2009-04-27

Graph Decompositions and Monadic Second Order Logic

Jonathan D. Adler
Worcester Polytechnic Institute

GRAPH DECOMPOSITIONS AND MONADIC SECOND ORDER LOGIC

by

Jonathan D. Adler

A Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Applied Mathematics

May 2009

APPROVED:

Dr. William Martin, Advisor

Dr. Daniel Dougherty, Advisor

Abstract

A tree decomposition is a tool which allows for analysis of the underlying tree structure of graphs which are not trees. Given a class of graphs with bounded tree width, many NP-complete problems can be computed in linear time for graphs in the class. Clique width of a graph G is a measure of the number of labels required to construct G using several particular graph operations. For any integer k, both the class of graphs with tree width at most k and the class of graphs with clique width at most k have a decidable monadic second order theory. In this paper we explore some recent results in applying these graph measures and their relation to monadic second order logic.

Acknowledgments

I would like to thank my advisors Professor William Martin and Professor Dan Dougherty for their guidance in writing this paper. I would also like to thank my parents for stressing the importance of a good education, and the residents of 26 Lancaster Street Apartment 3 for keeping me sane while I pursued a good education.

Contents

Chapter 1

Introduction

The concept of tree decompositions is a powerful tool in the field of graph theory. Introduced by Robertson and Seymour in 1984, a tree decomposition of a graph allows for the analysis of the underlying tree like structure of the graph. Many of the dynamic programming techniques used on trees can be used on classes of graphs with bounded tree widths. Originally, tree decompositions were used by Robertson and Seymour to prove the graph minor theorem. In 1988 Courcelle showed that for a fixed k, problems that can be written as logical statements in monadic second order logic could be solved in linear time on graphs with tree width less than or equal to k. Arnborg and Lagergren also proved this independently in 1991. They also showed that the theory of monadic second order logic is decidable for the class of graphs with tree width less than or equal to k. So given a monadic second order logic sentence σ, whether or not σ is true for all graphs of tree width no greater than k is decidable.

The similar concept of clique-width was introduced by Courcelle in 1997. Rather than looking at the underlying tree like structure of the graph, the clique-width of a graph measures the amount of vertex labels needed to construct the graph using a particular set of graph operations. For a fixed integer k, given a monadic second order logic sentence σ and a graph G with clique width less than or equal to k (as well as the sequence of graph operations needed to construct the graph), whether σ true for G can be determined in linear time. And similarly to tree-width, for the class of graphs with clique-width no greater than k the theory of monadic second order logic is decidable. Seese made a conjecture equivalent to the following: if a class of graphs C has decidable monadic second order theory, then C has bounded clique width. The conjecture remains open but it has been found to hold for special cases such as classes of planar graphs.

The paper is organized as follows: Chapter 2 introduces the definitions and main theorems of tree width and clique width. Chapter 3 introduces logic and specifically monadic second order logic which will be the focus of the later chapters. Chapter 4 introduces the notion of an interpretation and the connection between monadic second order logic and binary trees. Chapter 5 proves that the model checking problem can be computed in linear time for classes of graphs with bounded tree width by using interpretations. Chapter 6 discusses Seese's Conjecture and it's connection to clique width. Chapter 7 shows several examples of the techniques discussed in the paper, and Chapter 8 dicusses open problems in the field.

Chapter 2

Tree Decompositions, *k*-Expressions, and Finding Them

2.1 Tree Decompositions

There are many instances of problems which are trivial to solve on the class of trees; for example the 2-coloring problem. The question arises of if we can find classes of graphs which are not trees but have many of the similar underlying properties of trees. This can be done by finding the tree decomposition of a graph, which is defined as follows: Let G be a graph, let T be a tree, and let $\mathcal{V} = (V_t)_{t \in V(T)}$ be a set of vertex sets of G indexed by vertices in T, so $V_t \subseteq V(G)$ for each vertex t of T. The pair $(\mathcal{V}, T)$ is called a *tree decomposition* provided the following conditions hold:

1. For each $v \in V(G)$ there is some t such that $v \in V_t$;
2. For each edge $(v_1, v_2) \in E(G)$ there is some t such that $v_1, v_2 \in V_t$;
3. If $v \in V_{t_1}$ and $v \in V_{t_2}$ then $v \in V_i$ for all vertices i on the path between vertices t_1 and t_2 in T.

An example of a graph and a tree decomposition of the graph is shown in Figure 2.1. Trees have the special property that they have no cycles; removing any edge from the graph separates the graph into two components. This allows for dynamic programming algorithms to be used on trees. Again if we take 2-coloring of the tree we can start by arbitrarily coloring a vertex, we then color the adjacent vertices based only on what the previous vertex has already been colored. We repeat this process until the entire tree is 2-colored. With tree decompositions we would want to

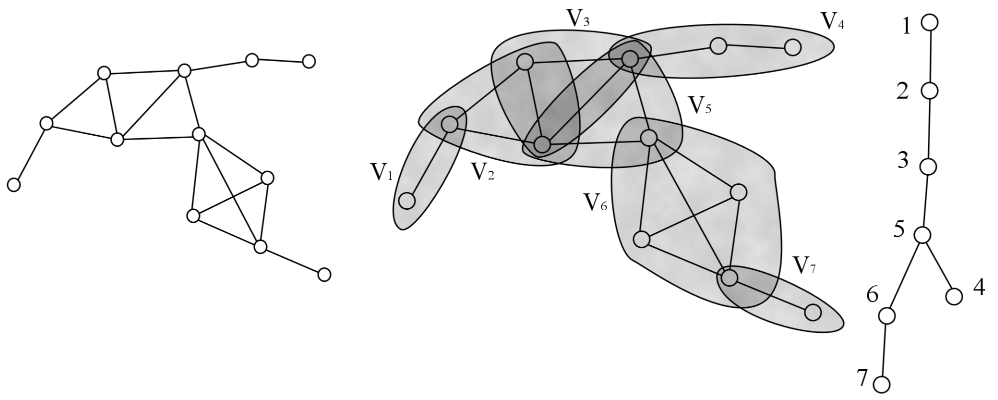


Figure 2.1: A graph and a tree decomposition of the graph.

utilize similar dynamic programming concepts, mainly that to solve a problem for a vertex subset we only have to consider adjacent vertex subsets in the tree of the tree decomposition. The following two theorems illustrate why this is possible:

Theorem 2.1.1. *Let G be a graph and $(\mathcal{V},T)$ a tree decomposition of G. If (t_1,t_2) is an edge in T then let T_1 and T_2 be the components of $T-(t_1,t_2)$ containing t_1 and t_2 respectively. Removing $V_{t_1} \cap V_{t_2}$ from G separates G into two subgraphs that are disconnected from each other: the induced subgraph G_1 with $V(G_1) = \left(\bigcup_{t \in T_1} V_t\right) \setminus (V_{t_1} \cap V_{t_2})$, and the induced subgraph G_2 with $V(G_2) = \left(\bigcup_{t \in T_2} V_t\right) \setminus (V_{t_1} \cap V_{t_2})$.*

Proof. By contradiction. For G_1 and G_2 not to be separated they must share a vertex or there must be an edge connecting the two graphs. Let $a \in V(G)$ be a vertex in both G_1 and G_2 but not in $V_{t_1} \cap V_{t_2}$. Then $a \in V_x$ for some $x \in V(T_1)$ and $a \in V_y$ for some $y \in V(T_2)$. But since T is a tree, the path between x and y contains t_1 and t_2. Thus $a \in V_{t_1} \cap V_{t_2}$; this is impossible. Let b be a vertex in G_1 and c be a vertex in G_2 with (b,c) an edge in G and $b,c \notin V_{t_1} \cap V_{t_2}$. Since (b,c) is an edge there must be a vertex $z \in T$ such that $b,c \in V_z$ by the definition of tree decomposition. Without loss of generality assume $z \in T_1$; since c is in G_2 and therefore is in a vertex set V_y for some y in T_2, c must be in all of the vertex sets V_t on the path between z and y. This path must include t_1 and t_2 so $c \in V_{t_1} \cap V_{t_2}$. □

Theorem 2.1.2. *For any $W \subseteq V(G)$, either there exists a $t \in T$ such that $W \subseteq V_t$ or there are vertices $w_1, w_2 \in W$ and an edge $(t_1,t_2) \in E(T)$ such that the removal of $V_{t_1} \cap V_{t_2}$ from G separates w_1 from w_2.*

Proof. For any given edge (x,y) in T exactly one of the following must occur:

1. there exists w_1 and w_2 in W such that the removal of $V_x \cap V_y$ separates w_1 from w_2;
2. vertex set W falls entirely in $V_x \cap V_y$;
3. vertex set W falls entirely into one of the two components of, say G_x and G_y, of G separated by the removal of $V_x \cap V_y$.

If for any edge in T one of the first two conditions occurs then the theorem holds, so assume that for all edges in T the third condition holds. Orient each edge (x,y) with the direction pointing towards x if $W \subseteq V(G_x)$ and orient the edge with the direction pointing towards y if $W \subseteq V(G_y)$. Once all the edges have an orientation, since T is a tree there must be some vertex z in T with no edges oriented away from it. No element of W can fall in any of the vertex sets labeled by vertices in T adjacent to z since z has no outward edges. Thus for each k adjacent to z let G_z^k be the component of the induced subgraph created by the removal of $V_k \cap V_z$ containing W. It can be seen that $\bigcap_k G_z^k \subseteq V_z$ and thus $W \subseteq V_z$. $\square$

So the removal of any edge of the tree of a tree decomposition separates the original graph. Also any two vertices in the original graph either share a vertex set in the tree decomposition or there exists an edge (x,y) in the tree decomposition such that the removal of $V_x \cap V_y$ separates the two vertices. Thus we have the properties of a tree we want captured in the tree decomposition. However now our dynamic programming techniques will require the analysis of each individual vertex subset of the tree decomposition. For a given graph G there may be many different tree decompositions, in fact there will always be one: the trivial tree decomposition $(\{V(G)\}, K_1)$. Since we need to analyze each vertex subset which may have no special properties, the smaller the cardinality of the vertex subsets the better the tree decomposition.

Definition 2.1.1. The *width* of a tree decomposition $(T, \mathcal{V})$ is $\max\{|V_t| - 1 : t \in T\}$. The *tree width* of a graph G is the minimum width of any possible tree decomposition of G.

The reason for the subtraction of 1 in the definition of width is to allow trees themselves to have a tree width of 1. Tree width can now be considered as a measure of how similar to a tree the graph is; by having a higher tree width the graph has large components which are not similar to trees. Throughout this paper we will be concerned with classes of graphs with bounded tree width, these are classes of graphs where all of the graphs have some underlying tree like structure. This is further seen in the following corollary:

Corollary 2.1.3. *The vertices of any complete induced subgraph must be contained in some member of $\mathcal{V}$ and therefore $twd(G) \geq \alpha(G) - 1$.*

The notion of tree width was introduced by Neil Robertson and Paul Seymour for use in the graph minor theorem. A *quasi-ordering* is a relation that is reflexive and transitive. Given a set X and quasi-ordering $\preceq$, $\preceq$ is a *well-quasi-ordering* if for every infinite sequence $x_0x_1x_2\ldots$ of elements of X there exists an $i < j$ such that $x_i \preceq x_j$. This is equivalent to saying that there does not exist an infinite antichain of elements of X nor a strictly decreasing sequence of elements of X.

Let G be a graph containing vertices x and y which are connected by an edge e. The graph G' created by the *edge contraction* of e is the induced subgraph of G with x, y, and e removed and a new vertex z inserted which is adjacent to exactly the vertices that are adjacent to x and y in G. Graph H is a *minor* of graph G if it can be obtained from G by a combination of edge deletions and edge contractions.

Theorem 2.1.4 (Robertson and Seymour)**.** *The set of finite graphs are well-quasi-ordered under the minor relation.*

This extremely deep theorem was a major breakthrough in the field of graph theory, more so than even the four color theorem. The proof was over 500 pages in total and relied heavily on bounded tree width. For a further analysis see [10].

2.2 Clique Width

The notion of tree width is not the only method of measuring special properties of a graph that allow for dynamic programming techniques. The notion of clique width characterizes graphs based on the number of colors needed to create the graph using certain graph operations, which we define as follows:

Let $\mathcal{C}$ be a countable set of labels. We define the following symbols to denote operations on graphs:

1. A nullary symbol a for every label in C.

2. A unary symbol $\rho_{a\to b}$ for each a, b, pair in C with $a \neq b$. This symbol represents relabeling all vertices labeled a with label b.

3. A unary symbol $\eta_{a,b}$ for each a, b, pair in C with $a \neq b$. This symbol represents adding an edge between each vertex pair in the set
$\{(x,y) |$"x has label a", "y has label b"$\}$.

4. A binary symbol $\oplus$. This symbol represents the disjoint union of two graphs.

Let $C \subseteq \mathcal{C}$. The set $T(C)$ is the smallest set of graphs satisfying the following conditions:

- for any $x_i \in C$, the graph with a single vertex labeled x_i is in $T(C)$;
- for any t_1, t_2 in $T(C)$, the graph $t = t_1 \oplus t_2$ is in $T(C)$;
- for any t_1 in $T(C)$, and any x_i, x_j be in C, $t = \rho_{x_i \to x_j}(t_1)$ is in $T(C)$;
- for any t_1 in $T(C)$, and any x_i, x_j be in C, $t = \eta_{x_i, x_j}(t_1)$ is in $T(C)$.

The *clique-width* of a graph G is defined as:

$$\mathrm{Cwd}(G) := \min\{|C| : G \in T(C)\}.$$

A *k-expression* is a string of symbols from the set
$\{a, b, \ldots, \rho_{a\to b}, \rho_{b\to c}, \ldots, \eta_{a,b}, \eta_{b,c}, \ldots, \oplus\}$ with $\{a, b, c, \ldots\} \in C$ requiring the use of at most k labels, but without any restriction on the length of the string.

With tree width there exist exponential time algorithms that, upon input of a graph G, will output a tree decomposition with width h where h is the tree width of G. Currently, no such algorithms exist for clique width, and the best result is the following:

Theorem 2.2.1 (Courcelle)**.** *[18] For all $k \in N$ there exists an $O(n^9 \log(n))$ algorithm that given a graph of clique-width k outputs a (2^{3k+2})-expression for G, where n is the number of vertices in G.*

Theorem 2.2.2 (Courcelle)**.** *For any graph G, $cwd(G) \leq 2^{twd(G)+1} + 1$.*

It is clear that there is no way to bound tree width as a function of clique width since the class of cliques have clique width bounded by 2 and unbounded tree width. Thus, having bounded tree width is a stronger requirement than having bounded clique width.

2.3 Algorithms for Finding Tree Decompositions and *k*-expressions

Since classes of graphs with bounded tree width and bounded clique width have special properties, it makes sense to look at the time complexity of questions asking for the widths of graphs. The following results have been discovered.

Theorem 2.3.1 (Arnborg)**.** *The following problem is NP-complete:*

INPUT: *A graph G and an integer k*

QUESTION: *Is the tree width of G bounded by k?*

Theorem 2.3.2 (Bolaender)**.** *Fix an integer k. The following problem can be solved in linear time:*

INPUT: *A graph G*

QUESTION: *Is the tree width of G at most k? If so, what is a valid tree decomposition of G with width at most k?*

Note the difference between the two theorems. Given a graph as input and computing its tree width takes exponential time if P $\neq$ NP. However, if we fix a natural number k and ask if the graph has tree width less than k we can find a solution in linear time on the number of vertices and edges in the graph. The distinction here is that in the first theorem the k is part of the input to the question while the question in the second theorem fixes k beforehand. In fact one exponential time algorithm to find a tree decomposition of a graph G would be to run the second algorithm with $k = 1$, $k = 2$, and so on until a valid tree decomposition is found.

Theorem 2.3.3 (Fellows)**.** *The following problem is NP-hard:*

INPUT: *A graph G and an integer k*

QUESTION: *Is the clique width of G bounded by k?*

This algorithm only answers the question of if a k-expression of graph G exists, it does not actually provide a valid k-expression. Currently no algorithm exists to find a valid k-expression for graph G with $cwd(G) = k$ except in the cases where $k < 4$.

Chapter 3

Logic Background

While some problems such as k-coloring may be easier on graphs with bounded tree width or bounded clique width than they are on graphs in general, this may not be true for every problem. It is beneficial to have a way of classifying problems based on their formation as a logical sentence. Thus we need to introduce the notion of graphs as relational structures, and problem as logical sentences about those relational structures.

Definition 3.0.1. A *language* $\mathcal{L}$ is a set of relation symbols and function symbols each with a non-negative integer arity. A *structure* of language $\mathcal{L}$ is an ordered pair $\langle U, R, F\rangle$ where U is a non-empty set called the *universe*, R is a set of relations on U indexed by the relation symbols in $\mathcal{L}$ and with corresponding arity, and F is a set of functions indexed by the function symbols in $\mathcal{L}$ with corresponding arity. For $f \in F$ with arity n, $f : U^n \to U$.

Definition 3.0.2. Let X be a set of variables and let F be a set of functions with $F_0 \subseteq F$ the set of functions of arity 0. The set $T(X)$ of *terms of type F over X* is the smallest set such that:

- $X \cup F_0 \subseteq T(X)$,
- if $p_1, \ldots, p_n \in T(X)$ and f is a function in F with arity n, then $f(p_1, \ldots, p_n) \in T(X)$.

Definition 3.0.3. Let $\mathcal{L} = \langle R, F\rangle$ be a language, let v be the set of *individual* variables, let V be the set of *relation* variables, and let W be the set of *functional variables*. The *atomic formulas of type $\mathcal{L}$* are precisely the expressions of the form:

- an expression $r(t_1, t_2, ..., t_n)$ where r is a relational symbol in $\mathcal{L}$ with arity n and each t_i is a term of type $F \cup W$ over v;

- an expression $t_1 = t_2$ where t_1 and t_2 are terms of type $F \cup W$ over v;
- an expression $T_1 = T_2$ where $T_1, T_2 \in V \cup W \cup \mathcal{L}$;
- an expression $T(t_1, \ldots, t_j)$ where for all $1 \leq i \leq j$ we have $t_i \in v$, $T \in (V \cup \mathcal{L})$, and the arity of T is j.

Definition 3.0.4. The *second order formulas of type* $\mathcal{L}$ (we abuse notation and abbreviate this as $\mathcal{L}$) are elements of the smallest set of expressions using the symbols $\mathcal{L} \cup v \cup V \cup W \cup \{\exists, \wedge, \neg, =\}$ containing the atomic formulas and satisfying, for $\Phi_1, \Phi_2 \in \mathcal{L}$, $x \in v$, and $X \in V$:

- $\neg\Phi_1 \in \mathcal{L}$
- $\Phi_1 \wedge \Phi_2 \in \mathcal{L}$
- $\exists x \Phi_1 \in \mathcal{L}$
- $\exists X \Phi_1 \in \mathcal{L}$

A *language of relational structures* is a language without any function symbols. For the rest of this paper, save section 4.2, we will deal only with languages of relational structures. The set of *first order formulas of type* $\mathcal{L}$ is the subset of the second order formulas of type $\mathcal{L}$ that do not contain relation variables. The set of *monadic second order formulas of type* $\mathcal{L}$ is the set of second order formulas where all relation variables have arity 1. The set of *second order existential formulas of type* $\mathcal{L}$ ($SO\exists$) are the second order formulas of the form $\exists V_1 \exists V_2 \ldots \exists V_n \phi$ where V_i is a second order relational variable and ϕ is a formula containing no quantification over set variables.

For a variable v in Φ say v is *bound* in Φ if it lies within the scope of a quantifier. Any variable that is not bound is *free*. A *sentence* is a formula containing no free variables. For a model $\mathcal{M}$ and sentence σ we write $\mathcal{M} \models \sigma$ when model $\mathcal{M}$ satisfies σ, for which a definition can be found in [5]. For a formula δ in language $\mathcal{L}$ and a relational structure G of $\mathcal{L}$ define $G(\delta)$ as the relation $\{(a_1, \ldots, a_n) | a_i \in G \text{ and } G \models \delta[a_1, \ldots, a_n]\}$.

Throughout this paper we are specifically concerned with graphs as relational structures. For our purposes, a graph will be definied as a relational structure in one of two ways. The first method is for a graph G to be defined as relational structure with universe A (containing all of the vertices and edges of G), unary predicates V and E, and binary predicate R. The set V is exactly the set of vertices of the graph, the set E is exactly the set of edges, and $R(x, y)$ holds if x is a vertex and y is an edge with x as an end. For vertices a and b of G that are adjacent we often abbreviate

$\exists z(R(a,z) \wedge R(b,z))$ as $\mathrm{adj}(x,y)$. For many graph problems we may need additional unary predicates over elements of the graph, so we use $P_1,\ldots,P_n$ (or as below, Y,P, and B) for those sets. Defining graphs in this manner allows us to quantify over both vertices and edges if we so desire. Monadic second order sentences of graphs of the form $\langle V,E,R\rangle$ are MS_2 *sentences*. We may refer to MS_2 sentences as simply MS sentences.

Alternatively, a graph will be defined as a relational structure with universe V, and binary predicate "adj" (adjacent). The set V is exactly the set of vertices, and for a and b in V, $\mathrm{adj}(a,b)$ holds if and only if a and b are adjacent. Monadic second order sentences of $\langle \mathrm{adj} \rangle$ are MS_1 *sentences*; they quantify over vertices or sets of vertices only.

The model checking problem is defined as follows:

INPUT: **A language $\mathcal{L}$, an MS sentence ϕ of $\mathcal{L}$, and a graph G which is a structure of $\mathcal{L}$**

QUESTION: **Does $G \models \phi$?**

This problem is nontrivial: for infinite graphs there is no possible way to check every possible mapping of the vertices and edges of the graph to the variables in the sentence. If we restrict the problem to finite graphs we can test every possible mapping to see if any provide a valid solution, but even this takes exponential time.

Many graph properties can be expressed in second order monadic logic. For example, here is a sentence σ such that for a graph G, $G \models \sigma$ if and only if G is 3-colorable:

$$\begin{aligned}\sigma := \exists Y \exists P \exists B \Big(&\forall x \Big(x \in V \to \big(((x \in Y) \wedge (x \notin P) \wedge (x \notin B)) \\ &\vee ((x \notin Y) \wedge (x \in P) \wedge (x \notin B)) \vee ((x \notin Y) \wedge (x \notin P) \wedge (x \in B)) \big) \Big) \wedge \\ &\forall x \forall y \Big(\big(x \in V \wedge y \in V \wedge \mathrm{adj}(x,y) \big) \to \\ &\neg \big((x \in Y \wedge y \in Y) \vee (x \in P \wedge y \in P) \vee (x \in B \wedge y \in B) \big) \Big) \Big)\end{aligned}$$

The graph G with unary relation P satisfies the sentence ψ if and only if the vertices of P are a dominating set:

$$\psi := \forall x (x \in V \to (\exists y (y \in V \wedge y \in P \wedge \mathrm{adj}(x,y))) \vee x \in P)$$

Let G be a graph with $V(G) = \{v_1, v_2, \ldots, v_n\}$. A monadic second order sentence ϕ such that for all graphs H, $H \models \phi$ if and only if H has G as an induced subgraph is the following:

$$\phi := \exists x_1 \exists x_2 \ldots \exists x_n \Bigg(\bigwedge_{1 \leq h \leq n} (x_h \in V) \bigwedge_{\substack{1 \leq i,j \leq n \\ (v_i,v_j) \in E(G)}} \mathrm{adj}(x_i,x_j)$$

$$\bigwedge_{\substack{1 \leq l,m \leq n \\ (v_l,v_m) \notin E(G)}} \neg \mathrm{adj}(x_l,x_m) \bigwedge_{\substack{1 \leq p,q \leq n \\ p \neq q}} x_p \neq x_q \Bigg)$$

Similarly, we can construct a MS sentence π to describe if a graph H has graph G as a minor. Again let $V(G) = \{v_1, v_2, \ldots, v_n\}$.

$$\pi := \exists X_1 \exists X_2 \ldots \exists X_N (\bigwedge_{1 \leq i \leq n} (\forall x \in X_i (x \in V) \wedge \text{“}X_i \text{ is connected”}) \wedge$$

$$(\forall v (v \in V \Rightarrow \text{“}v \text{ is in at most one of } X_i\text{”})) \wedge$$

$$\bigwedge_{\substack{1 \leq i,j \leq n \\ (v_i,v_j) \in E(G)}} \exists a \exists b ((a \in X_i) \wedge (b \in X_j) \wedge \mathrm{adj}(a,b)))$$

Intuitively, this sentence states that G is a minor in the following manner. There must exist a way to group connected vertex subsets of H, such that there exists a bijection between the vertex subsets and $V(G)$, each vertex of H is in at least one vertex subset. Also for vertex subsets X and Y of H mapped to vertices x and y in G, if x and y are adjacent then there must exist an edge between a vertex in X and a vertex in Y. This shows that G is a minor since the edges between vertices in the same vertex subset are the contracted edges, if there is not an edge between x and y then all of the edges between vertices of X and Y are deleted, and any deleted vertices are not mapped into vertex subsets.

The MS validity problem is defined as follows:

INPUT: **A language $\mathcal{L}$, an MS sentence ϕ of $\mathcal{L}$ and a class $\mathcal{C}$ of structures of language $\mathcal{L}$**

QUESTION: **For all $C \in \mathcal{C}$, is it true that $C \models \phi$?**

The MS validity problem is different than the previous problems since the input is a sentence and the difficulty lies in finding if all structures satisfy ϕ, whereas the model checking problem gave a structure and sentence as input and the difficulty lied in seeing if the specific structure satisfied the sentence. The *monadic second order theory* of K is the set of all MS_2 formulas that are true for all graphs in K, abbreviated as $\mathrm{TH}_2(K)$. The theory is *decidable* if there exists an algorithm that, given an MS sentence σ will determine if $\sigma \in \mathrm{TH}_2(K)$ in finite time. A class of structures K of language $\mathcal{L}$ has *decidable second order theory* if and only if the validity problem is decidable for every monadic second order sentence ϕ of language $\mathcal{L}$. A class of structures K of language $\mathcal{L}$ is *MS definable* if there exists an MS sentence σ such that a graph G is in K if and only if $G \models \sigma$.

3.1 Fagin's Theorem

In his 1974 Ph.D thesis, Fagin proved a deep connection between logic and computability. Specifically he showed a link between sets recognized by nondeterministic Turing machines and sets of structures definable by second order existential sentences. See [15] for a definition of Turing machine.

Theorem 3.1.1. *(Fagin) The set of all properties that are computable in polynomial time by a nondeterministic Turing machine (NP) is exactly the set of all properties that can be expressed as second order existential sentences.*

A sketch of the proof is provided. To show $SO\exists \subseteq NP$ we need to show that for any second order existential sentence σ there exists a nondeterministic polynomial time Turing machine N that accepts a binary encoding of a structure if and only if that structure satisfies σ. Let $\mathcal{A}$ be an input structure of language $\mathcal{L}$ with $|\mathcal{A}| = n$ and let $\sigma = \exists R_1 \ldots \exists R_k \phi$ with each variable R_i having arity r_i. All N does is, for each R_i, is encodes all possible r_i-tuples. This can be done in polynomial time with a nondeterministic algorithm (for each $a \in |\mathcal{A}|$ it nondeterministically chooses to encode a 0 or a 1). Once this is complete we have a structure $\mathcal{A}'$ over a new language $\mathcal{L}' = \mathcal{L} \cup \{R_1, \ldots R_k\}$. We have to check that $\mathcal{A}' \models \phi$ with a nondeterministic polynomial time Turing machine, but since ϕ is a first order query it can be shown that we may compute this by a deterministic log space Turing machine [15].

To show $NP \subseteq SO\exists$, we need to show that for a given Turing machine N that takes binary encodings of structures as input, there exists a sentence Φ such that the Turing machine accepts a structure $\mathcal{A}$ if and only if $\mathcal{A} \models \Phi$. Explicitly, $\Phi := \exists C_1 \ldots \exists C_g \exists \Delta \phi$ where ϕ is a first order formula and $(\mathcal{A}, \bar{C}, \Delta) \models \phi$ if and only if $(\bar{C}, \Delta)$ is an accepting run of N on $\mathcal{A}$. Specifically, let the alphabet of the tape be $\Gamma = \{\gamma_1, \ldots, \gamma_g\}$ and $C_i(s,t)$ holds true if at location s and time t the tape has symbol γ_i. Since our tape only has at most $n^k - 1$ locations on the tape that are not blank, we can code each C_i as a $2k$ unary relations with s and t each being binary encodings of a natural number less than n^k. Since N is a nondeterministic Turing machine without loss of generality we can assume it will make one of two possible choices. The relation $\Delta(t)$ holds for t if the machine chooses “1” at time t and it does not hold if the machine chooses “0” at time t. The first order formula ϕ is then a statement describing the rules of the Turing machine, such as both given time s and location t, $C_i(\bar{s},\bar{t})$ and $C_j(\bar{s},\bar{t})$ cannot hold for $i \neq j$.

Chapter 4

Interpretations

4.1 Definition

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be monadic second order languages over classes of structures K_1 and K_2. It would be useful if it were possible to express sentences in $\mathcal{L}_1$ as sentences in $\mathcal{L}_2$ along with a mapping from structures in K_1 to structures in K_2 such that a model in K_1 satisfies a sentence in $\mathcal{L}_1$ if and only if the mapping of the model satisfied the equivalent sentence in K_2. With that, one would be able to use the knowledge of $\mathcal{L}_2$ to answer questions about the original language.

Let α be a formula with one free variable, let ε be a formula with two free variables, and for each R_i relation in $\mathcal{L}_1$ let γ_{R_i} be a formula with r_i free variables where r_i is the arity of R_i. An *interpretation I* is the sequence of formulas $(\alpha, \gamma_{R_1}, \dots, \gamma_{R_k}, \varepsilon)$.

For each formula σ of $\mathcal{L}_1$ a formula for σ^I of $\mathcal{L}_2$ is defined inductively:

- $(x \in X)^I := \exists y(\varepsilon(x,y) \wedge y \in X)$
- $(x = y)^I := \varepsilon(x,y)$
- $(R_i(x_1,\dots,x_n))^I := \exists y_1,\dots,y_n \left(\left(\bigwedge_{j=1}^n \varepsilon(x_j,y_j)\right) \wedge \gamma_{R_i}(y_1,\dots,y_n)\right)$ for each R_i in L_1
- $(\Phi_1 \wedge \Phi_2)^I := \Phi_1^I \wedge \Phi_2^I$
- $(\neg\Phi)^I := \neg\Phi^I$
- $(\exists x\Phi)^I := \exists x(\alpha(x) \wedge \Phi^I)$
- $(\exists X\Phi)^I := \exists X(\forall x \in X(\alpha(x)) \wedge \Phi^I)$

Let G be a structure of language $\mathcal{L}_2$ with $G(\varepsilon)$ an equivalence relation such that for each relation R_i in $\mathcal{L}$, $G(\varepsilon)$ is congruent over the elements $G(\alpha)$ with respect to $G(R_i^I)$. We define the structure G^I of language $\mathcal{L}_1$ as the structure with universe $G(\alpha)/G(\varepsilon)$ and for each relation R_i we have $R_i^{G^I} = G(R_i^I)/G(\varepsilon)$.

Let K_1 and K_2 be classes of relational structures corresponding monadic second order languages $\mathcal{L}_1$ and $\mathcal{L}_2$. K_1 is *interpretable* into K_2 if there is an algorithm algorithm $\mathcal{A}$ to find the relations $\alpha, \varepsilon, \gamma_{R_1}, \ldots, \gamma_{R_l}$ such that:

$$G \cong (\mathcal{A}(G))^I$$

where I is the intrepretation $(\alpha, \varepsilon, \gamma_{R_1}, \ldots, \gamma_{R_l})$. In addition, K_1 is *linear time interpretable* if $\mathcal{A}$ runs in linear time with respect to the size of the universe of G.

Theorem 4.1.1. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be languages and let K_1 and K_2 be classes of structures of the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Let Φ be a sentence in $\mathcal{L}_1$, let I be a interpretation of K_1 onto K_2, and let G be a structure of K_2.*

$$G^I \models \Phi \text{ if and only if } G \models \Phi^I$$

Proof. First, we want to show that $G(\varepsilon)$ partitions $G(\alpha)$ such that the congruence of formula $G(\Phi^I)$ is preserved. Specifically let $b_i \in G(\alpha)$ and all $B_j \subseteq G(\alpha)$ with $(\ldots, b_i, \ldots, B_j, \ldots) \in G(\Phi^I)$. For all $c_i \in G(\alpha)$ with $\varepsilon(b_i, c_i)$ and for all $C_j \subseteq G(\alpha)$ such that for $c_j \in C_j$ it holds that for $\varepsilon(c_j, b_j)$ with $b_j \in B_j$, we have that
$(\ldots, c_i, \ldots, C_j, \ldots) \in G(\Phi^I)$ as well. We can show this using induction over Φ^I, which falls immediately from the inductive definition of Φ^I. Next we need to show that for all
$(\ldots, b_i, \ldots, B_j, \ldots) \in G(\Phi^I)$ it holds that $(\ldots, [b_i], \ldots, [B_j], \ldots) \in G^I(\Phi)$ where $[b_i]$ is the congruence class containing b_i. Again, this can be shown inductively by the construction of Φ^I. Once this is shown then we have our result. $\square$

An interpretation from K_1 to K_2 allows us to apply knowledge of the model checking problem of K_2 to K_1 as shown by the next theorem:

Theorem 4.1.2. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be languages and let K_1 and K_2 be classes of structures of the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, and K_1 is interpretable into K_2. If the model checking problem is decidable for K_2 then the model checking problem is decidable for K_1.*

Proof. Given a structure $G \in K_1$ and a sentence $\phi \in \mathcal{L}_1$ an algorithm for determining if $G \models \phi$ is as follows. Let $\mathcal{A}$ be the algorithm that interprets K_1 into K_2. Find $\mathcal{A}(G) \in K_2$ and ϕ^I and check if $\mathcal{A}(G) \models \phi^I$ (which we know takes finite time since the model checking problem is decidable for K_2). If $\mathcal{A}(G) \models \phi^I$ then $G \models \phi$, otherwise $G \not\models \phi$. □

With the concept of an interpretation we can also answer questions about decidability of theories. Let $\mathcal{L}_1$ and be $\mathcal{L}_2$ languages, let K_1 and K_2 be classes of structures of languages $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, and let I an interpretation from language $\mathcal{L}_1$ to $\mathcal{L}_2$ with the property that for every $G \in \mathcal{L}_2$ there exists an $H \in \mathcal{L}_1$ such that $G^I \cong H$, and for every $J \in \mathcal{L}_1$ there exists an $M \in \mathcal{L}_2$ with $M^I \cong J$. Then $\mathrm{TH}_{L_1}(K_1)$ is *interpretable* into $\mathrm{TH}_{L_2}(K_2)$. Rabin proved the following result:

Theorem 4.1.3. *If $TH_{L_1}(K_1)$ is interpretable into $TH_{L_2}(K_2)$ and $TH_{L_2}(K_2)$ is decidable then $TH_{L_1}(K_1)$ is decidable as well.*

If one were to find an MS interpretation from a class of graphs to the class of binary trees then the original class would have both a decidable theory and a model checking algorithm that could be done in linear time. The decidability of the theory monadic second order logic on classes of graphs with bounded tree width relies heavily on the work done by Rabin.

4.2 Tree Automata

A *tree* is a graph with no cycles. A *rooted tree* is a tree where one vertex is labeled as the *root*. The *children* of a vertex v in tree T with root r are all of the vertices adjacent to v whose distance from r is greater than that of v.

A *non-deterministic finite tree automaton* (NFTA) is a quintuple $M = (Q, \Sigma, \delta, Q_0, A)$ where Q is a set of states, Σ is an alphabet with arities $\rho(\Sigma)$, A is a subset of Q known as the accepting states, Q_0 is a subset of Q known as the start states, and δ is known as the set of transition rules with $\delta \subseteq \{Q^{\rho(\sigma)} \times \sigma \times Q | \sigma \in \Sigma\}$.

A *deterministic finite tree automaton* (DFTA) is a NFTA which has exactly one element in δ of the form $(q_1, q_2, \ldots, q_{\rho(\sigma)}, \sigma, q)$ for combination of $q_1, q_2, \ldots, q_{\rho(\sigma)} \in Q$ and $\sigma \in \Sigma$, and $|Q_0| = 1$.

A *binary deterministic finite tree automaton* is a DFTA where each σ in Σ has arity 2.

Let $\mathcal{M}$ be an NFTA and let T be a rooted tree where each vertex is labeled with an element of Σ with an arity greater than or equal to the number of children of the vertex. An *execution* of $\mathcal{M}$ on T is a relabeling of the vertices where the empty tree has a label q_0, and a vertex with label s and children assigned labels $l_1, l_2, \ldots, l_n$ respectively gets the new label q with $(l_1, l_2, \ldots, l_n, s, q) \in \delta$. An execution is *accepted* if the root is relabeled with an element of A. $\mathcal{M}$ *accepts* T if there exists an accepting execution of $\mathcal{M}$ on T. Note that in the case of DFTA there is exactly one execution of M on T.

Definition 4.2.1. Tree language $L(\mathcal{M})$ *recognized* by $\mathcal{M}$ is the set of all trees that are accepted by $\mathcal{M}$.

Property 4.2.1. *Let $\mathcal{M}$ be a deterministic finite tree automata with alphabet Σ. Testing if $L(\mathcal{M}) = \emptyset$ can be done in linear time; testing if $L(\mathcal{M})$ is the set of all terms of Σ can also be done in linear time.*

An alternative mathematical definition of tree automata exist using algebras. A finite Σ-*algebra* is a finite structure of language Σ. Let Σ be a set of functional symbols with exactly one constant $\Box$ and let A be a unary relation. The $\Sigma \cup \{A\}$-algebra $\mathcal{M}$ is equivalent to the deterministic finite tree automata $M = \langle |\mathcal{M}|, \Sigma, \delta_\Sigma, \Box^{\mathcal{M}}, A^{\mathcal{M}} \rangle$, where $\delta_\Sigma = \{(q_1, \ldots, q_n, \sigma, q) | \sigma \in \Sigma \wedge \rho(\sigma) = n \wedge q_i \in \mathcal{M} \wedge q = \sigma(q_1, \ldots, q_n)\}$. For each term t of Σ we associate an element of $|\mathcal{M}|$ inductively using the method: $\sigma(q_1, \ldots, q_n)^{\mathcal{M}} = \sigma^{\mathcal{M}}(q_1^{\mathcal{M}}, \ldots, q_n^{\mathcal{M}})$. A term t is accepted by M if and only if $t^{\mathcal{M}} \in A$.

Let A and B be two Σ-algebras. A Σ-*algebra homomorphism* from A to B is a map $\gamma : A \to B$ such that:

- for all function symbols $f \in \Sigma$ of arity n and all $a_1, \ldots, a_n \in A$,
  $\gamma(f^A(a_1, \ldots, a_n)) = f^B(\gamma(a_1), \ldots, \gamma(a_n))$;
- for all relation symbols $r \in \Sigma$ of arity n and all $a_1, \ldots, a_n \in A$,
  $\gamma(r^A(a_1, \ldots, a_n)) \Leftrightarrow r^B(\gamma(a_1), \ldots, \gamma(a_n))$.

An *epimorphism* is a homomorphism that is onto. If $\gamma : A \to B$ is an epimorphism, then B is the *homomorphic image* of A. Let Σ be a set of finitely many function symbols including exactly one constant symbol $\Box$ and a single unary relation symbol R. Let G be a subset of the terms of Σ and let $T_{\Sigma(G)}$ be the algebra with relation $R^{\Sigma(G)} = G$.

Theorem 4.2.2. *There exists a deterministic finite tree automaton M that recognizes G if and only if $T_{\Sigma(G)}$ has a finite homomorphic image.*

Proof. $\Leftarrow$ Let $M_G = \langle Q, \Sigma, \delta, Q_0, A \rangle$ be the DFTA that accepts G. Then the homomorphism γ which maps $t \in T_\Sigma$ to t^{M_G} is a homomorphism with a finite image.
$\Rightarrow$ Let γ be the homomorphism which maps $T_{\Sigma(G)}$ to a finite $\Sigma \cup \{A\}$-algebra B. Then the tree automata $M_G = \langle |B|, \Sigma, \delta, \Box^B, G^B \rangle$ with $\delta = \{(b_1, \ldots, b_n, \sigma, b) | \sigma \in \Sigma \wedge \rho(\sigma) = n \wedge b_i \in B \wedge b = \sigma^B(b_1, \ldots, b_n)\}$ is a DFTA which accepts $T_{\Sigma(G)}$. $\Box$

For further information on the relation between DFTAs and Σ-algebras see [16].

4.3 Decidability of the Theory of Monadic Second Order Logic on Binary Trees

Theorem 4.3.1 (Thatcher and Wright, 1968)**.** *[21] Let T be a tree and σ a monadic second order sentence. A DFTA M_σ can be constructed so that M accepts T if and only if $T \models \sigma$.*

Proof. We present the proof for the case where T is a binary tree. To prove this we will create a tree automata which recognizes the set of all binary trees which satisfy σ. Let $L(x,y)$ be the relation "x is a left child of y", and let $R(x,y)$ be the relation "x is a right child of y". For simplicity, we first take σ and apply transformation J by replacing all singular variables with set variables in the following manner:

- $(x \in Y)^J := (X \subseteq Y) \wedge \text{SING}(X)$
- $(x = y)^J := (X \subseteq Y) \wedge \text{SING}(X) \wedge \text{SING}(Y)$
- $$(L(x,y))^J := \hat{L}(X,Y) \wedge \text{SING}(X) \wedge \text{SING}(Y)$$ $$= (\text{“For all } x \in X \text{ and for all } y \in Y, L(x,y)\text{”}) \wedge \text{SING}(X) \wedge \text{SING}(Y)$$
- $$(R(x,y))^J := \hat{R}(X,Y) \wedge \text{SING}(X) \wedge \text{SING}(Y) =$$ $$= (\text{“For all } x \in X \text{ and for all } y \in Y, R(x,y)\text{”}) \wedge \text{SING}(X) \wedge \text{SING}(Y)$$
- $(P(x))^J := \hat{P}(X) \wedge \text{SING}(X) = (\text{“For all } x \in X, P(x)\text{”}) \wedge \text{SING}(X)$
- $(\exists x \Phi)^J := \exists X \Phi$
- $(\Phi_1 \wedge \Phi_2)^J := \Phi_1^J \wedge \Phi_2^J$
- $(\neg\Phi)^J := \neg\Phi^J$

Note that $\subseteq$ and SING are both shorthand for monadic second order logic statements. Thus we have a new *MS* problem σ^J. We build a binary finite tree automata to solve σ^J by inductively creating automata to solve subformulas of σ^J. We start with the following tree automata:

$\hat{P}_i(X)$

l	r	P_i	X	δ
q_1	-	-	-	q_1
-	q_1	-	-	q_1
-	-	0	1	q_1
-	-	-	-	q_0

$A = \{q_0\}$

$\hat{L}(X,Y)$

l	r	X	Y	δ
q_0	q_0	1	0	q_2
q_2	q_0	0	1	q_3
q_3	q_0	0	0	q_3
q_0	q_3	0	0	q_3
q_0	q_0	0	0	q_0
-	-	-	-	q_1

$A = \{q_3\}$

SING(X)

l	r	X	δ
q_0	q_0	0	q_0
q_0	q_0	1	q_1
q_1	q_0	0	q_1
q_0	q_1	0	q_1
-	-	-	q_2

$A = \{q_1\}$

$X \subseteq Y$

l	r	X	Y	δ
q_1	-	-	-	q_1
-	q_1	-	-	q_1
-	-	1	0	q_1
-	-	-	-	q_0

$A = \{q_0\}$

$\Sigma = \{0,1\}^{(p+m)}$ where p is the number of predicates and m is the number of free variables. s_0 is the initial state. In the charts above "-" denotes a wild card. The automata for $\hat{R}(X,Y)$ is constructed in the same way as the automata for $\hat{L}(X,Y)$. Using these rules, we inductively create larger subformulas in the following way:

- Let $M = (Q, \Sigma, A, Q_0, \delta)$ be the automata recognizing ϕ. $\neg\phi$ is recognized by $M_{\neg} = (Q, \Sigma, Q \setminus A, Q_0, \delta)$
- Let $M_1 = (Q_1, \Sigma, A_1, Q_1, \delta_1)$ be the automata recognizing ϕ_1, and $M_2 = (Q_2, \Sigma, A_2, Q_2, \delta_2)$ be the automata recognizing ϕ_1. $\phi_1 \wedge \phi_2$ is recognized by $M_{\wedge} = (Q_1 \times Q_2, \Sigma, A_1 \times A_2, (Q_1, Q_2), \delta_{\wedge})$ where $\delta_{\wedge}((s_a, s_b), (s_c, s_d), a) := (\delta_1(s_a, s_b), \delta_2(s_c, s_d))$
- Let $M = (Q, \Sigma, A, Q_0, \delta)$ be the automata recognizing ϕ. $\exists X\phi$ is recognized by $M := (2^Q, \Sigma, A_{\exists}, Q_0, \delta_{\exists})$, where $\delta_{\exists}(Q_l, Q_r, a) := \{\delta(q_l, q_r, b) | b \in \Sigma, q_l \in Q_l, q_r \in Q_r,$ and $b = a$ for all bits not representing $X\}$, and $A_{\exists} := \{Q | Q \cap A \neq \emptyset\}$.

□

Corollary 4.3.2. *The MS model checking problem on the set of all trees is decidable and can be preformed in linear time.*

Proof. Let σ be a sentence and let T a be tree. Construct M_σ using 4.3.1, if M_σ accepts T then $T \models \sigma$, otherwise $T \not\models \sigma$. Both the construction of M_σ and running M_σ on T can be done in linear time. □

Corollary 4.3.3. *The set of finite trees have a decidable MS validity problem.*

Proof. Given MS sentence σ an algoritm for determining if σ is true for all trees is as follows. let T_σ be the set of all trees with property σ and let M be the DFTA that accepts T_σ. If there exists a run of M which does not accept then σ is not valid; this can be checked in linear time because of 4.2.1. □

Thus, by providing an interpretation from a language to the language of binary trees, then model checking is decidable and the theory is decidable. Arnborg and Lagregern proved this for the class of graphs of bounded tree width, as shown in the next chapter. One final note, Rabin showed that the MS model checking problem and MS validity problem are decidable for infinite trees using the language of two successor functions.

Let $\Sigma = \{0, 1\}$ and Σ^* be the set of all finite words over Σ. The functions $r_0(x) = x0$ and $r_1(x) = x1$ are the *successor functions*, define the relation $x \leqq y \equiv \exists z(y = xz)$ and the lexicographic total ordering $x \preceq y \equiv x \leqq y \vee \exists u \exists v \exists z(x = z0u \wedge y = z1v)$. The monadic second order theory over the structure $\langle \Sigma^*, r_0, r_1, \leqq, \preceq \rangle$ is called the *second order theory of two successor functions*.

Theorem 4.3.4 (Rabin, 1964)**.** *The second order theory of two successor functions is decidable.*

Chapter 5

Arnborg, Lagergren, and Seese

Theorem 5.0.5 (Arnborg, Lagergren, Seese [1])**.** *Fix an integer k. The MS model checking problem for the class of graphs with tree width no more than k is decidable and can be done in linear time.*

The proof is structured as follows: first we define an algorithm for interpreting the class of graphs with tree width no more than k into the class of labeled binary trees. We then use the interpretation to find the equivalent MS-property for binary trees. We then apply the technique of 4.3.1 to determine if the binary tree models the interpreted sentence.

Theorem 5.0.6. *Every class of graphs K with bounded tree width k is linear time interpretable into a class of binary trees.*

Proof. Let $G := \langle A, V, E, P_1, \ldots, P_p, R \rangle \in K$ be a structure representing a graph with tree width no greater than k. Let A the universe of the structure which contains all the vertices and edges, let $V \subseteq A$ containing only the vertices, let $E \subseteq A$ containing only the edges, let each $P_i \subseteq A$ represent a labelling of the vertices and edges, and let R be the relation $R = \{\langle x, y \rangle :$ " the edge y has x as a vertex"$\}$. Since G has bounded tree width, we can find a tree decomposition with width k in linear time by 2.3.2. Let $(T, \mathcal{V})$ be that tree decomposition. Construct T' by connecting to each $t \in T$ adjacent vertices v_t for all $v \in V_t$ and e_t for any edge connecting two vertices in V_t with the rule that each edge must be represented exactly once. Now we construct new unary predicates $P_v := \{a_v | v \in V\}$ and $P_e := \{a_e | e \in E\}$. These predicates allow us to distinguish the new vertices in T'. Since the width of T is k, each V_t can have at most $k+1$ vertices. Thus there are at most $\binom{k+1}{2}$ pairs of vertices in each V_t. Arbitrarily enumerate the pairs for each vertex set and for $1 \leq i \leq \binom{k+1}{2}$ define the unary predicate R_i as follows. R_i is the collection of all a, b, and e where a and b are connected by edge e in G and for some V_t with $a, b \in V_t$ (a,b) was enumerated the value i. For each P_j define P'_j to be the set of all a_t where $P_j(a)$ for the equivalent a. By creating these new vertices in T' we may have created multiple a_t vertices that represent the same vertex, so we take this into account when constructing ε. To construct an ε which recognizes equivalent vertices we need additional unary predicates. Since we have a tree width k, we can color the graph G using $2k+2$ colors by starting with any V_t and coloring all of its vertices colors $C_1 \ldots C_{k+1}$. Then color all of the V'_t where (t,t') is an edge in T with colors $C_{k+2} \ldots C_{2k+2}$ in a manner that no color is shared by any vertex in $V_t \cup V'_t$. Repeat this process until the entire graph is fully colored. We create new unary predicates P_{C_i} to represent the vertices of each color.

We now define $\alpha, \varepsilon, \gamma_{P_1}, \ldots, \gamma_{P_p}$, and γ_R as follows:

- $\alpha(x) :=$ "x has degree 1 in T"
- $$\begin{aligned}\varepsilon(x,y) :=& (x = y) \vee (P_v(x) \wedge P_v(y) \wedge \bigvee_{1 \le i \le 2k+2} (P_{C_i}(x) \wedge P_{C_i}(y) \\ &\wedge \text{"there is a path between } x \text{ and } y \text{ where all of the internal} \\ &\text{nodes are adjacent to a node satisfying } P_{C_i}\text{"})\end{aligned}$$
- $\gamma_V(x) := P_V(x)$
- $\gamma_E(x) := P_E(x)$
- $\gamma_R(x,y) := P_V(x) \wedge P_E(y) \wedge \left(\bigvee_{1 \le i \le k+1} (R_i(x) \wedge R_i(y)\right)$
- For each P_i, $\gamma_{P_i}(x) := P'(x)$

It is trivial to show that the construction is valid. It is important to note that the number of unary predicates required to create the interpretation is bounded for all input graphs since the tree width of the input graph is at most k. Note that T' is still a tree, and if there are nodes with degree 2 in T' then each can be removed and replaced with a single edge and the tree decomposition is still valid. Thus from now on assume that T' has no vertices of degree 2.

We then interpret the tree T' into a binary tree. First create a new unary predicate P_c in such a way that 2-colors the tree T'. For each node in T' of degree $d \ge 3$ do the following: recursively replace each of these nodes with two nodes connected by an edge, with one node having degree $d-1$ and one having degree 3. Give both of the nodes the same P_c value as the original node. For one of the nodes assign all unary predicates besides P_c false and for the other assign the values exactly as was assigned to the original node. After this process take one of the vertices with degree 1 and replace it with a node of degree 2 and a node of degree 1 in the same fashion. Designate this new tree T''. All of the unary predicates are exactly the same as before, all we need to do is define the edge relation in our new language. $\gamma_a(x,y) := ((P_c(x) \wedge P_c(y)) \vee (\neg P_c(x) \wedge \neg P_c(y)) \wedge ($ "There exists a path between x and y of one color"$))$. Finally, we create a hierarchy of the nodes starting at the root and arbitrarily assign each child node as either left or right. We interpret nodes being adjacent in the previous tree as the one node being a child of the other. □

Now the work for 5.0.5 is complete. Since we have an interpretation from the class K of trees with tree width no greater than k to the class of binary trees, we can check if $T \in K$ satisfies MS sentence σ by applying the interpretation and then using the model checking result from 4.3.1. A similar result holds for clique-width:

Theorem 5.0.7. *[Courcelle] Fix an integer k. The MS model checking problem for the class of graphs with clique width at most k can be solved in linear time.*

Thus, if we fix k there is an algorithm such that if we are given a graph and its k-expression we can solve MS$_1$ problems in linear time. For classes of graphs with bounded clique-width there is also an analogue to 5.0.5:

Theorem 5.0.8 (Courcelle). *Let k be a fixed integer and let K be the class of graphs with clique width no more than k. The MS$_1$ theory of K is decidable.*

Chapter 6

Seese's Conjecture

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be monadic second order languages. An $(\mathcal{L}_1,\mathcal{L}_2)$-*definition scheme* is a tuple of formulas Δ satisfying the following properties:

- $\Delta = (\phi, \psi_1, \dots, \psi_k, (\theta_w)_{w\in\mathcal{L}_1 * k})$
- $k \geq 1, \mathcal{L}_1 * k = \left(\left(A,\hat{j}\right)|A \in \mathcal{L}_2, \hat{j} \in [k]^{\rho(A)}\right)$
- $\phi \in MS(\mathcal{L}_2, W)$
- $\psi_i \in MS(\mathcal{L}_2, W \cup \{x_1\})$ for all $i \in [k]$
- $\theta_{(A,\hat{j})} \in MS(\mathcal{L}_2, W \cup \{x_1 \dots x_\rho(A)\})$ for $(A,\hat{j}) \in \mathcal{L}_1 * k$

Let S be a structure of $\mathcal{L}_2$ and let η be a mapping from the set of variables W to elements of the structure S. A structure $T \in \mathrm{STR}(\mathcal{L}_1)$ with $|T| \subseteq |S| \times [k]$ is *defined in* (S,η) ($T = \mathrm{def}_\Delta(S,\eta)$) if the following properties hold:

- $S \models_\eta \phi$;
- $|T| = \{(d,i)|d \in |S|, i \in [k], S \models_\eta \psi_i[d]\}$;
- For each $A \in \mathcal{L}_1 : A_T = \{((d_1,i_1),\dots,(d_{\rho(A)}),i_(\rho(A)))$
  $\in |T|^{\rho(A)}|S \models_\eta \theta_{(A,(i_1,\dots,i_\rho(A)))}[d_1,\dots,d_{\rho(A)}\}$.

The *transduction defined by* Δ is the binary relation
$D_\Delta := \{(S,T)|T = \mathrm{def}_\Delta(S,\eta)$ for some W-assignment $\eta\}$. A transduction is an *MS (definable) transduction* if it is equal to some D_Δ for some $(\mathcal{L}_1,\mathcal{L}_2)$-definition scheme

Δ. A set of structures is *tree-definable* if it is the image of a set of trees under an MS transduction. If a class of graphs is tree definable then it has a decidable MS_1 theory.

Similar to an interpretation, these formulas allow us to create a structure of language $\mathcal{L}_1$ from a structure of language $\mathcal{L}_2$. In fact a definition scheme is a generalization of the notion of an interpretation, for any interpretation is also a definition scheme. All of our previous results about interpretations from Chapter 4 hold for transductions as well.

Conjecture 6.0.9. *Seese's Conjecture*
Let K be a class of graphs. If K has a decidable MS_1 theory, then K is tree-definable.

Courcelle, Engelfriet, and Oostrom [7] [11] showed that Seese's conjecture is equivalent to if K has a decidable MS_1 theory, then K has bounded clique width. That is to say that tree definability implies bounded clique width. If Seese's Conjecture is proven true then a *MS* recognizable class of graphs has decidable MS_1 theory if and only if it has bounded clique width.

A class of graphs K is tree definable if and only if the class of graphs can be represented by a vertex replacement grammar (written VR for short). Courcelle proved that every VR set is a subset of the class of graphs with clique width less than k for some k. Thus a class of graphs being tree-definable implies the class of graphs has bounded clique width. See [6] for a further explanation of VR sets.

A class of graphs *C satisfies Seese's Conjecture* (write $\mathcal{SC}(C)$) if all of its subsets having decidable MS_1 theory are tree definable. Clearly if a class of graphs has bounded clique width then it trivially satisfies Seese's Conjecture. Other special instances of the conjecture have been proven as well.

Theorem 6.0.10 (Seese [19])**.** *Let K be a class of graphs. If for each planar graph H there exists a planar $G \in K$ such that H is a minor of G, then the $TH_2(K)$ is undecidable.*

Corollary 6.0.11. *The class of planar graphs satisfy Seese's Conjecture*

Proof. Let K be a class of planar graphs. For K to have a decidable MS_1 theory, there must exist a planar graph H that is a forbidden minor of K. Robertson and Seymour proved that for every planar forbidden minor H, there exists a natural number n_H such that the tree widths of the graphs of K are bounded by n_H. Thus K has bounded tree width and there exists a transduction from them to the class of trees (as shown in 5). □

Seese also proved that Seese's Conjecture holds for classes of graphs with a decidable MS_2 theory. The class $\mathcal{U}_k$ of *uniformly k-sparse graphs* is the class of finite graphs which have the property that every finite subgraph has the number of edges in the subgraph bounded by k times the number of vertices in the subgraph. Courcelle proved that any MS_2 formula can be translated into an equivalent MS_1 formula for these classes of graphs, thus the class $\mathcal{U}_k$ satisfies Seese's Conjecture.

Let C and D be two classes of graphs. We say *D reduces to C with respect to Seese's Conjecture* if we can prove D satisfies Seese's Conjecture by assuming C satisfies Seese's Conjecture. An *MS coding* of C onto D is a pair of functional MS transductions (ϕ, ψ) with the ψ being the inverse of ϕ. If there exists an MS coding of D onto C then D reduces to C. If there exists an MS coding from C onto $\mathcal{U}_k$ then C satisfies Seese's Conjecture. The concept of reduction only makes sense because the image of a set of graphs with bounded clique width under an MS transduction have bounded clique width as well. Several types of graphs have been found to reduce to each other so that if any one of them satisfies Seese's Conjecture then all graphs satisfy Seese's Conjecture, and thus the conjecture is proven.

Theorem 6.0.12 (Courcelle [9])**.** *The following types of graphs all reduce to each other with respect to Seese's Conjecture:*

1. *Undirected Graphs*
2. *Bipartite undirected graphs*
3. *Chordal Graphs*
4. *Split Graphs*
5. *Directed Graphs*
6. *Directed Acyclic Graphs*

Chapter 7

Examples

7.1 Example of Dynamic Programming Using Tree Decompositions

In this section we will show how for a given graph G and a tree decomposition $\langle \mathcal{V}, T \rangle$ with width at most k, the weighted independent set problem can be solved in linear time.

The weighted independent set problem states that:

INPUT: A graph G, a weighting function $\tau : V(G) \to \mathcal{N}$, and an integer c

QUESTION: Is there a set of pairwise adjacent vertices $A \subseteq V(G)$ such that $\sum \tau(a)_{a \in A} \geq c$?

To solve this problem, we need to use a special type of tree decompositions. A *nice tree decomposition* is a tree decomposition that has one vertex labeled as the root and each vertex i in T falls into one of the following categories:

- Leaf: i is a leaf in T and $|V_i| = 1$;
- Join: i has exactly two children, j and k, and $V_i = V_j = V_k$;
- Introduce: i has exactly one child j and there exists a vertex in G such that $V_i = V_j \cup \{v\}$;
- Forget: i has exactly one child j and there exists a vertex in G such that $V_i = V_j \setminus \{v\}$.

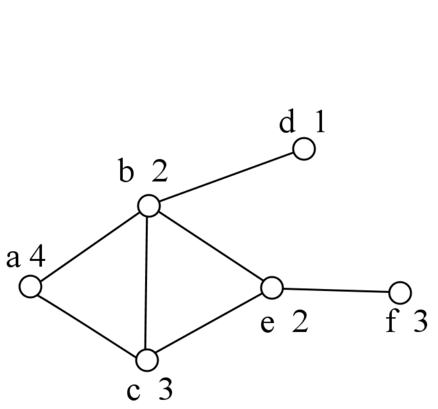
If G has a tree decomposition of width at most k then G has a nice tree decomposition with width at most k as well, which can be found in linear time given the original tree decomposition. Thus we assume that $\langle \mathcal{V}, T \rangle$ is a nice tree decomposition.

The algorithm for the weighted independent set problem is as follows [3]: For each i in T let D_i be the set of all of the descendants of i in T and let i itself be in D_i. We construct a graph $G_i = \bigcup_{j \in D_i} V_j$ for each i. For each i in T we also construct a table C_i of integers with an entry for each $S \subseteq V_i$. Since the width of the graph is at most k we know that each table has at most 2^{k+1} entries. For each $S \subseteq V_i$, $C_i(S)$ will be the maximum weight of an independent set of G_i containing S. In the case that S is not an independent set itself, $C_i(S) = -\infty$. The C_i tables are computed by starting with the leaves of T and working upwards. The values for each C_i table are computed from the children of i in the following manner:

- If i is a leaf node: table C_i will have exactly two entries: $\{\emptyset\}$ and $\{i\}$. Entry $C_i(\{\emptyset\}) = 0$ and entry $C_i(\{i\}) = \tau(i)$.
- If i is a join node: let j and k be the children of i. The power sets of V_i, V_j, and V_k are all equivalent. Thus to compute $C_i(S)$ we need to combine the values of $C_j(S)$ and $C_k(S)$. Since the value of $\tau(S)$ will have been added to both tables, we ensure we do not double count it. Table entry $C_i(S) = C_j(S) + C_k(S) - \tau(S)$.
- If i is an introduce node: Let $v \in V(G)$ be the vertex being introduced and let j be the child of i in T. Exactly half of the entries of C_i will have v in the subset. For any subset S that does not have v, $C_i(S) = C_j(S)$. For each subset $S \cup \{v\}$, if there exists an edge between v and any vertex w with $w \in S$, then $C_i(S \cup \{v\}) = -\infty$. If there does not exist such an edge, $C_i(S \cup \{v\}) = C_j(S) + \tau(v)$.
- If i is a forget node: Similarly to an introduce node, let $v \in V(G)$ be the vertex being forgotten and let j be the child of i in T. Since we can partition the subsets of V_j into pairs of S and $S \cup \{v\}$, let $C_i(S) = \max\{C_j(S), C_j(S \cup \{v\})\}$.

Let r be the root of the nice tree decomposition T. Graph G has a weighted independent set of size at least c if and only if $\max_{S \subseteq V_r}\{C_r(S)\} \geq c$. For a nice tree decomposition with $|\mathcal{V}| = n$, this algorithm takes time $O(2^k n)$.

Figure 7.1 shows a graph and a corresponding nice tree decomposition. Figure 7.2 shows the corresponding tables created by the algorithm.



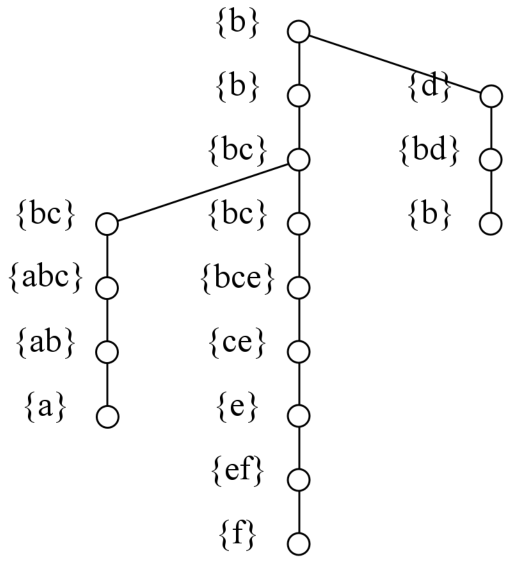
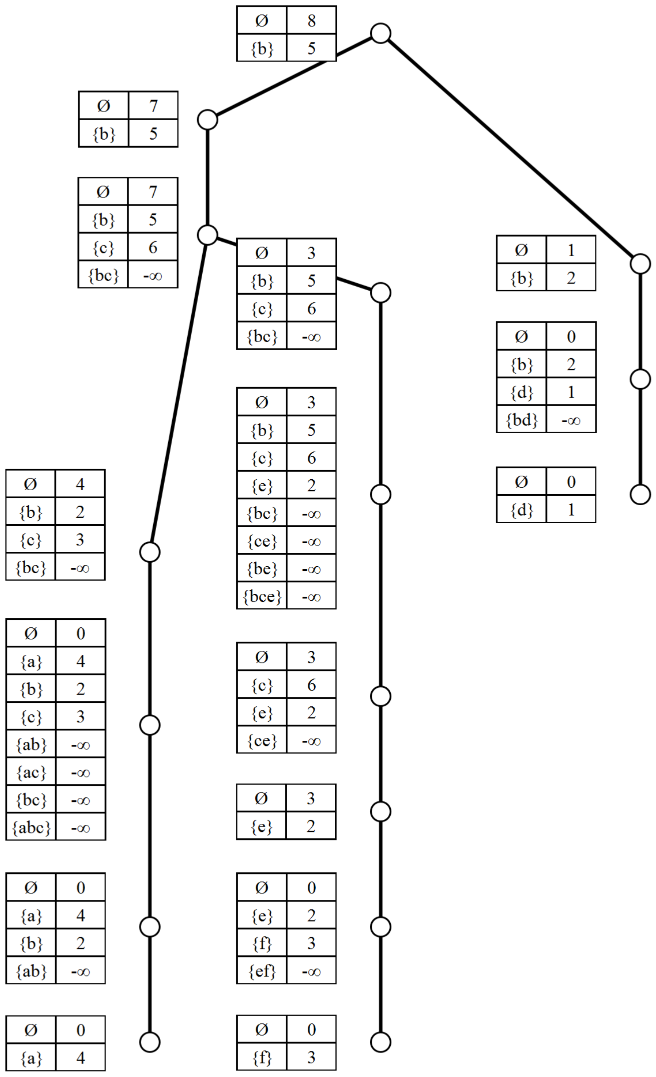


Figure 7.1: Graph G and corresponding nice tree decomposition $(\mathcal{V}, T)$ of G



Ø	8
{b}	5

Ø	7
{b}	5

Ø	7
{b}	5
{c}	6
{bc}	-∞

Ø	3
{b}	5
{c}	6
{bc}	-∞

Ø	1
{b}	2

Ø	0
{b}	2
{d}	1
{bd}	-∞

Ø	0
{d}	1

Ø	3
{b}	5
{c}	6
{e}	2
{bc}	-∞
{ce}	-∞
{be}	-∞
{bce}	-∞

Ø	4
{b}	2
{c}	3
{bc}	-∞

Ø	0
{a}	4
{b}	2
{c}	3
{ab}	-∞
{ac}	-∞
{bc}	-∞
{abc}	-∞

Ø	3
{c}	6
{e}	2
{ce}	-∞

Ø	3
{e}	2

Ø	0
{a}	4
{b}	2
{ab}	-∞

Ø	0
{e}	2
{f}	3
{ef}	-∞

Ø	0
{a}	4

Ø	0
{f}	3

Figure 7.2: Tables generated by running the weighted independent set algorithm on $(\mathcal{V}, T)$

7.2 Example Interpretation from a Graph of Bounded Tree Width to a Binary Tree

For this example we show how to interpret a graph with a tree width at most 2 into a binary tree described in Chapter 5. Let G be the graph with one unary predicate P_1 as seen in Figure 7.3. We are given an MS problem P as well however it is not relevant to this discussion. Assume G is also given with tree decomposition T as seen in Figure 7.4; clearly $\text{twd}(G) = 2$. T' is constructed from T and is shown in Figure 7.5. Immediately we can find the new unary predicates $P_v = \{a_1, b_1, c_1, b_2, d_2, f_2, b_3, d_3, e_3\}$ and $P_e = \{g_1, h_1, i_1, j_2, l_2, k_3\}$. Since $\binom{k+1}{2} = 3$, we require the unary predicates R_1, R_2, and R_3. To find these, we enumerate the pairs of vertices in each vertex set as follows:

-	V_1	V_2	V_3
1	$\{a_1, b_1\}$	$\{b_2, d_2\}$	$\{b_3, d_3\}$
2	$\{b_1, c_1\}$	$\{d_2, f_2\}$	$\{d_3, e_3\}$
3	$\{c_1, a_1\}$	$\{f_2, b_2\}$	$\{e_3, b_3\}$

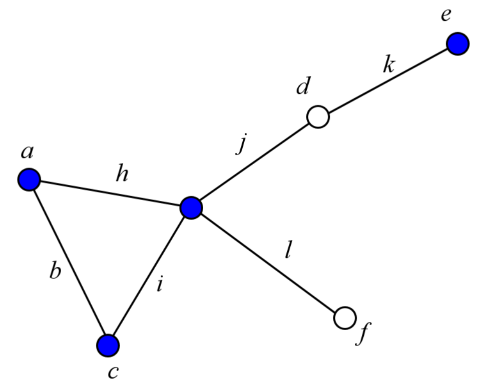


Figure 7.3: Graph G

Thus $R_1 = \{a_1, b_1, h_1, b_2, d_2, j_2\}$, $R_2 = \{b_1, c_1, i_1, d_3, e_3, k_3\}$, and $R_3 = \{c_1, a_1, g_1, f_2, b_2, l_2\}$. Also we have $P_1' = \{a_1, b_1, b_2, b_3, c_1, e_3\}$.

Since $2k + 2 = 6$ we have six colors with three colors in each of the two color groups $\{1, 2, 3\}$ and $\{4, 5, 6\}$. The graph coloring of G is shown in Figure 7.6, leaving the following unary predicates:

- $P_{C_1} = \{a_1, e_3\}$
- $P_{C_2} = \{c_1\}$

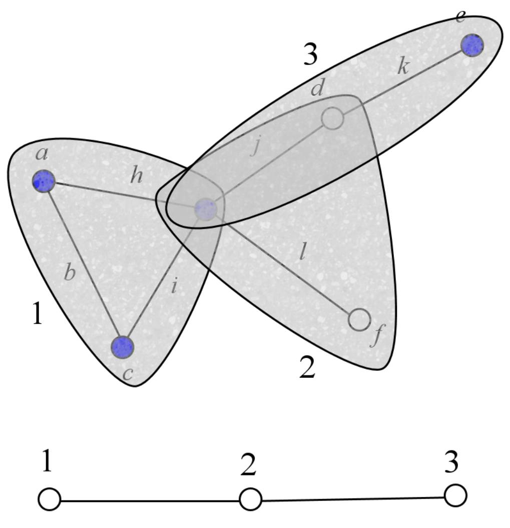


Figure 7.4: The tree decomposition of graph G

- $P_{C_3} = \{b_1, b_2, b_3\}$
- $P_{C_4} = \{d_2, d_3\}$
- $P_{C_5} = \{f_2\}$
- $P_{C_6} = \{\}$

We can now define $T'(\alpha)$, $T'(\varepsilon)$, $T'(\gamma_{P_1})$, and $T'(\gamma_R)$ as follows:

- $T'(\alpha) = \{a_1, b_1, b_2, b_3, c_1, d_2, d_3, e_3, f_2, g_1, h_1, i_1, j_2, k_3, l_2\}$
- $T'(\varepsilon) =$ the reflexive and symmetric closure of the following set: $\{\{b_1, b_2\}, \{b_2, b_3\}, \{b_3, b_1\}, \{b_1, b_2\}\}$
- $T'(\gamma_{P_1}) = P'_1 = \{a_1, b_1, b_2, b_3, c_1, e_3\}$
- $T'(\gamma_R) = \{(x,y) | P_V(x) \wedge P_E(y) \wedge \left(\bigvee_{i=1}^{3}(R_i(x) \wedge R_i(y))\right)\}$

Our next step is to interpret T' onto T''. T'' is the rooted binary tree shown in figure 7.7. Our new relations are the following:

- $T''(\alpha) = \{a_1^1, b_1^1, b_2^1, b_3^1, c_1^1, d_1^1, d_2^1, e_3^1, f_2^1, g_1^1, h_1^1, i_1^1, j_2^1, k_3^1, l_1^1, 1_1^1, 2_1^1, 3_1^1\}$. These are all the vertices with superscript 1.

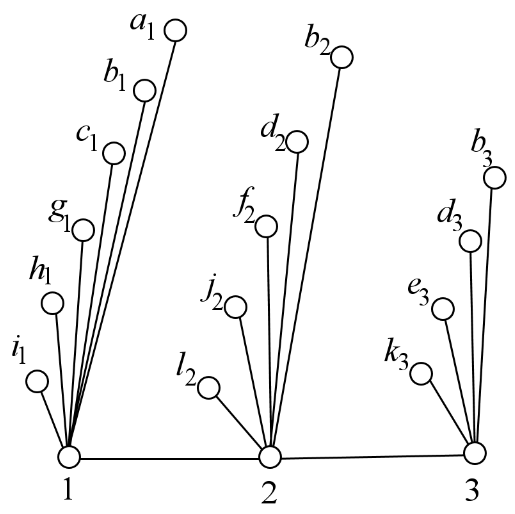


Figure 7.5: Tree T', constructed from the tree decomposition of G.

- $T''(\varepsilon) =$ the reflexive and symmetric closure of the following set: $\{\{1^1, 1^2\}, \{1^2, 1^3\}, \{1^3, 1^4\}, \ldots\}$. Any two vertices satisfy the relation if they are the same or if there is a path between them of exactly one color.
- $T''(P_i) = \{x^1 | x \in V(T') \wedge P_i(x)\}$
- $T''(\text{adj}) = \{(x,y) | \exists \varepsilon x, z_1 \exists \varepsilon y, z_2 (R(x,y) \vee R(y,x) \vee L(x,y) \vee L(y,x))\}$

Once we take the MS problem P and convert it to a problem using $T''(\alpha)$,$T''(P_i)$, $L(x,y)$, and $R(x,y)$ we can construct the tree automata inductively using the rules described in the previous section. Because the conversion process is creates a very long sentence, for an example of tree automata construction we shall use the following sentence:

$$\exists x(P_1(x))$$

We assume that there is only one unary predicate P_1. While the sentence could not have been created by using the construction from the original MS problem P, it better illustrates the process.

The first step is to remove the individual variables by constructing an equivalent sentence using only set variables. This sentence is

$$\exists X(\hat{P}_1(X) \wedge \text{SING}(X)))$$

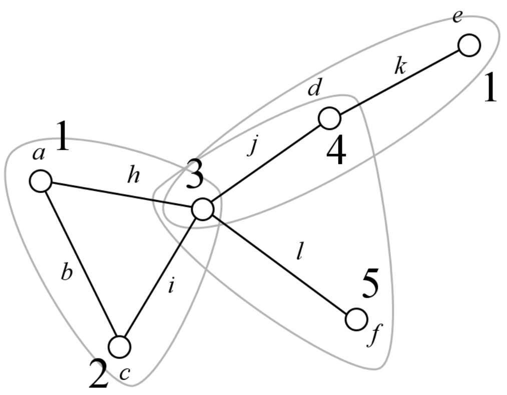


Figure 7.6: Graph G coloring using 6 colors.

We now inductively create the automata.

The automata for $\hat{P}_1(X)$ has $Q = q_0, q_1$, $\Sigma = \{00, 01, 10, 11\}$, $A = q_0$, and for δ:

l	r	P_1	X	δ
q_1	-	-	-	q_1
-	q_1	-	-	q_1
-	-	0	1	q_1
-	-	-	-	q_0

The automata for SING(X) has $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{00, 01, 10, 11\}$, $A = q_1$, and for δ:

l	r	X	δ
q_0	q_0	0	q_0
q_0	q_0	1	q_1
q_1	q_0	0	q_1
q_0	q_1	0	q_1
-	-	-	q_2

The automata for $\hat{P}_1(X) \wedge \text{SING}(X)$ then has $Q = \{q_{00}, q_{01}, q_{02}, q_{10}, q_{11}, q_{12}\}$, $\Sigma = \{00, 01, 10, 11\}$, $A = q_{01}$, and some sample transitions in δ are as follows:

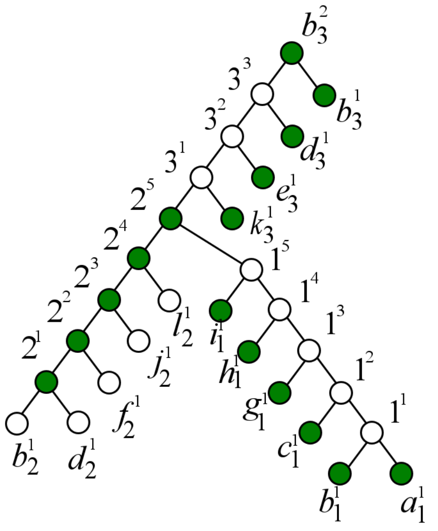


Figure 7.7: Rooted binary tree T'' with labels.

l	r	P_1	X	δ
q_{00}	q_{00}	0	0	q_{00}
q_{00}	q_{01}	0	1	q_{12}
q_{00}	q_{02}	0	0	q_{02}
q_{00}	q_{10}	0	1	q_{11}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

The automata for $\exists X(\hat{P}_1(X) \wedge \mathrm{SING}(X)))$, the final automata, has

$$Q = \{\{q_{00}\},\{q_{01}\},\{q_{02}\},\{q_{10}\},\{q_{11}\},\{q_{12}\},\ldots,\{q_{00},q_{01},q_{02},q_{10},q_{11},q_{12}\}\}$$

$\Sigma = \{0,1\}$, representing if the current vertex of the tree is in the set P_1

$$A = \{\{q_{01}\},\{q_{01},q_{00}\},\{q_{01},q_{02}\},\ldots,\{q_{00},q_{01},q_{02},q_{10},q_{11},q_{12}\}\}$$

The automata has δ transitions of the form:

l	r	P_1	δ
$\{q_{00}\}$	$\{q_{10},q_{01}\}$	0	$\{q_{01},q_{10},q_{11},q_{12}\}$
$\{q_{00}\}$	$\{q_{02}\}$	1	$\{q_{02}\}$
$\{q_{00}\}$	$\{\}$	1	$\{\}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Chapter 8

Concluding Remarks and Open Problems

We have seen throughout this paper the expressive power of tree decompositions and k-expressions. By using interpretations and transductions to map a class of structures K_1 of language $\mathcal{L}_1$ into a class of structures K_2 of language $\mathcal{L}_2$, we find that many of the properties of K_2 hold of K_1, and tree decompositions and k-expressions both provide the basis for mapping functions. One field of research is to rather than using classes of graphs with bounded tree width or clique-width, find a different measure and use it to make a transduction into a class of structures that are more convenient. Other measures currently exist, such as branch width and rank width, which are used on matroids and directed graphs respectively. Although if Seese's conjecture is proven, then we would know that any class of graphs with decidable MS_1 theory has bounded clique width. Thus, bounding other graph measures and proving that classes of graphs with that bounded measure have decidable MS_1 theory would not be necessary since those classes of graphs would also have bounded clique width. Of course, if an interpretation from the class of *all* graphs into a class of graphs with decidable MS theory could be found, then P=NP would be proven since the dominating set problem is NP-complete and can be written as a MS sentence.

Bibliography

[1] S. Arnborg, J. Lagergren, D. Seese, Easy Problems for Tree-Decomposable Graphs, in *Journal of Algorithms* 12, 308-340, 1991.

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

[3] H. Bodlaender and A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth, in *Computer Journal* (to appear).

[4] H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth, in *SIAM Journal on Computing*, 25:1305-1317, 1996.

[5] S. Burris and H.P. Sankappanavar. *A Course in Universal Algebra*. Millenium Edition. http://www.math.uwaterloo.ca/ snburris/htdocs/ualg.html

[6] B. Courcelle, Structural Properties of Context-Free Sets of Graphs Generated by Vertex Replacement, in *Information and Computation* 116 (1995), 275-293.

[7] B. Courcelle, J. Engelfriet, A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars, Mathematical Systems Theory 28 (1995) 515552.

[8] B. Courcelle, and S. Olariu. Upper Bounds to the Clique-Width of Graphs, in *Discrete Applied Mathematics* 101 (2000): 77-114.

[9] B. Courcelle, and S. Oum. Vertex-minors, monadic second-order logic, and a conjecture by Seese, in *Journal of Combinatorial Theory Series* 97-1: 91-126, 2007.

[10] R. Diestel. *Graph Theory*. 3rd. Springer, 2006.

[11] J. Engelfriet, V. van Oostrom, Logical description of context-free graph-languages, Journal Computing System Sciences 55 (1997) 489503.

[12] M. Fellows and F. Rosamond. Proving NP-Hardness for Clique-Width I: Non-Approximability of Sequential Clique-Width, in *Electronic Colloquium on Computational Complexity*. 80 (2005).

[13] M. Fellows and F. Rosamond. Proving NP-Hardness for Clique-Width II: Non-Approximability of Clique-Width, in *Electronic Colloquium on Computational Complexity*, 81 (2005).

[14] P. Hlineny, S. Oum, D Seese, and G. Gottlob. Width Parameters Beyond Tree-Width and Their Applications, in *Computer Journal* (to appear).

[15] N. Immerman. Descriptive Complexity. New York: Springer-Verlag, 1998.

[16] D. Kozen. *Automata and Computability*. 3rd Edition, New York: Springer, 1997.

[17] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Bulletin of the American Mathematical Society*, 74-5:1025-1029, 1968.

[18] S. Oum and P. Seymour. Approximating clique-width and branch-width, in *Journal of Combinatorial Theory Series B*, 96-4: 514-528, 2006

[19] D. Seese. The structure of the models of decidable monadic theories of graphs, in *Annals of Pure and Applied Logic*, 53 169-195, (1991).

[20] J.A. Telle and A. Proskurowski. Algorithms for Vertex Partitioning Problems on Partial k-Trees, in *SIAM Journal on Discrete Mathematics* 10 (1997): 529-550

[21] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decidion problem of second-order arithmetic. *Mathematical Systems Theory*, 2:57-81,1968.