHROMATIC NUMBER (Problem GT4): We need only add the requirement that each independent set contains at most t vertices. For the reasons explained in Section 4.2, however, we wish to define χ_t-COLORING with a CMS statement that does not depend upon the number of color classes in a solution. Thus we wish not to represent each color class by a distinct set variable. In Sections 4.4 and 4.5 we will show that χ_t-COLORING can be defined over partial k-trees in such a way.

A χ_2-coloring of a graph G is a *matching* in the complement graph $\overline{G}$: that is, a set of edges of $\overline{G}$ with pairwise disjoint endpoints. Such a matching is said to be *perfect* if it consists of $\frac{n}{2}$ edges, where $n = |V(G)| = |V(\overline{G})|$ is even. By analogy, we say that a χ_t-coloring of G is *perfect* if it uses exactly $\frac{n}{t}$ color classes, where $n = |V(G)| \equiv 0 \pmod{t}$.

Problem 4.3.1. PERFECT χ_t-COLORING: *A graph G is a yes-instance iff $V(G)$ can be partitioned into some number of independent sets, each with cardinality t.*

Since a maximal matching in any graph can be found in polynomial time (see *e.g.* Berge [Ber76]), it follows that PERFECT χ_2-COLORING has polynomial time complexity over all graphs. For $t \geq 3$, however, PERFECT χ_t-COLORING is NP-complete: This result is obtained by restricting the problem PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS—in which a graph G is a yes-instance iff $V(G)$ can be partitioned into sets that each induce a subgraph of G isomorphic to some fixed *pattern* graph. Kirkpatrick and Hell [KH78] have shown this problem to be NP-complete for any pattern graph with three or more vertices; and the PERFECT χ_t-COLORING problem is equivalent to this problem with a pattern graph consisting of t isolated vertices.

PERFECT χ_t-COLORING is the complement-problem of PARTITION INTO t-CLIQUES—in which a yes-instance is a graph G whose vertex set can be partitioned into $\frac{|V(G)|}{t}$ sets, each of cardinality t, and each inducing a clique in G. The latter problem can be defined in CMS logic by modifying Example 4.2.3 as follows:

$$\begin{aligned}(\exists V', E')(\quad & (E' \subseteq E) \ \wedge \ \text{“each component of } (V, E') \text{ is a clique”} \ \wedge \\ & \text{“each component of } (V, E') \text{ has exactly } t \text{ vertices”})\end{aligned}$$

Now, Question 4.0.4 asks whether PERFECT χ_t-COLORING can be defined in CMS logic over partial k-trees. We will show in Section 4.4 that this can be done; and in Section 4.5 we will use that result to show that CMS statements can also capture the optimization version of χ_t-COLORING over partial k-trees.

4.4 Encoding PERFECT χ_t-COLORING over Partial k-Trees

In this section we develop a CMS statement to encode Problem 4.3.1, PERFECT χ_t-COLORING, over the class of partial k-trees. This problem can be stated formally as follows:

$$\begin{aligned}\exists C_1, C_2, \ldots, C_r : \quad & \text{“}\{C_i\}_{i=1}^{r} \text{ is a partition of } V(G)\text{”} \ \wedge \\ & \textstyle\bigwedge_{i=1}^{r}(\text{ “}C_i \text{ is independent”} \ \wedge \ \text{“}|C_i| = t\text{” })\end{aligned} \qquad (4.4.1)$$

This length of this statement depends upon the size of G: that is, $r = \frac{n}{t}$. We will show, however, that if a partial k-tree G satisfies this statement, then the color classes $C_1, C_2, \ldots, C_r$ can be chosen in such a way that they can be grouped together into a constant number of larger independent sets: Thus, the following CMS statement is satisfied iff statement (4.4.1) is satisfied (provided G is a big enough partial k-tree):

$$\begin{aligned}\exists D_1, D_2, \ldots, D_{2k+1} : \quad & \text{“}\{D_i\}_{i=1}^{2k+1} \text{ is a partition of } V(G)\text{”} \ \wedge \\ & \textstyle\bigwedge_{i=1}^{2k+1}(\text{ “}D_i \text{ is independent”} \ \wedge \ \mathbf{card}_{0,t}(D_i))\end{aligned} \qquad (4.4.2)$$

It is clear that the former statement (4.4.1) is satisfied if the latter statement (4.4.2) is satisfied: An assignment of $D_1, D_2, \ldots, D_{2k+1}$ can be converted into an assignment of $C_1, C_2, \ldots, C_r$ by breaking each D_i into $\frac{|D_i|}{t}$ sets of t vertices (this is possible because $|D_i| \equiv 0 \pmod{t}$). We now state Lemma 4.4.3, which shows the converse (provided G is big enough).

Preconditions

- $|V(G)| \geq 3(k+1)t$
- $|V(G)| \equiv 0 \pmod t$
- $\mathcal{I} = \{I_1, I_2, \ldots I_{k+1}\}$ is a standard partition of $V(G)$, with $|I_i| \geq t$ $(1 \leq i \leq k+1)$

```
1. For $\ell = 1, 2, \ldots, k$:
    (a) Fix $i$ such that $I_i$ contains the minimum nonzero number of uncolored vertices.
    (b) Fix $j \neq i$ such that $I_j$ contains the maximum number of uncolored vertices.
    (c) Create 2 sets, $D_{2\ell-1}$ and $D_{2\ell}$ (each of cardinality $0 \pmod t$), using all of the
        uncolored vertices of $I_i$, and at most $t-1$ of the vertices of $I_j$.
2. Assign the remaining uncolored vertices to $D_{2k+1}$.
```

Figure 4.1: Algorithm to construct a perfect χ_t-coloring of a partial k-tree G

Lemma 4.4.3. *If statement (4.4.1) is satisfied for a partial k-tree G, then CMS statement (4.4.2) is also satisfied, provided* $|V(G)| \geq (5k+1)t$.

We will prove Lemma 4.4.3 first under a simplifying restriction (Lemma 4.4.5). Following this, we will give the technical details needed to obtain the general result. The proof is constructive: We show how a standard partition (Prop. 4.2.5) of $V(G)$ can be converted into an assignment of the sets $D_1, D_2, \ldots, D_{2k+1}$ to satisfy CMS statement (4.4.2). Figure 4.1 presents an algorithm to compute these sets for any partial k-tree G that satisfies several specified conditions. As we discuss this algorithm, we say that a vertex is "colored" after it has been assigned to $D_1, D_2, \ldots$ or D_{2k+1}. Initially, no vertex of G is colored.

Invariant 4.4.4. *Before each iteration of step (1c), I_j contains at least $2t$ uncolored vertices, and I_i contains at least t uncolored vertices.*

Proof. We will show, inductively, that at the start of the ℓ^{th} iteration $(1 \leq \ell \leq k)$:

C1: all vertices are colored in $\ell - 1$ of the independent sets in $\mathcal{I}$, and

C2: the other $k + 2 - \ell$ independent sets each contain at least t uncolored vertices, and

C3: there are (in total) at least $2(k+2-\ell)t + (k+1-\ell)t$ uncolored vertices

The invariant then follows easily, since I_i is one of the independent sets satisfying **C2**; and by the pigeonhole principle (PHP), I_j contains more than $2t$ uncolored vertices.

The base case ($\ell = 1$) follows directly from the stated preconditions. Assume, inductively, that **C1**, **C2** and **C3** are satisfied at the start of the ℓ^{th} iteration ($1 \leq \ell \leq k$). During this iteration, all vertices in I_i become colored (hence **C1** will be satisfied at the start of the next iteration). At most $t-1$ vertices of I_j become colored, leaving at least $2t-(t-1) > t$ uncolored vertices of I_j at the start of the next iteration (hence **C2** will be satisfied). Since I_i contains the minimum nonzero number of uncolored vertices, there are at least

$$\frac{k+1-\ell}{k+2-\ell}(2(k+2-\ell)t + (k+1-\ell)t) = 2(k+1-\ell)t + \frac{(k+1-\ell)^2 t}{k+2-\ell}$$

uncolored vertices that are *not* in I_i (by the PHP). So after coloring $t-1$ (or fewer) vertices of I_j, we are left with at least

$$\begin{aligned} 2(k+1-\ell)t + \tfrac{(k+1-\ell)^2 t}{k+2-\ell} - (t-1) &> 2(k+1-\ell)t + \tfrac{((k+1-\ell)^2-1)t}{(k+1-\ell)+1} \\ &= 2(k+1-\ell)t + ((k+1-\ell)-1)t \\ &= 2(k+2-(\ell+1))t + (k+1-(\ell+1))t \end{aligned}$$

uncolored vertices; hence **C3** will be satisfied at the start of the next iteration. □

Lemma 4.4.5. *Suppose G is a partial k-tree for which $|V(G)| \geq 3(k+1)t$ and $|V(G)| \equiv 0 \pmod t$. If there exists a standard partition of $V(G)$ with at least t vertices in each independent set, then CMS statement (4.4.2) is satisfied.*

Proof. Suppose $\mathcal{I} = \{I_1, I_2, \ldots I_{k+1}\}$ is a standard partition of $V(G)$ such that each I_i ($1 \leq i \leq k+1$) contains at least t vertices. Let $D_1, D_2, \ldots, D_{2k+1}$ be the sets computed by the algorithm of Figure 4.1. Since D_{2k+1} is a subset of some set in $\mathcal{I}$, it is an independent set. We need only show that it is possible to form two independent sets ($D_{2\ell-1}$ and $D_{2\ell}$) in each ℓ^{th} iteration ($1 \leq \ell \leq k$) of step (1c).

Let i, j be the indices chosen in steps (1a) and (1b) of iteration ℓ. Let F be the subgraph of G that is induced by the uncolored vertices of $I_i \cup I_j$. By Proposition 4.2.5, F is a forest. By Invariant 4.4.4, F contains at least $2t$ vertices of I_j and at least t vertices of I_i. Let V' be a set of $t-1$ uncolored vertices of I_j, chosen with maximal degree in F.

Fact. $F \backslash V'$ has at least t distinct trees that each contain at least one vertex of I_i.

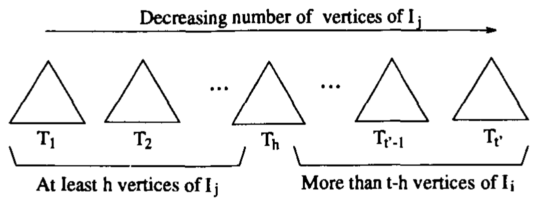
Decreasing number of vertices of I j
T1
T2
Th
Tt'-1
Tt'
At least h vertices of I j
More than t-h vertices of I i

Figure 4.2: The trees of $F \backslash V'$

To verify this fact, note that if $\delta_F(v) \leq 1$ for any $v \in V'$, then no two vertices of I_i will be in the same tree of $F\backslash V'$. Otherwise, $\delta_F(v) \geq 2$ for each $v \in V'$; so by deleting v we increase the number of trees containing a vertex of I_i.

Suppose $F\backslash V'$ has $t' \geq t$ trees; and let them be denoted $T_1, T_2, \ldots, T_{t'}$ such that T_h $(1 \leq h \leq t'-1)$ contains at least as many vertices of I_j as does T_{h+1}. Suppose there are f uncolored vertices of I_i; and let f' be the residue of f (modulo t). We form $D_{2\ell-1}$ with $f-f'$ uncolored vertices of I_i, chosen from the lowest-indexed trees of $F\backslash V'$. We form $D_{2\ell}$ with the remaining f' uncolored vertices of I_i (from the highest-indexed trees) and $t-1$ or fewer vertices of I_j (chosen from the lowest-indexed trees) such that $|D_{2\ell}|$ equals t or 0. To complete the proof, we need only show that $D_{2\ell}$ is an independent set.

Suppose $D_{2\ell}$ is not independent: hence, $D_{2\ell}$ contains a vertex of I_i and a vertex of I_j that belong to the same tree (say T_h) in $F\backslash V'$ (see Figure 4.2). Since the vertices of I_j (I_i) were chosen from the lowest-indexed (highest-indexed) trees, $D_{2\ell}$ contains all vertices of I_j that belong to $T_1, T_2, \ldots, T_{h-1}$ and all vertices of I_i that belong to $T_{h+1}, T_{h+2}, \ldots T_{t'}$. So $D_{2\ell}$ contains at least h vertices of I_j (one from each of $T_1, T_2, \ldots, T_h$). Suppose that some number $h' < h$ of the trees $T_1, T_2, \ldots, T_{h-1}$ each contain a vertex of I_i. Hence (by the "fact" above), at least $t-h'$ of the trees $T_h, T_{h+1}, \ldots T_{t'}$ each contain a vertex of I_i. Therefore, $D_{2\ell}$ contains at least $t-h'$ vertices of I_i, for a total of at least $h+t-h' > t$ vertices (a contradiction). □

It follows from Lemma 4.4.5 that if G is a (large enough) partial k-tree for which t divides $|V(G)|$, then G can fail to have a perfect χ_t-coloring only if every standard partition of $V(G)$ contains at least one small independent set. We now give the general proof of Lemma 4.4.3, without the restriction used in Lemma 4.4.5.

Proof of Lemma 4.4.3. Let G be a partial k-tree with $n \geq (5k+1)t$ vertices; and suppose

$C_1, C_2, \ldots, C_r$ are sets that satisfy formula (4.4.1). Let $\mathcal{I} = \{I_1, I_2, \ldots I_{k+1}\}$ be a standard partition of $V(G)$. Thus, each vertex of G belongs to some set I_i $(1 \le i \le k+1)$ and some set C_j $(1 \le j \le r)$. Without loss of generality, assume each set I_i is nonempty; and let v_i be a vertex in I_i such that the degree $\delta_G(v_i)$ is maximum.

We now wish to assign each vertex to one of the *color classes* $D_1, D_2, \ldots, D_{2k+1}$ to satisfy statement (4.4.2): We say that a vertex is "colored" after it has been assigned to one of these. We begin by coloring $\ell \cdot t$ vertices (where $0 \le \ell \le k$) by letting $\{D_1, D_2, \ldots D_\ell\}$ be a maximal subset of $\{C_j \mid v_i \in C_j;\ 1 \le j \le r;\ 1 \le i \le k+1\}$ such that for each $D_{i'}$ $(1 \le i' \le \ell)$, there is an index i $(1 \le i \le k+1)$ for which the following are true: $v_i \in I_i \cap D_{i'}$; I_i has fewer than t uncolored vertices; and v_i has degree $\delta_G(v_i) \ge 3kt$. Thus, without loss of generality, we assume the following for fixed indices ℓ and m (after possibly renumbering the sets of $\mathcal{I}$):

- For $1 \le i \le \ell$, the following conditions hold:

  C1: $v_i \in D_i \cap I_i$.

  C2: $\delta_G(v_i) \ge 3kt$.

  C3: There are fewer than t uncolored vertices in I_i.

- For $\ell + 1 \le i \le m$, **C3** is satisfied; and if $\delta_G(v_i) \ge 3kt$, then v_i is colored with $D_1, D_2, \ldots$ or D_ℓ.

- For $m + 1 \le i \le k + 1$: **C3** is not satisfied.

Since $|V(G)| \ge (5k+1)t$, some set in $\mathcal{I}$ has more than t uncolored vertices; so $m \le k$. Notice that when each set in $\mathcal{I}$ has sufficiently many vertices, then **C3** is never satisfied (so $\ell = m = 0$) and the proof of Lemma 4.4.5 carries through.

We now consider each set I_i in the order $1, 2, \ldots, k+1$. At the i^{th} step, we assign all uncolored vertices of I_i to one or two new color classes. There are several cases to consider: In cases 1 and 3, we also color at most $t-1$ of the vertices that are not in I_i; in case 2, we color at most $2(t-1)$ of the vertices that are not in I_i. These additional vertices are always chosen from a set of $\mathcal{I}$ that contains enough uncolored vertices so that at least t uncolored vertices remain after the i^{th} step.

Case 1: $1 \le i \le \ell$. Exactly $(\ell + i - 1)t$ vertices are already colored: *i.e.* $D_1 \cup D_2 \cup \ldots \cup D_\ell$, plus t vertices colored in each previous iteration. The sets $I_{i+1}, I_{i+2}, \ldots, I_m$ each satisfy **C3**

and, hence, their union contains at most $(m-i)(t-1)$ uncolored vertices. We have already colored a vertex $v_i \in I_i$ that has at least $3kt$ neighbors. Since $m \geq \ell$ and $1 \leq i \leq m \leq k$, the pigeonhole principle (PHP) shows that v_i has at least

$$\frac{3kt-(\ell+i-1)t-(m-i)(t-1)}{k-m+1} \geq 2t$$

uncolored neighbors in some I_j $(m+1 \leq j \leq k+1)$. By Corollary 4.2.6, each vertex of $I_i - \{v_i\}$ can be adjacent to at most one of those neighbors of v_i. This implies that we can form an color class of size t (or 0) that contains all of the uncolored vertices of I_i and at most $t-1$ of the neighbors of v_i.

Case 2: $\ell+1 \leq i \leq m$. At most $2(i-1)t$ vertices are already colored; and no more than $(m-i+1)(t-1)$ uncolored vertices are in $I_i \cup I_{i+1} \cup \ldots \cup I_m$. Since $1 \leq i \leq m \leq k$, the PHP shows there are at least

$$\frac{(5k+1)t-2(i-1)t-(m-i+1)(t-1)}{k-m+1} > 5t$$

uncolored vertices in some I_j $(m+1 \leq j \leq k+1)$.

Let V' be the set comprised of all uncolored vertices of I_i. If the maximal degree $\delta_G(v_i)$ of a vertex in I_i is $3kt$ or greater, then $v_i \in I_i$ has already been colored; so we can form one color class that is a superset of V', as in Case 1. Otherwise, we choose a vertex $v \in V'$ with the maximum number of neighbors belonging to I_j. Since $\delta_G(v) \leq 3kt-1$, it follows that there are at least

$$\frac{(5k+1)t-2(i-1)t-(m-i+1)(t-1)-(3kt-1)}{k-m+1} > 2t$$

uncolored vertices in some $I_{j'}$ that are *not* adjacent to v (where $m+1 \leq j' \leq k+1$; possibly $j=j'$). Now, we can color the vertices in V' in one of the following two ways. In each case, we also color at most $2(t-1)$ vertices of $I_j \cup I_{j'}$:

Firstly, suppose v has t or more uncolored neighbors belonging to I_j. By Corollary 4.2.6, each vertex of $V'-\{v\}$ is adjacent to at most one of those neighbors; so we can select $t-(|V'|-1)$ of those neighbors that are not adjacent to any vertex of $V'-\{v\}$. Therefore, we can form one color class of size t (or 0) that contains at most $t-1$ of those neighbors, in addition to the vertices of $V'-\{v\}$. We then form a second color class containing v and $t-1$ of the uncolored vertices of $I_{j'}$ that are not adjacent to v.

Otherwise, no vertex of V' has more than $t-1$ uncolored neighbors that belong to I_j. Since I_j contains at least $5t$ uncolored vertices, I_i contains fewer than t uncolored vertices, and $I_i \cup I_j$ induces a forest, we can easily form two color classes whose union contains all the uncolored vertices of I_i, and some of the vertices of I_j.

Case 3: $m+1 \leq i \leq k$. At most $(m+i-1)(t-1)$ vertices in $I_{i+1} \cup I_{i+2} \cup \ldots \cup I_{k+1}$ are already colored: *i.e.* while coloring each of $I_1, I_2, \ldots, I_m$, we colored at most $2(t-1)$ vertices of $I_{i+1} \cup I_{i+2} \cup \ldots \cup I_{k+1}$; and while coloring each of $I_{m+1}, I_{m+2}, \ldots, I_{i-1}$, we colored at most $t-1$ of them. Since $m < i \leq k$, the PHP shows there are at least

$$\frac{(5k+1)t-(m+i-1)(t-1)}{k-m+1} > 2t$$

uncolored vertices in some I_j $(i+1 \leq j \leq k+1)$. I_i contains at least t uncolored vertices, and can be handled (as in the proof of Lemma 4.4.5) using two color classes.

Case 4: $i = k+1$. All uncolored vertices belong to some set of the standard partition; hence, they can be assigned to one color class. □

Theorem 4.4.6. PERFECT χ_t-COLORING *is CMS-definable over the class of partial k-trees as follows:*

$$\begin{aligned} \exists D_1, D_2, \ldots, D_{5k}: \quad & \text{“}\{D_i\}_{i=1}^{5k} \textit{ is a partition of } V(G)\text{”} \wedge \\ & \textstyle\bigwedge_{i=1}^{5k}(\text{ “}D_i \textit{ is an independent set}\text{”} \wedge \mathbf{card}_{0,t}(D_i)) \end{aligned}$$

Proof. It is clear that if the above statement is satisfied for a graph G, then (4.4.1) is also satisfied, since each set D_i can be broken into $\frac{|D_i|}{t}$ sets of t vertices.

Suppose, conversely, that formula (4.4.1) is satisfied for a partial k-tree G: Thus $V(G)$ is partitioned into independent sets $C_1, C_2, \ldots, C_r$, each of cardinality t. If $r \leq 5k$, then the above statement is satisfied by $D_i = C_i$ for $1 \leq i \leq r$, and $D_i = \emptyset$ for $r+1 \leq i \leq 5k$. Otherwise, $|V(G)| \geq (5k+1)t$, and Lemma 4.4.3 shows that only $2k+1$ nonempty sets D_i are required. □

4.5 Optimization of χ_t-COLORING over Partial k-Trees

In this section we are interested in finding the minimum number $\chi_t(G)$ of color classes in a χ_t-coloring of a partial k-tree G. A color class C will be called *full* if $|C| = t$, and will be called *light* if $1 \leq |C| < t$. After augmenting G with enough isolated vertices to fill all color classes, we can use the algorithm for PERFECT χ_t-COLORING (from Section 4.4) as a subroutine to

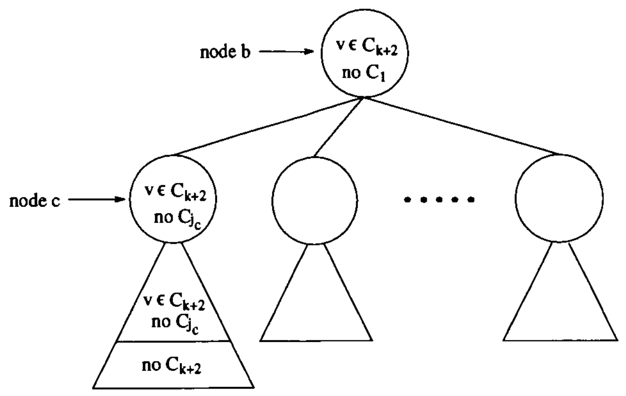

Figure 4.3: Recoloring vertices

solve the optimization problem. A careful analysis of the proof of Lemma 4.4.3 shows that, by removing the requirement that the χ_t-coloring be perfect, the only color classes that need be light are $D_1, D_2 \ldots D_\ell$ (where $0 \le \ell \le k$) and the last color class to be used: *i.e.* there are at most $k+1$ light color classes—provided $|V(G)| \ge (5k+1)t$. In this section we derive this bound directly (regardless of the size of G). It follows that the optimization problem can be solved by $k+1$ calls to the algorithm for PERFECT χ_t-COLORING.

Lemma 4.5.1. *A partial k-tree can be optimally χ_t-colored such that no more than $k+1$ color classes are light.*

Proof. Suppose G is a partial k-tree for which every optimal χ_t-coloring has more than $k+1$ light color classes. Let $C_1, C_2, \ldots, C_r$ be an optimal χ_t-coloring of G, chosen according to the following criteria:

C1: As many color classes as possible are full; and $1 \le |C_i| \le t-1$ for $1 \le i \le k+2$.

C2: $|C_{k+2}|$ is minimized among all χ_t-colorings that satisfy criterion **C1**.

Let G' be the subgraph of G induced by $C_1 \cup C_2 \cup \ldots \cup C_{k+2}$; and let H be any component of G' that contains at least one vertex of C_{k+2}. Let $(T, r, \mathcal{X})$ be a width-k rooted tree decomposition of H; we assume $V_{\text{term}}(H) = \emptyset$, so that r does not have any non-drop vertex (Def. 2.6.5). Now, we may recolor the vertices of H, using only $C_1, C_2, \ldots$ and C_{k+2}, without any regard for the vertices of $G \backslash H$.

Suppose a is a leaf of T. If a vertex (say v) belongs to $X_a \cap C_{k+2}$, then there is a color class C_j $(1 \leq j \leq k+1)$ such that $X_a \cap C_j = \emptyset$. If v is a drop vertex of a, then (by Def. 2.6.1) X_a contains each vertex that is adjacent to v; hence, v can be recolored with C_j, reducing the size of C_{k+2} (contradicting criterion **C2**). This establishes the basis of the following

Inductive Hypothesis. For $a \in V(T)$, X_{T_a} can be partitioned into $C_1, C_2, \ldots, C_{k+2}$ such that:

H1 if $v \in X_{T_a} \cap C_{k+2}$, then v is a non-drop vertex of a, and

H2 there is a color class C_j $(1 \leq j \leq k+1)$ such that no bag in $\mathcal{X}_{T_a}$ contains both a vertex of C_j and a vertex of C_{k+2}.

Suppose that b is an internal node of T, such that each child of b satisfies the inductive hypothesis. By **H1** (inductively), no vertex of $X_{T_b} - X_b$ belongs to C_{k+2}. Since $|X_b| \leq k+1$, there is a color class (without loss of generality, say C_1) such that X_b does not contain both a vertex of C_1 and a vertex of C_{k+2}. For each child c of b (inductively) there is a color class C_{j_c} $(1 \leq j_c \leq k+1)$ such that no bag in $\mathcal{X}_{T_c}$ contains both a vertex of C_{j_c} and a vertex of C_{k+2}. Hence, if either $X_c \cap C_{k+2} = \emptyset$ or $C_{j_c} = C_1$ holds for each child c, then **H2** is satisfied by the node b. If neither of those conditions is satisfied for some child c, then there is some vertex (say v) belonging to $X_b \cap X_c \cap C_{k+2}$; and $X_b \cap C_1 = X_c \cap C_{j_c} = \emptyset$ (see Figure 4.3). It follows that we can swap the color classes C_1 and C_{j_c} for the vertices of $X_{T_c} - X_b$. To see that this is possible, we recolor the vertices one-at-a-time in the following order:

1. For each $u \in C_{j_c} \cap X_{T_c}$, recolor u with C_{k+2}.
2. For each $u \in C_1 \cap X_{T_c}$, recolor u with C_{j_c}.
3. For each $u \in C_{k+2} \cap (X_{T_c} - X_b)$, recolor u with C_1.

We may repeat this process for each child c. If, at any stage of the recoloring, a color class becomes filled, we have contradicted criterion **C1**. After the recoloring, no bag in $\mathcal{X}_{T_a}$ contains both a vertex of C_1 and a vertex of C_{k+2} (hence a satisfies **H2**). Furthermore, no vertex of $X_{T_a} \cap C_{k+2}$ can be a drop vertex of a, since this would contradict **C1**. It follows inductively that **H1** is satisfied by the root of T (which has no non-drop vertex). Therefore, all the vertices of H can be colored using $C_1, C_2, \ldots, C_{k+1}$, without using C_{k+2} (contradicting criterion **C2**). □

It is easy to see that the upper bound of $k+1$ light color classes (given by Lemma 4.5.1) is tight whenever $t > 1$. Consider the clique on $k+1$ vertices: This is a partial k-tree, and no two vertices may share the same color class.

The following corollary follows immediately from Lemma 4.5.1:

Corollary 4.5.2. *If G is a partial k-tree on n vertices, then $\lceil \frac{n}{t} \rceil \leq \chi_t(G) \leq \lceil \frac{n}{t} \rceil + k$.*

So for any partial k-tree G (on n vertices), $\chi_t(G)$ is the minimum value of $\lceil \frac{n}{t} \rceil + \ell$ (over $0 \leq \ell \leq k$) such that G is a yes-instance of a decision problem Π_ℓ, which we now show to be CMS-definable.

Theorem 4.5.3. *For $0 \leq \ell \leq k$, the following decision problem is CMS-definable over partial k-trees:*

$$\Pi_\ell = \left\{ G \mid \chi_t(G) = \left\lceil \frac{n}{t} \right\rceil + \ell \right\}$$

Proof. Let G be a partial k-tree; and suppose $|V(G)| + c \equiv 0 \pmod{t}$, for $0 \leq c \leq t-1$. For $0 \leq \ell \leq k$, let G_ℓ be the graph obtained by adding $c + \ell t$ isolated vertices to G. Hence, $G \in \Pi_\ell$ iff G_ℓ is a yes-instance of PERFECT χ_t-COLORING. By Theorem 4.4.6, this is definable by a CMS statement in which $V(G)$ is partitioned into $5k$ independent sets, each with cardinality divisible by t. The isolated vertices of $G_\ell \backslash G$ are free to belong to any of the independent sets; hence, Π_ℓ is CMS-definable by modifying the statement of Theorem 4.4.6 as follows:

$$\begin{aligned} \exists D_1, D_2, \ldots, D_{5k} : \quad & \text{“}\{D_i\}_{i=1}^{5k} \text{ is a partition of } V(G)\text{”} \wedge \\ & \textstyle\bigwedge_{i=1}^{5k}(\text{ “}D_i \text{ is an independent set”} \wedge \Phi(D_1, D_2, \ldots, D_{5k})) \end{aligned}$$

where Φ encodes the following (using the **card** predicates of CMS logic):

$$k' - \left\lceil \frac{1}{t} \sum_{i=1}^{5k} \|D_i\|_t \right\rceil \leq \ell$$

where k' is the number of color classes D_i $(1 \leq i \leq 5k)$ whose cardinality is not divisible by t; and $\|D_i\|_t$ denotes the residue of $|D_i| \pmod{t}$. □

By Theorem 2.8.3, it can be decided in linear time whether a partial k-tree belongs to each Π_ℓ $(0 \leq \ell \leq k)$. This provides a linear-time algorithm to compute $\chi_t(G)$ for a partial k-tree G.

4.6 f-Factors and their Complements

An *f-factor* of a graph G is a subgraph F of G such that $V(F) = V(G)$ and each vertex v has degree $\delta_F(v) = f$ (for some fixed constant $f \in \mathbb{Z}^+$). A graph G is a yes-instance of the f-FACTOR problem iff G contains an f-factor. The CONNECTED f-FACTOR problem is the variation in which G is a yes-instance iff it contains an f-factor that is connected. The 1-FACTOR problem is simply a reformulation of the PERFECT MATCHING problem—which is the complement of PERFECT χ_2-COLORING (Problem 4.3.1). A yes-instance of the 2-FACTOR problem is a graph whose vertices can be covered with disjoint cycles. The CONNECTED 2-FACTOR problem is also known as the HAMILTONIAN CIRCUIT problem.

f-Factors were first studied by Petersen [Pet91] in 1891. Tutte [Tut52] considered a more general problem, where an instance consists of a graph G and a function $f : V(G) \to \mathbb{N}$. For the general problem, Tutte's *f-Factor Theorem* elegantly characterizes all graphs that contain a factor F with degree $\delta_F(v) = f(v)$ for each vertex v. Such graphs can be recognized in polynomial time, by reducing the problem to recognizing graphs that contain a perfect matching (*i.e.* 1-factor); the details of the reduction can be found in [Bol78]. In the remainder of this chapter, we are only concerned with f-factors for a fixed constant f.

It is much more difficult to recognize graphs with *connected* f-factors. For $f = 1$ the problem is trivial, but for $f \geq 2$ the problem is NP-complete. The HAMILTONIAN CIRCUIT (or CONNECTED 2-FACTOR) problem is one of the classic NP-complete problems [GJ79], and it is easily reducible to the CONNECTED f-FACTOR problem for any $f \geq 3$. There are classes of graphs for which each member contains a Hamiltonian circuit: for example, Dirac [Dir52] showed that every graph with minimum degree $\frac{n}{2}$ contains a Hamiltonian circuit (where n is the number of vertices of the graph). Remarkably, the HAMILTONIAN CIRCUIT problem remains NP-complete over the class of graphs with minimum degree $\delta(n)$, for any $\delta(n) < \frac{n}{2}$ [DHK93]. The complement of a partial k-tree has average degree exceeding $\frac{n}{2}$ (except for some small graphs); however, there is no nontrivial lower bound on the minimum degree, and the Hamiltonian circuit problem is still interesting over this class of graphs.

We have shown (Example 2.7.8) that HAMILTONIAN CIRCUIT is CMS-definable over general graphs. It is not difficult to modify that example to define f-FACTOR (with or without connectedness) for any $f \in \mathbb{N}$. Question 4.0.4 now asks whether the complements of these problems are CMS-definable over partial k-trees. In the following sections we answer this question affirmatively. The complement of f-FACTOR can be expressed as either of the following (equivalent) questions, for an input graph G:

- Does there exist an irreflexive and symmetric relation on $V(G)$ in which every vertex v is paired with exactly f vertices that are not adjacent to v?
- Does $\overline{G}$ contain an f-factor?

Because the latter question is more "natural", our presentation will be more intuitive by considering the f-FACTOR problem over the class $\overline{\mathcal{G}_k}$ of partial k-tree complements. In Section 4.7 we develop a CCMS formula encoding the f-FACTOR problem over $\overline{\mathcal{G}_k}$. It then follows (by Lemma 4.1.3) that the complement problem can be encoded in CMS over the class of partial k-trees. In Sections 4.8 and 4.9 we will use this same approach to show that the complement of CONNECTED f-FACTOR is CMS-definable over partial k-trees.

4.7 Encoding an f-Factor in CCMS Logic

In this section we derive a CCMS statement to define f-FACTOR (for any $f \in \mathbb{N}$) over the class $\overline{\mathcal{G}_k}$ of partial k-tree complements, thus showing that the complement of f-FACTOR is CMS-definable over the class $\mathcal{G}_k$ of partial k-trees. CCMS does not allow us to explicitly represent any edge subset of the evaluation graph; but we can explicitly represent the endpoints of a constant number of edges. The vertices of a graph $G \in \overline{\mathcal{G}_k}$ can be partitioned into $k+1$ cliques; and we will show that an arbitrary f-factor in G can be transformed, by a series of *edge flips*, into a standardized f-factor in which most of the edges are contained within these cliques. Such edges need not be explicitly enumerated by a CCMS statement because their existence is guaranteed whenever the corresponding clique is big enough and a certain parity condition is satisfied.

Throughout this section G is some graph in $\overline{\mathcal{G}_k}$, and $\mathcal{C}$ is a standard partition (Prop. 4.2.5) of $V(G)$. Hence, $\mathcal{C}$ is a collection of $k+1$ sets partitioning the vertices of G such that each set induces a clique in G (or an independent set in $\overline{G}$). We will refer to each set in $\mathcal{C}$ as a *color class.*

Definition 4.7.1. An edge of G is called *monochromatic* if its endpoints belong to the same color class; otherwise it is called *dichromatic.*

Definition 4.7.2. Suppose H is a subgraph of G. Two dichromatic edges are said to be *parallel* in H if

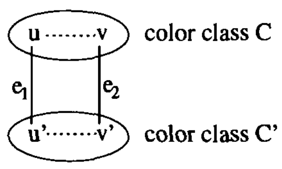

Figure 4.4: Parallel edges in a subgraph of G

- their endpoints belong to the same two (distinct) color classes, and
- all four of the endpoints are distinct, and
- there are no monochromatic edges induced in H by the four endpoints.

Figure 4.4 illustrates a pair of dichromatic parallel edges in a subgraph (say H) of G; these edges are depicted by solid lines. Dashed lines are used to indicate that the like-colored endpoints are not adjacent in H. The like-colored endpoints are, of course, adjacent in G (because each color class induces a clique).

We easily obtain the following proposition:

Proposition 4.7.3. *Let $f \in \mathbb{Z}^+$, and suppose F is an f-factor of G. If e_1 and e_2 are parallel dichromatic edges of F, then the graph obtained by the following sequence of operations is an f-factor of G.*

1. *Add, to F, the two monochromatic edges between the endpoints of e_1 and e_2.*
2. *Delete e_1 and e_2 from the resulting graph.*

We will refer to this sequence of operations as a parallel flip *of e_1 and e_2.*

We wish to transform an arbitrary f-factor of G into an f-factor for which the number of dichromatic edges is bounded by a constant: to do this, we use the following lemma to identify two parallel dichromatic edges which can be flipped. We present a more general result (to identify any constant number of parallel edges) which will be useful in Section 4.9:

Lemma 4.7.4. *Let $f, p \in \mathbb{Z}^+$, and suppose F is an f-factor of G. If F has $2pf^2$ or more dichromatic edges between two fixed color classes, then p of these edges are pairwise parallel.*

Proof. The lemma is trivially true when $p = 1$. We inductively assume it is true for some p, and suppose F contains $2(p+1)f^2$ dichromatic edges between two fixed color classes (say C and C'). Let e be one of these edges: Any other (say e') of these edges is parallel to e unless either e and e' share an endpoint, or their like-colored endpoints are adjacent (in F). Since each vertex has degree f, it follows that F contains at most $2(f-1)^2$ edges between C and C' that are not parallel to e. Therefore, F contains at least $2(p+1)f^2 - 2(f-1)^2 - 1 > 2pf^2$ edges (excluding e) between C and C' that are parallel to e. Among these edges (inductively) there exist p pairwise parallel edges. □

Lemma 4.7.5. *For $f \in \mathbb{Z}^+$, there exists a constant f' (depending only on f and k) such that if G contains an f-factor, then G contains an f-factor with fewer than f' dichromatic edges.*

Proof. If an f-factor of G contains $4f^2$ (or more) dichromatic edges between two fixed color classes, then (by Lemma 4.7.4) two of these edges are parallel. Using Proposition 4.7.3, we can flip pairs of parallel edges until fewer than $4f^2$ dichromatic edges remain between each pair of color classes. Since there are only $\binom{k+1}{2}$ distinct pairs of color classes, the resulting f-factor has fewer than $4f^2\binom{k+1}{2}$ dichromatic edges. □

We have now established that an f-factor with a bounded number of dichromatic edges necessarily exists if any f-factor exists in G. In order to prove the sufficiency of representing only the dichromatic edges (and possibly also a constant number of monochromatic edges) we will need the following lemma—which is a consequence of Tutte's f-Factor Theorem [Tut52]. We give here a constructive proof, which will be helpful in Section 4.9.

Lemma 4.7.6. *Suppose H is a factor of G containing only a constant number of edges, and let f be an upper bound on the degree $\delta_H(v)$ of any vertex v. Suppose further that $V(G)$ is partitioned into $k+1$ color classes, each inducing a clique in G. There exists a constant f'' (depending only on f) such that if a color class (say C) contains f'' or more vertices, and*

$$\sum_{v \in C} (f - \delta_H(v)) \text{ is even,} \tag{4.7.7}$$

then it is possible to replace the monochromatic edges between vertices of C with a larger set of monochromatic edges so that each vertex in C has degree f.

```
1: If two light vertices in $C$ are non-adjacent, then add an edge between them.
2: If $C$ contains only one light vertex (say $v$), then let $u, u' \in C$ be adjacent non-light
   vertices such that neither $u$ nor $u'$ is adjacent to $v$:
   Now, delete the edge $\{u, u'\}$ from $H$, and add two new edges $\{u, v\}$ and $\{u', v\}$.
3: Otherwise, $C$ contains two adjacent light vertices (say $v, v'$). Let $u, u' \in C$ be adjacent
   non-light vertices such that $u$ is not adjacent to $v$, and $u'$ is not adjacent to $v'$:
   Now, delete the edge $\{u, u'\}$ from $H$, and add two new edges $\{u, v\}$ and $\{u', v'\}$.
```

Figure 4.5: Adding monochromatic edges to a factor H
(C is a color class that satisfies the parity condition (4.7.7))

Proof. Suppose that the parity condition (4.7.7) is satisfied for a color class C. Figure 4.5 gives three different ways to modify H, incrementing the number of monochromatic edges with endpoints in C. In the rest of this proof we treat H as a dynamic variable: A vertex $v \in V(H)$ is called *light* as long as $\delta_H(v) < f$. When we say that two vertices of C are *adjacent*, we mean that H currently contains an edge between them.

If some vertex in C has degree less than f, then one of the steps (in Figure 4.5) can be applied to increment the number of edges. In steps **1** and **3**, two distinct light vertices have their degrees incremented by one. In step **2**, there is only one light vertex, and its degree is incremented by two; by the parity constraint (4.7.7), its degree was at most $f - 2$ beforehand. Therefore, no vertex is ever made to exceed degree f. To complete the proof, we need only show that the required vertices $u, u' \in C$ exist when steps **2** and **3** are applied.

Suppose step **2** is to be applied: So C contains only one light vertex, v. Let X be the subset of $C - \{v\}$ comprised of the non-light vertices that are not adjacent to v. Now, $C - X$ contains at most $f - 1$ vertices, each of which is adjacent to fewer than f vertices of X; so there are only a constant number of edges between X and $C - X$. Furthermore, there are only a constant number of dichromatic edges incident to vertices in X. But X contains more than $f'' - f$ vertices, each of which has degree f. Therefore, provided the constant f'' is large enough, there must be a monochromatic edge between two vertices in X: These are the required vertices, u and u'.

The proof for in step **3** is similar: There are two light vertices, v and v'. Let X be the subset of $C - \{v, v'\}$ comprised of the non-light vertices that are adjacent to neither v nor v'. Without loss of generality, we assume step **1** cannot be applied; hence, the light vertices are

$$\begin{array}{lll} \exists & C_j \subseteq V(G) & 1 \leq j \leq k+1 \\ \exists & v_i, v_i' \in V(G) & 1 \leq i \leq f_k \\ \exists & E_j^1, E_j^2, \ldots, E_j^f \subseteq V(G) & 1 \leq j \leq k+1 \end{array}$$

1. $\{C_1, C_2, \ldots, C_{k+1}\}$ is a partition of $V(G)$ into cliques.
2. For $1 \leq i \leq f_k$, either v_i, v_i' are "null", or $v_i \sim v_i'$. No adjacency of G is represented by two different pairs in this way.
3. For $1 \leq j \leq k+1$: if $|C_j| < f''$, then
   (a) $\forall x \in C_j$: x appears exactly f times as an enumerated endpoint, v_i or v_i'
4. For $1 \leq j \leq k+1$: if $|C_j| \geq f''$, then
   (a) $\forall x \in C_j$: x appears at most f times as an enumerated endpoint, v_i or v_i'
   (b) $E_j^1, E_j^2, \ldots E_j^f \subseteq C_j$ and each $x \in C_j$ appears in exactly one of these sets for each appearance of x as an enumerated endpoint
   (c) $f|C_j| - |E_j^1| - |E_j^2| - \ldots - |E_j^f|$ is even

Figure 4.6: CCMS definition of f-FACTOR for the complement G of a partial k-tree (constants f_k and f'' are dependent only on f and k)

pairwise adjacent. Therefore, $C - X$ contains at most f light vertices, and at most $2(f-1)$ non-light vertices. It follows that there is a monochromatic edge between two vertices of X (provided the constant f'' is large enough). □

Theorem 4.7.8. *For $f \in \mathbb{N}$, the f-FACTOR problem is CCMS-encodable over partial k-tree complements.*

Proof. Suppose the complement G of a partial k-tree is a yes-instance of the f-FACTOR problem. We instantiate the variables shown in Figure 4.6 as follows: $C_1, C_2, \ldots, C_{k+1}$ are the color classes comprising a standard partition of $V(G)$. By Lemma 4.7.5, G has an f-factor in which only a constant number of edges are dichromatic. We explicitly enumerate the endpoints v_i, v_i' of each dichromatic edge, and each monochromatic edge in a "small" color class (*i.e.* those with cardinality less than the constant f'' of Lemma 4.7.6); we have used f_k to denote a constant bounding the number of these explicitly enumerated edges. Finally, we let the sets E_j^i $(1 \leq j \leq k+1;\ 1 \leq i \leq f)$ be as specified by item (4b). The statements in the figure are not difficult to encode in CCMS logic, and it is easy to see that they are all satisfied when the variables are as described above.

Suppose, conversely, that the statement of Figure 4.6 is satisfied for the complement G of a partial k-tree. Let H be the factor of G whose edge set contains exactly the enumerated edges $\{v_i, v_i'\}$ $(1 \leq i \leq f_k)$. So, for each "large" clique C_j $(1 \leq j \leq k+1)$, the following quantity is bounded by a constant:

$$d = \sum_{v \in C_j} \delta_H(v) = |E_j^1| + |E_j^2| + \ldots + |E_j^f|$$

Item (4c) states that $f|C_j| - d$ is even: This is exactly the parity condition (4.7.7). Hence, by Lemma 4.7.6, the f-factor can be completed by adding monochromatic edges to the factor H. □

From Theorem 4.7.8 and Lemma 4.1.3, we obtain

Corollary 4.7.9. *For $f \in \mathbb{N}$, the complement of the f-FACTOR problem is CMS-encodable over partial k-trees.*

4.8 Encoding HAMILTONIAN CIRCUIT in CCMS Logic

In this section we derive a CCMS statement to define HAMILTONIAN CIRCUIT over the class $\overline{\mathcal{G}_k}$ of partial k-tree complements. Throughout this section (just as in Section 4.7), G is some graph in $\overline{\mathcal{G}_k}$, and $\mathcal{C}$ is a standard partition of $V(G)$ into $k+1$ color classes. We show that if H is a connected 2-factor of G, then Proposition 4.7.3 can be used to flip pairs of parallel edges so that the number of dichromatic edges is bounded by a constant. We must be careful, though, that this operation does not disconnect the 2-factor.

Definition 4.8.1. Suppose e_1 and e_2 are parallel dichromatic edges in a subgraph H of G. If $H \backslash \{e_1, e_2\}$ has a component containing an endpoint of e_1 and the oppositely-colored endpoint of e_2, then we say that e_1 and e_2 have an *N-configuration* in H.

Suppose, for example, that the parallel edges are $e_1 = \{u, u'\}$ and $e_2 = \{v, v'\}$, as shown in Figure 4.4. These edges have an N-configuration iff $H \backslash \{e_1, e_2\}$ contains either a path between u and v', or a path between u' and v. If H is a Hamiltonian circuit, then $H \backslash \{e_1, e_2\}$ is simply the disjoint union of those two paths.

Lemma 4.8.2. *If a Hamiltonian circuit of G has three dichromatic edges between two fixed color classes, then some pair of them can be flipped to give a (still connected) Hamiltonian circuit.*

Proof. It is readily verified that if three dichromatic edges pass between two fixed color classes of a Hamiltonian circuit, then two of those edges are parallel and have an N-configuration. Furthermore, a connected subgraph of G does not become disconnected by making a parallel flip of edges in an N-configuration. □

Since there are only $\binom{k+1}{2}$ different pairs of color classes, we obtain

Lemma 4.8.3. *If G contains a Hamiltonian circuit, then G contains a Hamiltonian circuit with at most $2\binom{k+1}{2}$ dichromatic edges.*

Theorem 4.8.4. HAMILTONIAN CIRCUIT *is CCMS-definable over the class of partial k-tree complements.*

Proof. A CCMS formula can state that, for some constant $r \leq 6\binom{k+1}{2}$, there exist nonempty vertex subsets $X_1, X_2, \ldots X_r$ for which:

- $X_1, X_2, \ldots X_r$ is a partition of $V(G)$, and
- a clique is induced in G by each of $X_i \cup X_{i+1}$ $(1 \leq i \leq r-1)$ and $X_1 \cup X_r$.

When these conditions are satisfied, a Hamiltonian circuit of G can be obtained by first visiting all of the vertices of X_1 (in any order), then all of the vertices of X_2, and so on.

Conversely, suppose H is a Hamiltonian circuit in G. Without loss of generality (by Lemma 4.8.3), assume that H has no more than $2\binom{k+1}{2}$ dichromatic edges. Let the vertices of G be enumerated $v_1, v_2, \ldots v_n$, such that $E(H)$ consists of $\{v_i, v_{i+1}\}_{i=1}^{n-1}$ and $\{v_n, v_1\}$. The vertices can be packed into $r \leq 6\binom{k+1}{2}$ nonempty sets $X_1, X_2, \ldots, X_r$ to satisfy the conditions itemized above: We treat $[X_1, X_2, \ldots, X_r]$ as a circular sequence; and for each edge of H, we pack its endpoints either into the same set or into consecutive sets, such that a singleton set contains each endpoint of each dichromatic edge, and some set contains all the intervening vertices between any pair of dichromatic edges. Hence, the union of any two consecutive sets is either a monochromatic set or a dichromatic edge; so the conditions itemized above are satisfied. □

Figure 4.7 shows an example of how the first ten vertex sets $X_1, X_2, \ldots X_{10}$ are constructed in the proof of Theorem 4.8.4 when G has $n \geq 10$ vertices. Let $i \geq 3$, and let $i + 4 \leq j << n$, and suppose all but five of the edges of H induced by $\{v_n, v_1, v_2, \ldots, v_{j+3}\}$ are monochromatic: The five dichromatic edges are marked with triangles in Figure 4.7.

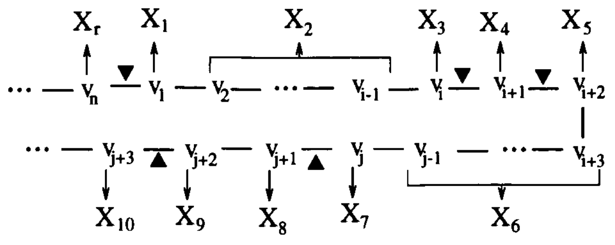

Figure 4.7: Hamiltonian circuit
(Triangles indicate dichromatic edges)

4.9 Encoding a Connected f-Factor in CCMS Logic

We now generalize the result of the previous section, showing that CONNECTED f-FACTOR is CCMS-definable (for any $f \in \mathbb{N}$) over the class $\overline{\mathcal{G}_k}$ of partial k-tree complements. Again, throughout this section G is some graph in $\overline{\mathcal{G}_k}$, and $\mathcal{C}$ is a standard partition of $V(G)$ into $k+1$ color classes. We show that, by a series of edge flips, any connected f-factor in G can be transformed into one in which the number of dichromatic edges is bounded by a constant.

Definition 4.9.1. Suppose F is an f-factor of G. An edge in $E(G) - E(F)$ is said to be a *nonedge* of F. An *alternating circuit* in F is an even-length cycle in G that is composed by alternating edges of F with nonedges of F.

In order to transform the f-factor, we will repeatedly find an alternating circuit, and *flip* its edges with its nonedges. An alternating circuit of a connected f-factor will be called *useful* if this flip operation results in a connected f-factor with fewer dichromatic edges. In Section 4.8 we flipped pairs of parallel edges in an N-configuration: This is one type of useful alternating circuit. In this section we also need other types.

Lemma 4.9.2. *Let $f \geq 2$; and suppose F is a connected f-factor of G. If F contains* $\max\{4, \lceil \frac{k+1}{f-1} \rceil\}$ *or more pairwise parallel edges between two fixed color classes, then there exists a useful alternating circuit in F.*

Proof. Suppose F contains $r \geq \max\{4, \lceil \frac{k+1}{f-1} \rceil\}$ pairwise parallel edges between two fixed color classes, say C and C'. Let $E' = \{e_1, e_2, \ldots, e_r\}$ be the set of these parallel edges; for

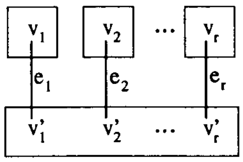

Figure 4.8: Connected f-factor without N-configuration
(Rectangles represent components after deleting the edges $e_1, e_2, \ldots e_r$)

$1 \leq i \leq r$, let $v_i \in C$ and $v'_i \in C'$ be the endpoints of e_i. If any pair of edges in E' have an N-configuration, then they are the edges of a useful alternating circuit. So suppose that no two of them have an N-configuration.

Claim. $F \backslash E'$ consists of $r+1$ components. The vertex set of one of these components is a superset of either $\{v_1, v_2, \ldots, v_r\}$ or $\{v'_1, v'_2, \ldots, v'_r\}$.

Suppose the claim is false. Since F is connected and $|E'| = r$, there are at most $r+1$ components of $F \backslash E'$. Since no pair of edges in E' has an N-configuration, no component of $F \backslash E'$ contains the oppositely-colored endpoints of two distinct edges in E'. Hence, $F \backslash E'$ has two distinct components (say H and H'), each containing at least two vertices of $\{v_i, v'_i \,|\, 1 \leq i \leq r\}$. Let i $(1 \leq i \leq r)$ be the largest index such that H and H' belong to the same component of $F \backslash \{e_1, e_2, \ldots, e_{i-1}\}$. Let I and I' be components of $F \backslash \{e_1, e_2, \ldots, e_i\}$ such that $v_i \in V(I)$ and $v'_i \in V(I')$.

Now, for $1 \leq j \leq r$, $I \backslash E'$ does not contain a path between v_i and v'_j (otherwise e_i and e_j would have an N-configuration). Hence, for some $j \neq i$, there is a path in $I \backslash E'$ between v_i and v_j (for otherwise I would not be connected). Similarly, for some $h \neq i$, there is a path in $I' \backslash E'$ between v'_i and v'_j. Since $|E'| \geq 3$, we can assume without loss of generality that $h \neq j$. Therefore, e_j and e_h have an N-configuration—because a path between v_j and v'_h need not pass through any edge of $E - \{e_i\}$. This contradiction establishes the claim.

Without loss of generality, we now assume that $v'_1, v'_2, \ldots$ and v'_r belong to some component of $F \backslash E'$ (see Figure 4.8). Let T_i $(1 \leq i \leq r)$ be a maximal induced subgraph of F such that T_i is a tree containing the vertex v_i, and each vertex of T_i belongs to color class C. We say that a vertex x of $F \backslash T_i$ is a *neighbor* of T_i if there is an edge of F between x and some vertex of T_i. Clearly, each tree T_i has at least $f-1$ neighbors, and no vertex is a neighbor of more than one of them. Therefore, there are at least $\lceil \frac{k+1}{f-1} \rceil (f-1) \geq k+1$

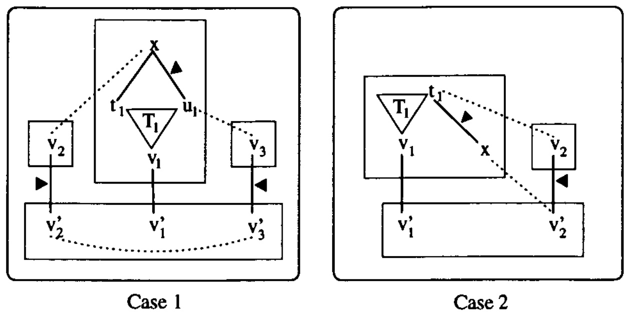

Figure 4.9: Useful alternating circuits
("$*$" = 0 or more edges; "$+$" = 1 or more edges)

neighbors in total. We consider two cases:

Case 1. The color class C contains a vertex (say x) that is a neighbor of one of the trees, say tree T_1. Since T_1 is a *maximal* induced tree whose vertices have color C, it follows that x is adjacent to two distinct vertices (say t_1 and u_1) of T_1. This is illustrated in Figure 4.9, where the tree T_1 is depicted as a triangle surrounded by the vertices t_1, u_1, v_1 (possibly $v_1 = t_1$ or $v_1 = u_1$). Since v_2 and x (respectively, v_3 and u_1) have the same color, but belong to different components of $F \backslash E'$, it follows that they are the endpoints of a non-edge of F (depicted by a dashed line in Figure 4.9). Therefore, the vertices $x, u_1, v_3, v_3', v_2', v_2$ induce an alternating circuit—the edges of which are labeled with filled triangles in Figure 4.9. Two of these edges are dichromatic, and each of the non-edges is monochromatic. So this is a useful alternating circuit.

Case 2. None of the $k + 1$ (or more) neighbors belong to C. By the pigeonhole principle, two of these neighbors (say x and y) belong to the same color class. Since there are at least four trees T_i, we can assume, without loss of generality, that x and y are neighbors of trees other than T_2 and T_3. By Corollary 4.2.6, there exists an edge of G (*i.e.* a non-edge of F) between x (or y) and v_2' (or v_3'). Without loss of generality assume that $\{x, v_2'\}$ is a non-edge of F, and that x is a neighbor of T_1: So let t_1 be a vertex of T_1 such that $\{x, t_1\}$ is an edge of F (see Figure 4.9). Therefore, the vertices x, t_1, v_2, v_2' induce a useful alternating circuit—with two dichromatic edges and (at most) one dichromatic non-edge. □

Combining Lemma 4.7.4 with Lemma 4.9.2, we see that whenever there are enough dichromatic edges in a connected f-factor, there exists a useful alternating circuit whose edges can be flipped to reduce the number of dichromatic edges:

Lemma 4.9.3. *For fixed $f \in \mathbb{N}$, there exists a constant f' such that if G contains a connected f-factor, then G contains a connected f-factor that has fewer than f' dichromatic edges.*

Theorem 4.9.4. *For $f \in \mathbb{N}$, the* CONNECTED f-FACTOR *problem is CCMS-encodable over partial k-tree complements.*

Proof. The problem is trivial to encode for $f = 0$ and $f = 1$. By Theorem 4.8.4, it can be encoded for $f = 2$. To encode it for $f \geq 3$, we augment the CCMS formula of Figure 4.6 with an additional statement to express the following:

$$\forall u, v \in V(G): \text{“the factor contains a path between } u \text{ and } v\text{”}$$

This path can be represented by explicitly enumerating the sequence of vertices $[x_1, x_2, \ldots, x_\ell]$ where $u = x_1$, $v = x_\ell$ and (for $1 \leq i \leq \ell - 1$) either

- x_i and x_{i+1} are endpoints of an explicitly enumerated edge, or
- x_i and x_{i+1} belong to the same “large” color class, and neither is incident with f of the explicitly enumerated edges

In the latter case, we appeal to Lemma 4.7.6 to ensure that the represented f-factor ($f \geq 3$) has a path between x_i and x_{i+1} that consists of only monochromatic edges. Note that connectedness is always preserved by the procedure given (in Figure 4.5) to add monochromatic edges between vertices of “large” color classes. We just need to start with a connected graph among those vertices—which is easy to do when the color class is large enough and $f \geq 3$: □

From Theorem 4.9.4 and Lemma 4.1.3, we obtain

Corollary 4.9.5. *For $f \in \mathbb{N}$, the complement of the* CONNECTED f-FACTOR *problem is CMS-encodable over partial k-trees.*

4.10 Partition Into Isomorphic Subgraphs

As defined by Garey and Johnson [GJ79, Problem GT12], an instance of PARTITION INTO ISOMORPHIC SUBGRAPHS consists of two graphs G and H, with $|V(G)| = r|V(H)|$ for some $r \in \mathbf{Z}^+$. Since this does not fit into our formalism (Def. 4.0.1), we will fix the *pattern* graph H in the problem description:

Problem 4.10.1. PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS*: For a fixed pattern graph H, a graph G is a yes-instance iff $V(G)$ can be partitioned into $\frac{|V(G)|}{|V(H)|}$ sets that each induce a subgraph of G isomorphic to H.*

Kirkpatrick and Hell [KH78] have shown Problem 4.10.1 to be NP-complete for any pattern graph on three or more vertices. This is a generalization of several well-known problems: Probably the most famous is VERTEX MATCHING—for which the pattern graph consists of just two vertices and an edge between them. Another example is PERFECT χ_t-COLORING (Problem 4.3.1)—for which the pattern graph consists of t vertices and no edges.

For a *connected* pattern graph H, PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS is CMS-definable for any graph $G = (V, E)$:

$$(\exists E' \subseteq E)(\text{ “each component of } (V, E') \text{ is isomorphic to } H\text{” }) \qquad (4.10.2)$$

The complement of this problem, then, requires G to be partitioned into copies of $\overline{H}$, which is possibly disconnected. The disconnectedness causes difficulties in obtaining a logical encoding: We now show that, for some disconnected pattern graphs, PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS is not CMS-definable—not even over the class of trees. (The proof carries through even if the subgraphs need not be induced.) Since the complement of a disconnected pattern graph is always connected, this provides a negative answer to Question 4.0.4.

Theorem 4.10.3. *Let P_3 be the path on three vertices; and let S_4 be the star on four vertices. For the pattern graph $P_3 \sqcup S_4$,* PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS *is not CMS-definable over the class of trees.*

Proof. Suppose there is a CMS statment defining the class (say Π) of trees that can be partitioned into copies of $P_3 \sqcup S_4$. For $n, m \geq 2$, let $G_{n,m}$ be the tree (shown in Figure 4.10) on $4n + 3m$ vertices, containing a path with vertex sequence $v_{3n}, v_{3n-1}, \ldots, v_1, u_1, u_2, \ldots, u_{3m}$

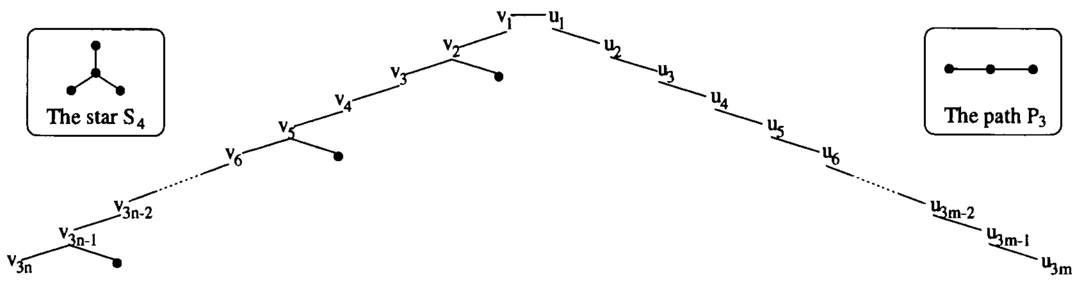

Figure 4.10: The tree $G_{n,m}$

such that, for $i \equiv 2 \pmod 3$, the vertex v_i is adjacent to one other pendant vertex (not counting v_{3n}). So $G_{n,m} \in \Pi$ iff $m = n \geq 2$.

By Theorem 2.8.3, there exists a tree automaton (say $\mathcal{A}$) that recognize the yes-instances of Π. Let $(T, \mathcal{X})$ be a width-1 tree decomposition of $G_{n,n}$; and choose the root r of T such that $X_r = \{v_1, u_1\}$. Say the subtree rooted at some child of the root is a path $P \sqsubseteq T$ on $3n - 1$ nodes, such that the bag indexed by the i^{th} node of P contains u_i and u_{i+1}. By Pumping Lemma 2.9.3 (provided n is large enough), there exist nodes $b, d \in V(P)$ (where b is an ancestor of d) such that the graph $G_{n,n}[T \backslash T_b \cdot (T_b \backslash T_d)^{\ell} \cdot T_d]$ is a yes-instance of Π, for any $\ell \in \mathbb{N}$ (see Definition 2.9.2). But this yes-instance is $G_{n,m}$ for some $m > n$ (a contradiction). □

Corollary 4.10.4. *There exists a CMS-definable decision problem* Π *such that* $\overline{\Pi} \cap \mathcal{G}_k$ *is not CMS-definable, for any* $k \in \mathbb{N}$ *(where* $\mathcal{G}_k$ *is the class of partial* k*-trees).*

Proof. Let H be the complement-graph of $P_3 \sqcup S_4$, where P_3 is the path on three vertices, and S_4 is the star on four vertices. Let Π be the decision problem PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS for the the pattern graph H. Since H is connected, Π can be defined by a CMS statement (4.10.2).

Now, $\overline{\Pi}$ is the decision problem PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS for the pattern graph $\overline{H} = P_3 \sqcup S_4$. Suppose there exists a CMS statement Φ defining $\overline{\Pi} \cap \mathcal{G}_k$. But a CMS-statement Φ' can easily be written such that $G \models \Phi'$ iff G is a tree. Therefore, $\Phi \wedge \Phi'$ defines PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS over the class of trees (contradicting Theorem 4.10.3). □

PERFECT χ_t-COLORING (Problem 4.3.1) provides an example of Problem 4.10.1 where the pattern graph is disconnected (provided $t \geq 2$), and yet it is CMS-definable over partial k-trees. Such a pattern graph consists of t isolated vertices. The CMS-definability can be generalized to any pattern graph consisting of t pairwise isomorphic components.

Theorem 4.10.5. *For any pattern graph consisting of t isomorphic components,* PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS *is CMS-definable over partial k-trees.*

Proof. Suppose the pattern graph H consists of t components, each isomorphic to a fixed graph (say H'). A CMS statement can encode the following for an evaluation graph $G = (V, E)$:

$$\begin{aligned} &\exists V_1, V_2, \ldots, V_{5k}, W_1, W_2, \ldots, W_{5k} : \text{“}\{V_i\}_{i=1}^{5k} \text{ is a partition of } V(G)\text{”} \wedge \\ &\quad \textstyle\bigwedge_{i=1}^{5k}(\text{ “each component of } G_{[V_i]} \text{ is isomorphic to } H'\text{”} \wedge \\ &\qquad \text{“}W_i \text{ contains exactly one vertex from each component of } G_{[V_i]}\text{”} \wedge \\ &\qquad \mathbf{card}_{0,t}(W_i)) \end{aligned}$$

Suppose the above statement is satisfied for a partial k-tree G. So W_i $(1 \leq i \leq 5k)$ contains exactly one vertex from each component of $G_{[V_i]}$; and $|W_i| \equiv 0 \pmod{t}$. Hence, the number of components of $G_{[V_i]}$ is divisible by t, and no pair of these components is adjacent. Therefore, any t of these components comprise an induced subgraph isomorphic to H; and $G_{[V_i]}$ can be subdivided into $\frac{|W_i|}{t}$ such subgraphs.

Conversely, suppose G is a yes-instance of the problem; and let $G' \sqsubseteq G$ such that G' consists of $\frac{|V(G)|}{|V(H)|}$ induced copies of H. Let M be the minor of G' obtained by contracting (Def. 2.5.1) each component of G' into a single vertex. So M has t independent vertices corresponding to each copy of H. Therefore, M is a yes-instance of PERFECT χ_t-COLORING, and the CMS formula given in Theorem 4.4.6 is satisfied for M. It follows easily that the formula above is satisfied for G. □

Although PARTITION INTO FIXED ISOMORPHIC SUBGRAPHS is not CMS-definable over partial k-trees for an arbitrary pattern graph, it is definable in the more powerful *Extended* Monadic Second-order (or EMS) logic [ALS91]. An EMS statement can be automatically translated into a polynomial-time algorithm to solve the corresponding problem over partial k-trees. In their seminal paper, Arnborg *et al.* [ALS91] claimed that a EMS statement could encode this problem independently of the treewidth k; this claim has since been withdrawn [Lag94]. In [GKMS96], we develop an EMS statement to encode the problem (for any fixed pattern graph) over the class of partial k-trees: The length of the EMS statement is dependent upon the treewidth k.

Chapter 5

Conclusions and Open Problems

This thesis 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 some tree automaton. It is 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 remains an open question whether CMS-definability is necessary (for $k \geq 4$). We conjecture that CMS-definability is necessary. In other words, we conjecture that a subclass of the partial k-trees is CMS-definable iff it is recognizable by a tree automaton. We have presented a general strategy which may lead to a proof of this conjecture—we need only establish that CMS logic can encode a tree decomposition of any partial k-tree G. We have shown 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 of a partial k-tree G is to first decompose 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. Encoding a tree decomposition of G is thus reduced to the following two tasks:

1. Determine which vertices belong to each partial k-path (the edge sets can then be determined easily from this information).

2. Encode a path decomposition for each partial k-path (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 $\{v_1, v_2, \ldots, v_{i-1}\}$: There exists some v_j $(1 \leq j \leq i-1)$ in this set such that 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 of G). 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 groups of up to $\min\{\lfloor\frac{k}{2}\rfloor, k - \ell + 1\}$ vertices at each step—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 a 2-connected partial 3-tree; and this can be generalized quite easily to the case of an ℓ-connected partial k-tree where $\ell \geq k-1$. 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. We are not able to conclude, at this time, that recognizability implies CMS-definability for $(k-1)$-connected partial k-trees—because we have not shown how to perform the first task enumerated above for such graphs.

Under the conjecture that CMS-definability equals recognizability of partial k-trees, CMS logic would elegantly characterize the graph decision problems that can be solved in linear time by tree automata over the class $\mathcal{G}_k$ of partial k-trees. In this thesis we investigated how tree automata can also be used to solve problems over the class $\overline{\mathcal{G}_k}$ of partial k-tree complements. The graphs in $\overline{\mathcal{G}_k}$ have $\Theta(n^2)$ edges (where n is the number of vertices), but they can be represented in $O(n)$ space with a list of their "non-edges". The "complement" of CMS logic (called CCMS logic) encodes properties of a graph $G \in \overline{\mathcal{G}_k}$ by referring to the non-edges of G (*i.e.* the edges of $\overline{G} \in \mathcal{G}_k$). A CCMS statement over G, then, can be interpreted as a CMS statement over $\overline{G}$; so a tree automaton can be used to evaluate it in $O(n)$ time. This provides a linear-time algorithm over $\overline{\mathcal{G}_k}$ for any graph problem that can be defined by a (CCMS) logical statement over a universe consisting of the vertices and non-edges of the evaluation graph.

A graph decision problem can be represented by the class (say Π) of yes-instances. A CMS statement defining the class $\Pi \cap \mathcal{G}_k$ is equivalent to a linear-time algorithm for deciding whether a partial k-tree (in $\mathcal{G}_k$) is a yes-instance. Similarly, a CCMS statement defining

$\Pi \cap \overline{\mathcal{G}_k}$ is equivalent to a linear-time algorithm for deciding whether a graph in $\overline{\mathcal{G}_k}$ is a yes-instance. The latter type of algorithm is implemented by a tree automaton that recognizes $\overline{\Pi} \cap \mathcal{G}_k$, where $\overline{\Pi}$ is defined as the class containing the graph-theoretic complement of each graph in Π. By definition, $\overline{\Pi} \cap \mathcal{G}_k$ is CMS-definable iff $\Pi \cap \overline{\mathcal{G}_k}$ is CCMS-definable. In this thesis we addressed the question of whether $\overline{\Pi} \cap \mathcal{G}_k$ is CMS-definable if Π is CMS-definable. We showed that this question is answered affirmatively for some CMS-definable decision problems, but is answered negatively for others. Thus the graph-theoretic complement-problem $\overline{\Pi}$ does not necessarily have linear time complexity over the class of partial k-trees. In other words, it is not necessarily the case that a CMS-definable decision problem has linear time complexity over the class of partial k-tree complements.

Although CMS logic can capture most of the decision problems that are known to have linear time complexity over partial k-trees, there are exceptions. For example, a CMS statement cannot define whether or not a tree has a unique *center*: A center of a graph G is a vertex $v \in V(G)$ such that, over the collection of shortest paths between v and each vertex of G, the length of the longest path is minimized. Using Pumping Lemma 2.9.3, it can be shown that the class (say Π) of trees with a unique center is not CMS-definable. However, there is a linear-time dynamic-programming algorithm to decide whether a tree belongs to Π: This algorithm can be implemented by a tree automaton equipped with a *counter* that allows an integer to be associated with each terminal—representing the maximum length of the shortest path between the terminal and any vertex that has been seen so far. Network flows provide another example where it might be useful to augment tree automata with some sort of counter: Hagerup *et al.* [HKNR95] noted that a minimum-weight cut-set can be encoded by an MS statement (over weighted graphs); thus, the value of a maximum flow (which is equal to the minimum weight of a cut-set) can be computed in linear time over partial k-trees—because this is a *Linear Extended MS Extremum Problem*, as defined by Arnborg *et al.* [ALS91]. Hagerup *et al.* also show how dynamic-programming techniques can be used to compute the flow through each edge (to achieve a a maximum flow). However, this cannot be expressed within the MS logical formalism. It would be interesting, then, to see if CMS logic can be extended somehow so that problems of this sort can be defined.

Bibliography

[ACP87] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

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

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.

[APC90] S. Arnborg, A. Proskurowski, and D.G. Corneil. Forbidden minors characterization of partial 3-trees. *Disc. Math.*, 80:1–19, 1990.

[APS90] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. In *Lecture Notes in Computer Science (Proc. 4th CSL)*, volume 533, pages 1–16. Springer-Verlag, 1990.

[Arn85] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability. *BIT*, 25:2–33, 1985.

[Ber76] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 2nd edition, 1976.

[BJ93] H.L. Bodlaender and K. Jansen. On the complexity of scheduling incompatible jobs with unit-times. In *Lecture Notes in Computer Science (Proc. 18th MFCS)*, volume 711, pages 291–300. Springer-Verlag, 1993.

[BLW87] 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.

[Bod88] H.L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Lecture Notes in Computer Science (Proc.* 15th *ICALP)*, volume 317, pages 105–119. Springer-Verlag, 1988.

[Bod93] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc.* 25th *STOC*, pages 226–234, 1993.

[Bol78] B. Bollobás. *Extremal Graph Theory.* Academic Press, London, 1978.

[BPT92] 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.

[CM93] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109:49–82, 1993.

[Cou90a] B. Courcelle. Graph rewriting: an algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier, Amsterdam, 1990.

[Cou90b] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[Cou91] 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.

[DHK93] E. Dahlhaus, P. Hajnal, and M. Karpinski. On the parallel complexity of Hamiltonian cycle and matching problem on dense graphs. *J. Algorithms*, 15:367–384, 1993.

[Dir52] G.A. Dirac. Some theorems on abstract graphs. *Proc. London Math. Soc. (Ser. 3)*, 2:69–81, 1952.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[GKMS96] A. Gupta, D. Kaller, S. Mahajan, and T. Shermer. Vertex partitioning problems on partial k-trees. In *Lecture Notes in Computer Science (Proc. 5th SWAT)*, volume 1097, pages 161–172. Springer-Verlag, 1996.

[GS84] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[Hal71] R. Halin. Studies on minimally n-connected graphs. In *Combinatorial Mathematics and its Applications*, pages 129–136. Academic Press, London, 1971.

[HKNR95] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizations of k-terminal flow networks and computing network flows in partial k-trees. In *Proc. 6th SODA*, pages 641–649, 1995.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[Kal96] D. Kaller. Definability equals recognizability of partial 3-trees. In *Lecture Notes in Computer Science (Proc. 22nd WG).* Springer-Verlag, 1996.

[KGS95a] D. Kaller, A. Gupta, and T. Shermer. The χ_t-coloring problem. In *Lecture Notes in Computer Science (Proc. 12th STACS)*, volume 900, pages 409–420. Springer-Verlag, 1995.

[KGS95b] D. Kaller, A. Gupta, and T. Shermer. Regular-factors in the complements of partial k-trees. In *Lecture Notes in Computer Science (Proc. 4th WADS)*, volume 955, pages 403–414. Springer-Verlag, 1995.

[KH78] D.G. Kirkpatrick and P. Hell. On the complexity of a generalized matching problem. In *Proc. 10th STOC*, pages 240–245, 1978.

[Lag94] J. Lagergren, October 1994. Personal communication.

[Lon91] Z. Lonc. On the complexity of some chain and antichain problems. In *Lecture Notes in Computer Science (Proc. 17th WG)*, volume 570, pages 97–104. Springer-Verlag, 1991.

[Men27] K. Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

[MP94] S. Mahajan and J.G. Peters. Regularity and locality in k-terminal graphs. *Disc. Appl. Math.*, 54:229–250, 1994.

[Per90] D. Perrin. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 1–57. Elsevier, Amsterdam, 1990.

[Pet91] J. Petersen. Die Theorie der regularen Graphen. *Acta Math.*, 15:193–220, 1891.

[Pro84] A. Proskurowski. Separating subgraphs in k-trees: cables and caterpillars. *Disc. Math.*, 49:275–285, 1984.

[RS] N. Robertson and P.D. Seymour. Graph minors. XX. Wagner's conjecture. To appear.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier, Amsterdam, 1990.

[TNS82] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *JACM*, 29:623–641, 1982.

[Tut52] W.T. Tutte. The factors of graphs. *Can. J. Math.*, 4:314–328, 1952.

[vL90] J. van Leeuwen. Graph algorithms. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 527–631. Elsevier, Amsterdam, 1990.

[WW89] E. Wanke and M. Wiegers. Undecidability of the bandwidth problem on linear graph languages. *Inform. Process. Lett.*, 33:193–197, 1989.