Space and Circuit Complexity of Monadic Second-Order Definable Problems on Tree-Decomposable Structures

Michael Elberfeld

From the Institute of Theoretical Computer Science
of the University of Lübeck
Director: Prof. Dr. Rüdiger Reischuk

Space and Circuit Complexity of Monadic Second-Order Definable Problems on Tree-Decomposable Structures

Dissertation
for Fulfillment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Science / Engineering

Submitted by

Michael Elberfeld
from Friesoythe

Lübeck, 2012

First referee:	Prof. Dr. Till Tantau
Second referee:	Prof. Dr. Heribert Vollmer
Third referee:	Prof. Dr. Rolf Niedermeier
Date of oral examination:	10 July 2012
Approved for printing.	Lübeck, 14 December 2012

Preface

During the summer of 2009 I started to work on the topic that turned into my doctoral thesis project. Together with my advisor Till Tantau and in collaboration with Andreas Jakoby, I considered the question of "how hard is it to count the number of paths between vertices in series-parallel graphs?" During our first meetings it was a pleasure to see Andreas and Till solving this problem; I went on and generalized it to graphs of bounded tree width (Chapter 6 of the thesis). Adjusting the developed techniques to solve monadic second-order definable problems resulted in a paper published in the proceedings of the *51st Annual* IEEE *Symposium on Foundations of Computer Science* (Elberfeld, Jakoby, and Tantau, 2010). After spending the summer of 2010 in Tel Aviv working in computational biology, I picked up the logspace theme in Lübeck again. Driven by understanding the complexity of the subset sum problem with unary weights, it turned out that the results for logspace can be adjusted to work in the context of constant-depth circuits. I developed the proof of this result in spring 2011, which is based on the notion of multiset tree automata and an algebraic representation of their computations (Chapters 2 and 4). Many discussions with Till during the summer of 2011 led to refining parts of the logspace approach in terms of logarithmic-depth circuits (Chapter 5). The circuit-based results are published in the proceedings of the *29th International Symposium on Theoretical Aspects of Computer Science* (Elberfeld, Jakoby, and Tantau, 2012b). The question of whether it is possible to prove a variant of the results for constant-depth circuits in terms of first-order formulæ without build-in predicates was raised by Martin Grohe when I visited his group in the summer of 2011; we discussed proving it based on his notion of treelike decompositions. Later, I came up with the idea of proving it based on the recursive definition of tree depth, which avoids the use of any decompositions inside the proof (Chapter 3). This result is part of a larger work that will appear in the proceedings of the *27th Annual* ACM/IEEE *Symposium on Logic in Computer Science* (Elberfeld, Grohe, and Tantau, 2012a).

With the present thesis I want to give a comprehensive picture of the computational and descriptive complexity of monadic second-order definable problems on tree-decomposable structures. While, especially in the area of circuit complexity and first-order logic, much work is devoted to lower bounds—showing what cannot be done—, my thesis focuses on what can be done using small complexity classes and related logical formalisms. I hope that the general framework of problem definitions that are based on monadic second-order logic and input instances that can be represented as tree decompositions makes understanding the power of logspace, uniform circuit families, and first-order formulæ more accessible and, thus, helps to develop future works in this area.

I want to thank Till Tantau for his support during the last five years. I also want to thank Martin Grohe and Andreas Jakoby for the fruitful collaborations that helped to shape the present thesis, and Christoph Stockhusen for our discussions on logspace algorithms and number problems.

My warmest thanks go out to my family, friends, and my partner Bianka; for their support and close companionship during the last years.

Michael Elberfeld
Lübeck, May 2012

Abstract

A famous theorem of Courcelle states that every problem that is definable in monadic second-order (MSO) logic can be solved in linear time on input structures of bounded tree width. While Courcelle's result is optimal from the algorithmic point of view, this thesis shows how to solve *monadic second-order definable decision, counting, and optimization problems* on tree-width-bounded structures optimally from the complexity theoretic point of view. Beside bounded tree width, the related notion of bounded tree depth is considered.

A first set of results covers input *structures of bounded tree width*, which admit tree decompositions of bounded width. It is shown that all MSO-definable decision, counting, and optimization problems are solvable in *deterministic logarithmic space* (logspace) in this setting. Transferring the powerful framework of MSO-definability to logspace allows one to settle the complexity of problems related to *finding paths and matchings* in graphs: It is shown that answering reachability queries and counting the number of perfect matchings in graphs of bounded tree width can be done in logspace. Moreover, by solving problems on tree-width-bounded graphs as subroutines, problems on inputs of unbounded tree width are solvable in logspace; like deciding whether undirected loop-free graphs have cycles whose length is a multiple of any constant. These results rest on the new technique of constructing width-bounded tree decompositions in logspace.

Once *tree decompositions are available*, a task shown to be complete for logspace, actually solving MSO-definable decision and counting problems can be done by *logarithmic-depth Boolean and arithmetic circuits*, respectively. This finding itself unifies many *automaton evaluation* tasks studied in the area of logarithmic-depth circuits in an elegant manner using MSO-formulæ. To let logarithmic-depth circuits work on tree decompositions whose underlying trees have arbitrary depth, a preprocessing step is used that balances them using constant-depth threshold circuits.

The second set of results spans input *structures of bounded tree depth*, which admit tree decompositions of bounded width whose underlying trees have bounded depth. The logspace results are transferred to this setting to solve MSO-definable decision, counting, and optimization problems by *constant-depth Boolean, arithmetic, and threshold circuits*, respectively. These theorems apply to *number problems* like solving systems of any constant number of linear equations whose coefficients are given in unary. For their proofs input structures are turned into trees of bounded depth and unbounded degree. Translating MSO-formulæ to just the right automaton model for this situation and representing automata computations algebraically by arithmetic circuits lies at the heart of the proofs of these results.

In terms of computational power, constant-depth circuits are equivalent to first-order formulæ that have access to ordering and arithmetic predicates defined on the input's universe. It is shown that these build-in predicates are not needed to evaluate MSO-formulæ once input structures are extracted from their string encodings: *First-order formulæ and* MSO*-formulæ define the same problems on any class of structures of bounded tree depth.* The main building block in the proof of this result is a new constructive Feferman–Vaught-type composition theorem for unbounded partitions.

Table of Contents

CHAPTER 1

Introduction

Computer science and mathematical logic are intimately connected. Computer science is concerned with solving *computational tasks* like finding traveling routes in street networks by *programming* computing devices. Mathematical logic is concerned with understanding the interplay between *properties of logical structures* like graphs and formulæ that *describe* such properties. While both fields seem to be unrelated at first sight, mathematical logic found numerous applications in computer science. This is basically due to two fundamental connections. First, formulæ help to define computational tasks unambiguously and in an exact formal way: Instead of writing a functional specification of the computational task at hand in a semi-formal way, it can often be described in a succinct and exact way using logical formulæ that describe properties of structures. In case of the route planning task this means that, instead of writing a document that talks about maps with streets that cross in junctions and specify valid routes, one talks about graphs with edges that connect vertices and writes down a logical formula that defines paths. Second, the question of whether a formula is satisfied for a given structure can, itself, be viewed as a computational task; and solved using algorithms. The magic starts when we combine both observations. Once we know how to evaluate all formulæ of a certain logical formalism algorithmically, we can solve any problem that can be phrased as properties of logical structures using formulæ. For the route planning task this means that, instead of implementing a program based on a functional specification, just feeding the problem-describing formula into a formula evaluation procedure will, ideally, solve the problem. This *programming by describing* paradigm is present in almost all areas of computer science, and in the early to mid-20th century it was used to clarify the notion of computability itself. Today, logic is used to formulate database queries, verify hardware and software designs, and understand the limits of designing efficient algorithms.

The later application—designing efficient algorithms and understanding the limits thereof—can be seen as an offspring of the works of Büchi (1960), Elgot (1961), and Trakhtenbrot (1961), who found close connections between monadic second-order logic on strings and finite automata, as well as the works of Doner (1970) and Thatcher and Wright (1968), who extended this connection to trees. Monadic second-order logic is an appealing framework since its formulæ can express a wide range of problems, and translating them into automata yields the holy grail of algorithmics: linear time algorithms. While formulæ from monadic second-order logic can sometimes be hard to evaluate for structures of any kind, great success has been made in recent years on finding efficient algorithms that evaluate monadic second-order formulæ on ever larger classes of input structures that can be decomposed in a tree-like fashion. The thesis at hand complements the extensive work on understanding the *algorithmics* behind evaluating monadic second-order formulæ on tree-decomposable structures by working towards understanding its *computational complexity*. Upper bounds on the complexity of formula-defined problems in terms of space-bounded sequential and depth-bounded circuit computations are given. The given upper bounds turn out to be tight since the involved computations can be described in terms of monadic second-order logic on tree-decomposable structures.

1.1. Definable Problems on Tree-Decomposable Structures

Many computational problems can easily be defined in the language of monadic second-order logic (MSO-logic). Here inputs to the problems are string encodings of logical structures over a certain vocabulary for which we want to know whether some fixed formula holds, or not. In the case of the well known 3-COLORABILITY problem, an input is a string encoding of a graph $\mathcal{G}$, which is just a logical structures over the vocabulary $\tau = \{E^2\}$. For this graph we want to know whether the formula $\varphi_{\text{3-colorable}} :=$

$$\exists R\,\exists G\,\exists B\;\forall v\,\Big(R(v) \lor G(v) \lor B(v) \land$$
$$\forall w\,\big(E(v,w) \to \neg(R(v) \land R(w)) \land \neg(G(v) \land G(w)) \land \neg(B(v) \land B(w))\big)\Big)$$

holds; that is, whether the model relation $\mathcal{G} \models \varphi_{\text{3-colorable}}$ is true. The formula describes that vertices of the graph can be covered by three sets (colored by the three colors R, for red, G, for green, and B, for blue), such that adjacent vertices are not in a common set (do not have the same color). Thus, we have $\models \varphi_{\text{3-colorable}}$ by assigning a unique color to each vertex, but $\not\models \varphi_{\text{3-colorable}}$ since all pairs of vertices are adjacent. Beside using MSO-formulæ to define *decision* problems like 3-COLORABILITY, which only ask whether the model relation holds for a given structure and an MSO-formula without free variables, we can also use MSO-formulæ to define *counting* and *optimization* problems. Consider the formula $\varphi_{\text{dominates}}(X_1) = \forall v\,(X_1(v) \lor \exists w\,(X_1(w) \land E(v,w)))$ with a free set variable X_1 that is exactly true for a graph $\mathcal{G}$ and a subset of its vertices $D \subseteq V(\mathcal{G})$ if D is a dominating set in $\mathcal{G}$; that is, every vertex of $\mathcal{G}$ is part of D or is adjacent to a vertex that is part of D. Based on the formula $\varphi_{\text{dominates}}(X_1)$, we can define the problem of computing the number of dominating sets of $\mathcal{G}$; that means, *counting the number* of $D \subseteq V(\mathcal{G})$ with $\mathcal{G} \models \varphi_{\text{dominates}}(D)$. Moreover, we can ask to compute the smallest, *optimal*, $s \in \mathbb{N}$, such that there is a $D \subseteq V(\mathcal{G})$ with $\mathcal{G} \models \varphi_{\text{dominates}}(D)$ and $s = |D|$. This is the well known optimization problem DOMINATING-SET.

The problem 3-COLORABILITY and the decision variant of DOMINATING-SET are NP-complete when allowing graphs of any kind as inputs (Garey and Johnson, 1979), but if we consider only input graphs that have tree decompositions whose width is bounded by some constant, they can be solved in polynomial time. A *width-w tree decomposition* decomposes a graph into subgraphs of size w, called *bags*, that are arranged in a tree to satisfying certain connectedness and cover conditions; the tree structure of the decomposition captures the global tree-likeness of the input and the small subgraphs cover local connectivity patterns that may be far from being trees. Graphs that admit tree decompositions of width w have *tree width* at most w. A related notion we consider is the *tree depth* of a graph; graphs with tree depth d have tree decompositions where not only the width is bounded in terms of d, but, in addition, the depth of the underlying trees is bounded in terms of d. While tree width can be seen as measuring the similarity of graphs to trees (trees have tree width 1 and graphs with cycles have tree width at least 2), tree depth can be seen as measuring the similarity of graphs to star graphs (star graphs have tree depth 2, and the tree depth of a graph grows with the length of paths in it). We say that the tree width (tree depth) of a class of structures is *bounded* if there is a constant that upper-bounds the tree width (tree depth) of them. Solving computational problems on such *tree-decomposable* structures is often done by a two-step approach that first computes a tree decomposition and, then, solves the problem by using a bottom-up dynamic programming approach along the tree.

For structures of bounded tree width, the polynomial-time bound of solving MSO-definable problems has been refined from the algorithmic point of view: Linear-time sequential algorithms (Courcelle, 1990) and logarithmic-time parallel algorithms (Bodlaender and Hagerup, 1998) are known to solve MSO-definable decision problems for structures of bounded tree width and similar results hold for counting and optimization problems whose definition is based on MSO-formulæ. A variety of results of a similar flavour have been developed during the last years and are commonly called *algorithmic meta theorems* (Grohe and Kreutzer, 2011): Instead of presenting an algorithmic result for some particular problem, these theorems state that "all problems of a certain kind on structures of a certain kind are solvable by an efficient algorithm of a certain kind". The finding of Courcelle (1990), which is often just called *Courcelle's Theorem*, falls into this category since it shows that all MSO-definable problems for structures whose tree width is bounded are solvable in linear time.

Since many important problems are MSO-definable, Courcelle's Theorem and its variants yield unified frameworks for showing that numerous problems on structures of bounded tree width are efficiently solvable. Moreover, often algorithmic meta theorems show their real power when used as subroutines in algorithms solving problems that are normally (1) not MSO-definable, or (2) whose inputs are not tree-decomposable in an algorithmically useful way. A problem of the first kind is computing the chromatic number of a graph, which is the least number of colors that are needed to properly color a given graph. No MSO-formula $\varphi(X)$ is known that expresses that a graph has chromatic number $|X|$, but it is not hard to see (Flum and Grohe, 2006) that graphs of tree width w have chromatic number at most $w+1$. Thus, to compute the chromatic number, we can successively test whether a graph is 1-, 2-, ..., $(w+1)$-colorable using extensions of the formula for 3-COLORABILITY. A problem of the second kind is EVEN-CYCLE—deciding whether an undirected loop-free graph has a cycle of even length. This problem can be solved by first testing whether the tree width of the graph is higher than some constant. If this is the case, graph-theoretic methods show (Thomassen, 1988) that there is always an even cycle in such graphs and we answer "yes". If the tree width is bounded by the constant, we use Courcelle's Theorem to solve the problem via an MSO-formula that defines even-length cycles on the incidence representation of graphs.

Beside the impact on applying algorithmic meta theorems to particular problems, proving them contributes to understanding the fundamentals of solving computational problems: It drives the generalization and extension of existing algorithmic techniques that where developed for specific problems individually, as well as the development of new general techniques.

1.2. Main Results and Their Applications

The algorithmic meta theorems mentioned above provide a unified framework for finding fast sequential and parallel algorithms to solve MSO-definable problems on tree-decomposable structures. In the present thesis I transfer the idea of using MSO-based problem definitions to give algorithmic results in a unified way to giving tight complexity theoretic results in a unified way. During the course of the thesis, we work on the question of "what is the complexity of MSO-definable problems on tree-decomposable structures?" We already know that MSO-definable decision problems on structures of bounded tree width can be solved in polynomial time and, thus, lie in P, but how deep inside P can we place them? Bodlaender and Wanke contributed important steps to clarify this question: Bodlaender (1989) showed that all problems that are covered by Courcelle's Theorem lie in NC and Wanke (1994) refined this finding by showing that they lie in LOGCFL, a complexity class sandwiched between NL and AC^1, but proving hardness for this class and any of the covered problems remained an open question. The thesis exactly characterizes the complexity of the problems that are covered by Courcelle's Theorem and its variants for counting and optimization problems as well as for structures of bounded tree depth using space-bounded Turing machines and depth-bounded circuits. It contributes in three ways to the understanding of these problems and resource-bounded computational models:

Theorems: The theorems show that, depending on the particular tree decompositions that the inputs admit, decision, counting, and optimization problems that are defined using MSO-formulæ are solvable using logarithmic-space-bounded Turing machines, uniform logarithmic-depth circuits, or uniform constant-depth circuits.

Applications: The MSO-based framework for defining problems allows to detect the computational complexity of many problems whose complexity was unknown before, and unifies known results that were previously proved using problem-dependent techniques. The results cover complete problems for the corresponding classes.

Proof Techniques: Many proof techniques are either newly developed or transferred from the area of time-efficient sequential and parallel algorithms to fit the needs of space-efficient and circuit computations.

In the remainder of the present section, the main theorems and an overview over their applications is given. Proof techniques are only mentioned briefly in the present Section, a detailed discussion of them is given in Section 1.3.

Bounded Tree Width and Logarithmic Space. Two key problems in the study of logarithmic-space-bounded deterministic Turing machines (logspace DTMS) are REACHABILITY, detecting whether there is a path from some start to some target vertex in a directed graph, and PERFECT-MATCHING, detecting whether an undirected graph has a perfect matching. The reachability problem was shown to lie in logspace when restricted to graphs of tree width 2 (Jakoby, Liśkiewicz, and Reischuk, 2006) and the same upper bound holds for directed REACHABILITY and PERFECT-MATCHING when the inputs are restricted to be k-trees (Das, Datta, and Nimbhorkar, 2010), the maximal graphs of tree width k. Since paths and matchings are MSO-definable on the incidence representation of graphs, these results are covered by the following theorem, which drops the LOGCFL bound of Wanke (1994) to logspace. For Theorems 1.1 to 1.8, the input structures $\mathcal{A}$ are encoded as strings $\mathrm{str}(\mathcal{A})$ in a standard way; the details of the string encoding are described in Section 4.1.

THEOREM 1.1 (Decision in Logarithmic Space). *For every $w \in \mathbb{N}$, and every* MSO*-formula φ over some vocabulary τ, there is a logspace* DTM *that, on input of a τ-structure $\mathcal{A}$ of tree width at most w, decides whether $\mathcal{A} \models \varphi$ holds.*

Beside its application to detect the existence of paths and perfect matchings, the theorem can be used as a subroutine in a logspace algorithm that detects, for undirected loop-free input graphs whose tree width is not restricted, whether they contain a cycle of even length. Moreover, this even holds for any constant number $m \in \mathbb{N}$ and the question whether the length of the cycle is a multiple of m. (This result and the applications to L-hard path and matching problems are detailed in Section 6.4.)

To move from MSO-definable decision problems to counting and optimization problems, our work horse is a theorem that shows how to *count the number of solutions with respect to their cardinalities* in logspace. To formulate it, we need a bit of terminology: Let $\varphi(X_1, \dots, X_k, Y_1, \dots, Y_\ell)$ be an MSO-formula with two sets of free set variables, namely the X_i and the Y_j, and let $\mathcal{A}$ be a structure with universe A. The *solution histogram* of $\mathcal{A}$ and φ, denoted by $\mathrm{histogram}(\mathcal{A}, \varphi)$, is a k-dimensional integer array that tells us how many solutions of a certain size exist. In detail, let $s = (s_1, \dots, s_k) \in \{0, \dots, |A|\}^k$ be an index vector that prescribes sizes for the sets that are substituted for the X_i. Then $\mathrm{histogram}(\mathcal{A}, \varphi)[s]$ equals the number of $(S_1, \dots, S_k, S'_1, \dots, S'_\ell) \in \mathcal{P}(A)^{k+\ell}$ with $|S_1| = s_1, \dots, |S_k| = s_k$ and $\mathcal{A} \models \varphi(S_1, \dots, S_k, S'_1, \dots, S'_\ell)$. In other words, we count how often φ can be satisfied when the sets assigned to the X_i-variables have certain sizes, but impose no restrictions on the sizes of the Y_j. For the formula $\varphi_{\text{dominates}}(X_1)$ with $k = 1$ and $\ell = 0$ used above, $\mathrm{histogram}(\mathcal{G}, \varphi_{\text{dominates}})[s]$ is the number of dominating sets of size s in the graph $\mathcal{G}$. For an MSO-formula $\varphi_{\text{is-matching}}(Y_1)$ with $k = 0$ and $\ell = 1$ over the incidence representation of graphs that defines sets of edges that are matchings, $\mathrm{histogram}(\mathcal{G}, \varphi_{\text{is-matching}})$ is just a scalar value that tells us how many matchings $\mathcal{G}$ contains. For $k = \ell = 0$ we get Theorem 1.1 as a corollary from the following theorem; in this case the output is a scalar that is 1 if $\mathcal{A} \models \varphi$, and 0, otherwise. The DTM from the following theorem outputs $\mathrm{histogram}(\mathcal{A}, \varphi)$ in a linearized fashion as a string $\mathrm{str}(\mathrm{histogram}(\mathcal{A}, \varphi))$.

THEOREM 1.2 (Histogram Computation in Logarithmic Space). *For every $w \in \mathbb{N}$, and every* MSO*-formula $\varphi(X_1, \dots, X_k, Y_1, \dots, Y_\ell)$ over some vocabulary τ, there is a logspace* DTM *that, on input of a τ-structure $\mathcal{A}$ of tree width at most w, outputs* $\mathrm{str}(\mathrm{histogram}(\mathcal{A}, \varphi))$.

Applying this theorem to the formula $\varphi_{\text{is-matching}}$ shows that counting the number of matchings can be done in logspace for any class of graphs of bounded tree width. Moreover, the output of the theorem, which is a string that encodes a histogram, can be used in many ways by looking up individual bits in it. For example, it can be used to show that the optimization problem DOMINATING-SET lies in L when restricted to any class of graphs of bounded tree width: On input of a graph $\mathcal{G}$ of bounded tree width, we first compute $\text{histogram}(\mathcal{G}, \varphi_{\text{dominates}})$ and, then, output the smallest position s with $\text{histogram}(\mathcal{G}, \varphi_{\text{dominates}})[s] > 0$. Moreover, we can answer, in logspace, whether there is a dominating set of some given *cardinality* or, even, *count the number* of dominating sets *of a given cardinality.* In particular, the histogram theorem for logspace generalizes the findings of Jakoby and Tantau (2007) that the length of shortest and longest paths in tree width-2 graphs can be computed in logspace and the results of Das et al. (2010) that the same upper bound holds for these problems in directed acyclic orientations of k-trees.

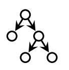
A newly developed technique that is crucial for proving the theorems related to tree-width-bounded structures and logspace is the computation of tree decompositions of bounded width using only logarithmic space in Chapter 6. This chapter also contains the proof of the following theorem, which can be used to remove the witness on the tree width bound from all theorems stated above, and test the tree width bound together with the MSO-defined property.

THEOREM 1.3 (Exact-Width Tree Decompositions in Logarithmic Space). *For every $w \in \mathbb{N}$, and vocabulary τ, there is a logspace* DTM *that, on input of a τ-structure $\mathcal{A}$, either (1) outputs a width-w tree decomposition of $\mathcal{A}$, or (2) outputs "no" and the tree width of $\mathcal{A}$ exceeds w in this case.*

The previous Theorem stands at the end of a line of work on detecting the complexity of constructing tree decompositions using the functional variants of P (Arnborg et al., 1987), NC (Bodlaender, 1989), and LOGCFL (Gottlob et al., 2002); it is tight in the sense that deciding whether a given graph has tree width at most w is L-complete for any width bound $w \in \mathbb{N}$ (Chapter 6).

Tree Decompositions as Terms and Logarithmic-Depth Circuits. The inputs of the theorems related to logspace are structures of bounded tree width. Many MSO-definable problems on such structures are complete for L like the reachability problem restricted to trees (Cook and McKenzie, 1987), which have tree width 1. Thus, if we want to state theorems that place MSO-definable problems on tree-width-bounded structures inside subclasses of L, which is what we do next, we either need to restrict the logic or the kinds of inputs allowed. We will consider the latter case. For logarithmic-depth circuit classes we allow structures of bounded tree width as inputs, but require that they are *accompanied by tree decompositions whose underlying trees are given in term representation.*

The inputs now consist of the string encoding of a logical structure $\mathcal{A}$ together with the encoding of a tree decomposition (T, B) of $\mathcal{A}$, where T is given in term representation; for example, a term representation for the tree right is [[ ] [[ ] [ ]]] (a detailed definition of tree decompositions in term representation is given in Section 5.1). Terms are a natural form of input in the study of logarithmic-depth circuits that make the L-complete task of computing the transitive closure of trees easier.



All circuit families in this thesis are DLOGTIME-uniform as defined by Mix Barrington, Immerman, and Straubing (1990), and discussed in Section 4.1.

THEOREM 1.4 (Decision Using Boolean Logarithmic-Depth Circuits). *For every $w \in \mathbb{N}$, and every* MSO*-formula φ over some vocabulary τ, there is a* DLOGTIME*-uniform* NC^1*-circuit family that, on input of a τ-structure $\mathcal{A}$ along with a width-w tree decomposition in term representation for $\mathcal{A}$, decides whether $\mathcal{A} \models \varphi$ holds.*

To state a histogram-based theorem for logarithmic depth, we use arithmetic circuits instead of Boolean circuits. The inputs of these circuits are (binary) strings, while their inner gates compute addition and multiplication operations, and the outputs are integers. In order to represent the output histogram h using a single number $\mathrm{num}(h) \in \mathbb{N}$, imagine h to be stored in computer memory with a word size large enough so that each of its entries fits into one memory cell (choosing the word size as a multiple of $|A|$, where A is the universe of the input structure $\mathcal{A}$, suffices). Then $\mathrm{num}(h)$ is the single number representing the whole of the memory contents (the formal definition of $\mathrm{num}(h)$ is given on page 56f.). Theorem 1.4 is a corollary of the following theorem since testing whether a function from $\#\mathrm{NC}^1$ outputs a value greater than 0 is an NC^1-computable property.

THEOREM 1.5 (Histogram Computation Using Arithmetic Logarithmic-Depth Circuits). *For every $w \in \mathbb{N}$, and every* MSO*-formula $\varphi(X_1, \dots, X_k, Y_1, \dots, Y_\ell)$ over some vocabulary τ, there is a* DLOGTIME*-uniform* $\#\mathrm{NC}^1$*-circuit family that, on input of a τ-structure $\mathcal{A}$ along with a width-w tree decomposition in term representation for $\mathcal{A}$, outputs* $\mathrm{num}(\mathrm{histogram}(\mathcal{A}, \varphi))$.

The theorems for logarithmic-depth circuits can be applied to solve problems whose inputs are already accompanied by tree decompositions of bounded width with underlying trees given in term representation, or from which it is easy to derive decompositions. Many *evaluation* and *simulation* tasks are of the later kind: evaluating Boolean and arithmetic sentences, as well as simulating the acceptance behavior and counting the number of accepting computations of visible pushdown automata. These applications are discussed in Section 5.3. Encoding the problems of evaluating Boolean and arithmetic sentences as MSO-defined decision and counting problems, respectively, proves that Theorems 1.4 and 1.5 cover problems that are complete for NC^1 and $\#\mathrm{NC}^1$, respectively. Previous NC^1 and $\#\mathrm{NC}^1$ approaches for these problems consisted of input balancing strategies that were deeply intertwined with the actual problem solving steps. The thesis shows how to divide these steps by first balancing the given tree decomposition and, then, solving the problem at hand along a logarithmic-depth decomposition; which results in simpler proofs, and more powerful and flexible theorems

Bounded Tree Depth and Constant-Depth Circuits. For *structures of bounded tree depth*, which admit tree decompositions of bounded width whose underlying trees have bounded depth, the complexity of MSO-definable problems drops even further to constant-depth circuit classes. In order to solve MSO-definable decision, counting, and optimization problems on tree-depth-bounded structures, we need to use different kinds of constant-depth circuits: Boolean circuits for solving decision problems, arithmetic circuits for computing number representations of histograms and, hence, solving counting problems, and Boolean circuits with threshold gates for computing string representations of histograms and, hence, solving optimization problems. Thus, depending on its type, a problem will either lie in AC^0, GapAC^0, or TC^0. We start with MSO-definable decision problems.

THEOREM 1.6 (Decision Using Boolean Constant-Depth Circuits). *For every $d \in \mathbb{N}$, and every* MSO*-formula φ over some vocabulary τ, there is a* DLOGTIME*-uniform* AC^0*-circuit family that, on input of a τ-structure $\mathcal{A}$ of tree depth at most d, decides $\mathcal{A} \models \varphi$.*

An example application of Theorem 1.6 is to put the MSO-definable decision problem of whether a graph of bounded tree depth has a perfect matching into AC^0. In contrast, in Section 6.4, we see that the same problem for graphs of bounded tree width is L-complete. In case of input structures of bounded tree depth, we can state the following theorem on counting the number of solutions with respect to their cardinalities.

THEOREM 1.7 (Histogram Computation Using Arithmetic Constant-Depth Circuits). *For every $d \in \mathbb{N}$, and every* MSO*-formula $\varphi(X_1, \dots, X_k, Y_1, \dots, Y_\ell)$ over some vocabulary τ, there is a* DLOGTIME*-uniform* GapAC^0*-circuit family that, on input of a τ-structure $\mathcal{A}$ of tree depth at most d, outputs the number* $\mathrm{num}(\mathrm{histogram}(\mathcal{A}, \varphi))$.

Applying Theorem 1.7 to the formula $\mathrm{histogram}(\mathcal{G}, \varphi_{\text{is-matching}})$ shows that counting the number of matchings lies in GapAC^0 on graphs of bounded tree depths.

Like in the case of logspace, the solution histogram subsumes many optimization problems that ask for the existence of a solution with a certain cardinality. For this, we need to look up individual bits in a string representation of the solution histogram and not only compute a number representation of it. Using the result of Hesse, Allender, and Mix Barrington (2002) that DLOGTIME-uniform GapAC^0-circuit families, which compute numbers, can be simulated by DLOGTIME-uniform FTC^0-circuit families that compute the binary string representations of these number, we get the following theorem as a corollary from Theorem 1.7.

THEOREM 1.8 (Histogram Computation Using Boolean Constant-Depth Circuits with Threshold Gates). *For every $d \in \mathbb{N}$, and every* MSO*-formula $\varphi(X_1, \dots, X_k, Y_1, \dots, Y_\ell)$ over some vocabulary τ, there is a* DLOGTIME*-uniform* FTC^0*-circuit family that, on input of a τ-structure $\mathcal{A}$ of tree depth at most d, outputs the string* $\mathrm{str}(\mathrm{histogram}(\mathcal{A}, \varphi))$.

We cannot hope to compute string representations of histograms in any complexity class smaller than FTC^0 since the TC^0-complete problem MAJORITY is expressible using an MSO-formula and the histogram: Turning a string $w = w_1, \dots, w_n \in \{0,1\}^*$ into a logical structure $\mathcal{A} = (\{1, \dots, n\}, P_1^{\mathcal{A}})$ in the usual manner by setting $i \in P_1^{\mathcal{A}} \Leftrightarrow w_i = 1$, for the MSO-formula $\varphi(X_1) = \forall x(X_1(x) \to P_1(x))$ we have $\mathrm{histogram}(\mathcal{A}, \varphi)\big[\lfloor n/2 \rfloor + 1\big] > 0$ if, and only if, more than half of the input bits are 1. Like the histogram theorem for logspace, the histogram theorem for FTC^0 can be used to solve a wide range of problems that where not known to lie in DLOGTIME-uniform TC^0 before: In Section 4.4, Theorem 1.8 is applied to show that the SUBSETSUM problem with input numbers that are encoded in unary lies in DLOGTIME-uniform TC^0. Moreover, the same upper bound holds for the problem of solving linear equation systems with any constant number of equations whose coefficients are integer values that are encoded in unary.

The proofs of the theorems for constant-depth circuits rest on the development of a tree automaton model for degree-unbounded labeled trees and an algebraic representation of the automata's computations using arithmetic circuits.

Bounded Tree Depth and First-Order Logic. Mix Barrington, Immerman, and Straubing (1990) characterize DLOGTIME-uniform AC^0 in terms of first-order logic with build-in order and certain arithmetic predicates and, thus, Theorem 1.6 implies that MSO-definable decision problems on structures of bounded tree depth can be solved using this logical formalism. The next theorem shows that turning MSO-formulæ into first-order formulæ is possible even without accessing any ordering or arithmetic predicates. Since first-order logic is a fragment of MSO-logic, this shows that MSO- and first-order logic have the same expressive power on any class of structures of bounded tree depth. With this result, Theorem 1.6 follows from Theorem 1.9 as a corollary by using the alternative definition of DLOGTIME-uniform circuits based on first-order formulæ and the fact that a structure can be extracted from its string encoding using such formulæ.

THEOREM 1.9 (First-Order Definability). *For every tree depth bound $d \in \mathbb{N}$, and every* MSO*-formula φ over some vocabulary τ, there is a first-order formula ψ over τ, such that for all τ-structure $\mathcal{A}$ of tree depth at most d, we have $\mathcal{A} \models \varphi$ if, and only if, $\mathcal{A} \models \psi$.*

Theorem 1.9 still holds if we additionally test whether the tree depth of the given structure is, indeed, d since the property of whether a structure has tree depth d is first-order definable for any $d \in \mathbb{N}$. Moreover, in Section 3.4 it is shown that Theorem 1.9 still holds when MSO-logic is replaced by the more general guarded second-order logic, which also allows quantifying over sets of tuples that are present in the input structure; like, for example, subsets of edges in a graph. For turning MSO-formulæ into first-order formulæ, the main technical development is a new Feferman–Vaught-type theorem that is constructive and works for an unbounded number of partitions.

1.3. Technical Contributions

The present section discusses the overall structure of the proofs of the theorems and highlights the main technical developments of the thesis.

The proofs of the theorems exploit in either an explicit or implicit way the fact that the given structures admit tree decompositions of bounded width. Explicitly means that the first step in the proof is the construction or adjustment of a tree decomposition. For such proofs, on input of a tree-decomposable structure $\mathcal{A}$,

(1) the first step is to compute a tree decomposition for $\mathcal{A}$, and

(2) the second step is to solve the problem using the tree decomposition as a guide of how to process the input.

A Constructive Feferman–Vaught-type Theorem for Unbounded Partitions. The proof of Theorem 1.9 avoids the constructing of a decomposition for the input structure altogether. Instead, first-order formulæ that define tree-depth-bounded structures are developed and extended to not only check whether a structure's tree depth is bounded by some constant, but also evaluate the MSO-formula at the same time. For this, the first-order formula recursively computes the set of MSO-formulæ that are true in substructures of the input structure and combines this information to compute the set of MSO-formulæ that are true in the whole structure; this information is known as the *type* of a structure. The task of combining the information is done by a new variant of the type composition theorem of Feferman and Vaught (1959) that shows how to *construct* the type of a structure based on the types of an *unbounded* number of substructures that have a *constant-size overlap.* In the case of first-order formulæ and structures of bounded tree depth, this theorem is the basic building block to process the input structure in order to evaluate the MSO-formula.

Automata for Trees of Unbounded Degree and Their Arithmetic-Circuit-Based Evaluation. Since Theorem 1.6 follows from Theorem 1.9 using the definition of DLOGTIME-uniform AC^0 in terms of first-order formulæ we also do not need to construct tree decompositions for its proof. In contrast, for the proof of Theorem 1.7, tree decompositions of bounded width whose underlying trees have bounded depth are constructed. Once such tree decompositions are available, the proof of Theorem 1.7 proceeds to evaluate the MSO-formula from the problem definition along the tree decomposition. This is done by refining proof step (2) into

(2.1) first transforming the formulæ on the input structure into equivalent tree automata on the tree decomposition,

(2.2) and, then, evaluating tree automata using arithmetic circuits.

This basic structure is often used to prove Courcelle's Theorem and its variants, but for implementing it using constant-depth circuits, new techniques needed to be developed. In the proof of Theorem 1.2 of Elberfeld, Jakoby, and Tantau (2010), the trees underlying the involved tree decompositions have bounded degree and this is used to translate MSO-formulæ into the classical tree automata of Doner (1970) and Thatcher and Wright (1968). Evaluating automata is then done by constructing arithmetic formulæ that are evaluated in logspace. The nodes of the depth-bounded trees underlying the tree decompositions that are involved in the proof of Theorem 1.7 necessarily have an unbounded degree since, otherwise, they would only be able to cover constant-size structures. The technically involved step in proving Theorem 1.7 is the development of just the right variant of an automaton model for unbounded-degree labeled trees that can simulate MSO-formulæ and whose computations can be represented algebraically using arithmetic circuits. Since the automaton model and the translation between MSO-formulæ and automata also serve as building blocks in the proof of Theorem 1.9, the automaton model is defined in Chapter 2 before proving the Theorems 1.9 and 1.7 in Chapters 3 and 4, respectively.

Balancing Tree Decompositions Using Constant-Depth Circuits. For proving the theorems related to logarithmic-depth circuits, we do not need to construct tree decompositions since they are already part of the input, but for proving the theorems we run across a different problem: The tree we want to process may have a linear depth and, thus, may contain data dependencies whose ad hoc resolution would need the same linear circuit depth. While we are lucky in the case of Theorem 1.4 where we can translate the MSO-formula that defines our problem into a tree automaton and plug in a result from the literature that shows that the acceptance behaviour of tree automata can be simulated using logarithmic-depth Boolean circuits (Lohrey, 2001), it is less obvious to prove Theorem 1.5. For the proof of this theorem, we proceed as follows: Before translating the formula on the structure into a tree automaton, we insert an extra layer into the proof that transforms the given tree decomposition into a tree decomposition whose underlying tree is binary and balanced and, thus, has logarithmic depth. Interestingly, for this step we do not even need logarithmic-depth circuits; constant-depth Boolean circuits with threshold gates are enough to balance the tree decomposition. Evaluating tree automata can then be done with logarithmic-depth circuits for the resulting logarithmic-depth decomposition.

Computing Tree Decompositions in Logarithmic Space. After proving the theorems related to logarithmic-depth circuits, the main task for proving the theorems related to tree-width-bounded structures and logspace, for which no tree decompositions are given in the input, is to compute a tree decomposition and output the underlying tree in term representation. Then the theorems for logarithmic-depth circuits are plugged in to complete the proof. While, for proving the Theorems 1.1 and 1.2, computing tree decompositions of some approximate constant width would be enough, the developed techniques also lead to the logspace construction of tree decompositions of exact constant width.

1.4. Organization of This Dissertation

The main part of the thesis is made up by five chapters that are organized according to how the theorems from the introduction are proved. Each chapter presents one of the techniques described above, uses it to prove the related theorems, and discusses their applications.

Chapter 2 contains the definition of the new notion of multiset tree automata and its equivalence to MSO-definable properties on tree structures. It is also shown how to move from MSO- to first-order formulæ in case of depth-bounded trees.

Chapter 3 picks up the transformation from MSO- to first-order formulæ from Chapter 2 and generalizes it from trees of bounded depth to logical structures of bounded tree depth. The development of a new Feferman–Vaught-type theorem for unbounded partitions and its combination with the recursive definition of tree depth culminates in a proof of Theorem 1.9—how to move from MSO- to equivalent first-order formulæ in case of structures of bounded tree depth.

In Chapter 4 we begin to consider computations based on shallow circuits. We start to review the definition of uniform circuits and their equivalent definition in terms of first-order formulæ that describe computational problems. In conjunction with the equivalence of the expressive power of MSO- and first-order logic in case of tree-depth-bounded structures proved in Chapter 3, we get Theorem 1.6—MSO-definable decision problems lie in AC^0 for classes of structures whose tree depth is bounded. Next, after defining the notion of tree decompositions, the main part of Chapter 4 is devoted to proving how computations of multiset tree automata can be represented algebraically using arithmetic circuits. This proves Theorem 1.7—computing number representations of histograms in GapAC^0—and Theorem 1.8—computing string representations of histograms in FTC^0. The final proof steps are given on page 58.

With Chapter 5 we move our focus from constant-depth circuits that work on tree-depth-bounded structures to logarithmic-depth circuits that work on tree-width-bounded structures that are accompanied with tree decompositions in term representation. After discussing the details of term representations, we show how to balance tree decompositions using constant-depth Boolean circuits with threshold gates. Once balanced tree decompositions, which always have logarithmic depth, are available, we can plug in the generic arithmetic circuit construction from Chapter 4 to prove Theorem 1.5 for $\#\mathrm{NC}^1$ on page 69. As discussed above, this implies Theorem 1.4 for NC^1 as a corollary.

In Chapter 6, we consider the case that tree decompositions of bounded width are not part of the input, but need to be computed. We show how to do this using logspace DTMs. Since logspace DTMs can simulate logarithmic-depth Boolean and arithmetic circuits and, hence, are able to solve all problems from Chapter 5, Theorem 1.2 follows once we know how to construct tree decompositions and output them in term representation (the final steps for proving this theorem are given on page 82f.).

At the end of each chapter, except Chapter 2, applications of the developed theorems and proofs are discussed.

The thesis concludes with reviewing the developed techniques, theorems, and their applications. Moreover, future research directions related to the presented results are discussed.

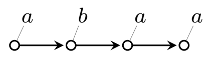
CHAPTER 2

Unbounded Degree Trees and Automata

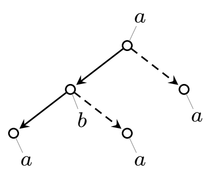
Courcelle's Theorem is commonly proved by translating an MSO-formula on a structure into an equivalent automaton that works on a tree. Then, instead of evaluating the formula for the structure, the automaton's computation on the tree is simulated. We will follow the same basic approach in the proofs of the theorems from the introduction, but define a new automaton notion to work with the trees that arise in the context of tree-depth-bounded structures in Chapters 3 and 4. The trees that appear inside the proofs of these chapters have a bounded depth and, to cover structures of any size, necessarily have an unbounded node degree. Thus, classical automata for bounded-degree trees cannot be used in this context. The present chapter is devoted to discuss the need for a new automaton model, its definition, and equivalence to monadic second-order formulæ on trees.

The technique of translating between MSO-formulæ and equivalent automata has a long history. One of the first results is a theorem of Büchi (1960), Elgot (1961), and Trakhtenbrot (1961) showing that the languages accepted by finite automata are exactly the MSO-definable sets of strings. To establish a connection between the language-theoretic concept of strings and relational structures from mathematical logic, formal statements of this result use back and forth translations between strings and equivalent path structures in which monadic relations indicate the symbols that are present in the string.

For example, the string *abaa* is translated into the structure $\mathcal{A}$ with universe $A = \{1, 2, 3, 4\}$, containing one element for each position of the string, successor relation $E^{\mathcal{A}} = \{(1, 2), (2, 3), (3, 4)\}$, encoding the order of positions in the string, and the monadic relations $R_a^{\mathcal{A}} = \{1, 3, 4\}$ and $R_b^{\mathcal{A}} = \{2\}$, indicating which symbols are present at the different positions of the string; a convenient way to represent this structure is to draw it like the path on the right where the edges represent the successor relation and the labels indicate which monadic relations hold. An equivalent way of bridging the world of automata with the world of logic is to directly define the automaton on a labeled path (or tree), which is what we later do in the present chapter.



The Büchi–Elgot–Trakhtenbrot Theorem was extended to labeled trees of bounded degree by Doner (1970), and Thatcher and Wright (1968). In this case, the considered structures not only contain one, but a constant number of successor relations $E_1^{\mathcal{A}}, \dots, E_d^{\mathcal{A}}$ whose number equals a degree bound d as well as a monadic relation $R_\sigma^{\mathcal{A}}$ for each symbol σ from the considered alphabet Σ. An example of such a structure is shown on the right for $d = 2$ where the relation $E_1^{\mathcal{A}}$ is indicated using solid and the relation $E_2^{\mathcal{A}}$ is indicated using dashed edges. Doner, and Thatcher and Wright

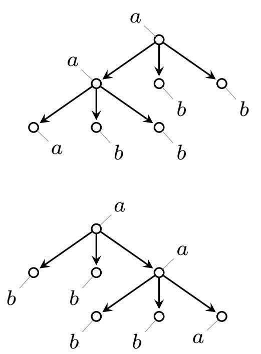


translate MSO-formulæ on degree-d trees into tree automata whose transition functions $\delta\colon \Sigma \times Q^d \to Q$ assign a state to a node based on its label from Σ and the (implicitly ordered) sequence of states from Q^d that are assigned to its children. The finding of Büchi, Elgot, and Trakhtenbrot follows from this result for degree $d = 1$.

Beside the classical application to the *satisfiability problem* for MSO-formulæ in case of tree structures, a recent use of Büchi–Elgot–Trakhtenbrot-type theorems is in the area of finding efficient algorithms for *evaluating* an MSO-formula for a given tree. To solve this problem one constructs an automaton that is equivalent to the considered MSO-formula and, then, evaluates the automaton. This approach can be generalized to MSO-formulæ on structures of bounded tree width where one first turns a tree-width-bounded structures and an MSO-formula on it into an equivalent tree of bounded degree and an MSO-formula on the tree. Then the formula is evaluated using an equivalent automaton for bounded degree trees. The proofs in the present thesis follow the same idea, but, in case of tree-depth-bounded structures, cannot be based on the classical tree automata for bounded-degree trees since the trees that appear in the proofs of these results have an unbounded number of children. Moreover, they do not impose an order on sibling nodes. Such trees are called unranked unordered in the literature, where *unranked* means that nodes have an unbounded degree (in opposition to ranked trees whose node-degree is bounded by some constant) and *unordered* means that there is no total order on sibling nodes. The automaton notion we define for this setting and its equivalence to MSO-formulæ is tailored to serve two tasks in later chapters:

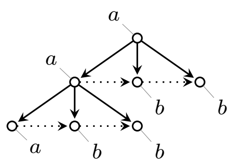
Task 1: The transition function of the automaton is definable in first-order logic. That means, whether some state q of the automaton can be assigned to a node n of the tree can be decided by a first-order formula $\varphi_q(n)$ that has access to the states assigned to the children of n. This feature of multiset tree automata is used in Chapter 3 to prove Theorem 1.9—first-order and MSO-logic have the same expressive power on tree-depth-bounded structures.

Task 2: The transformation from MSO-formulæ to automata preserves the sizes and number of solutions and, thereby, establishes a reduction from counting the number of solutions to an MSO-formula to evaluating an automaton. Moreover, the specific form of the automaton's transition function allows to implement its evaluation using arithmetic circuits in Chapter 4.



Towards an automaton notion for task 1. The theorems that handle structures of bounded tree depth cover, as one of the simplest example, trees that have bounded depth, but an unbounded degree. Adopting the classical strategy for designing time-efficient algorithms, which means in this case transforming trees of unbounded degree into trees of bounded degree, would come at the cost of increasing the depth of the tree by at least a logarithmic factor and this would imply vertical data dependencies in the tree that we cannot hope to handle by first-order formulæ in Chapter 3 or constant-depth circuits in Chapter 4. The automaton notion from this chapter helps to sidestep this problem since it directly works on unbounded-degree trees and, thus, does not force us to change the topology of the tree. The successor relation in each of the considered trees is encoded using a single binary relation $E^{\mathcal{A}}$ and, thus, sibling nodes are not distinguished by multiple edge relations. Such trees can be seen has having only a single type of edges and having unordered children that are not distinguishable solely based on successor relations. As a result, there are multiple ways of drawing the same tree as shown in the example above.

For a similar reason, we are not able to use automata notions that are developed (Brüggemann-Klein et al., 2001) and used (Gottlob et al., 2005) in the context of XML processing. Such automata work on trees where every node may have an unbounded number of children, but whose sibling nodes are totally ordered in an implicit way using an order relation on sibling nodes, as indicated by the dotted lines in the tree right. This perfectly fits the needs of XML processing where documents not only encode a hierarchical bracket structure, but also a total order on the children of each node by the text file encoding in an implicit way. Without further restricting this automaton model we are not able to use it for our applications since the horizontal data dependencies on sibling nodes are too high. In fact, MSO-formulæ and the corresponding automata on this structures are able to define any regular property on the ordered children of a node and, thus, cannot be simulated by constant-depth circuits, let alone first-order formulæ. For similar reasons our automaton notion needs to be different from the order-invariant tree automata of Courcelle (1989), which, for example, can answer the question of whether the number of sibling nodes that satisfy a certain property is even.

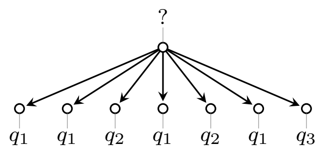


Towards an automaton notion for task 2. An automaton notion from the literature that is equivalent to our model are the *counting unranked tree automata* of Libkin (2006). Transitions of these automata are defined in terms of Boolean functions: They allow us to assign a state q to a node with symbol σ if a Boolean function $\delta(\sigma, q)$, which depends on the number of occurrences of states at the children, evaluates to 1. Another equivalent automaton notion would result from using hedge automata (Brüggemann-Klein et al., 2001) whose horizontal languages are recognized by finite monoids that are commutative and aperiodic (Straubing, 1994). The automaton notion that is presented in this chapter makes it a relatively easy task to solve MSO-definable counting problems using arithmetic circuits since it allows to relate the states that are assigned to the children of a node with the state that is assigned to the node itself in a transparent way, without hiding the transition relation inside a Boolean function or a monoid.

The present chapter starts with the definition of multiset tree automata and proofs of their closure properties in Section 2.1. Then monadic second-order logic is formally defined in Section 2.2 and its equivalence to multiset tree automata on unordered labeled trees is shown in Section 2.3.

2.1. Definition and Closure Properties of Multiset Tree Automata

The basic idea behind the definition of a multiset tree automaton is as follows: In order to determine the state reached for a given node of the tree, we consider the states reached at the children of the node. Since the trees under consideration are *unordered,* we need to consider the *set* of states reached rather than the *sequence* of states reached at the children. However, just considering the set of states reached turns out to be insufficient: For our proofs we need to be able to distinguish whether, say, the state q_1 is reached twice and the state q_2 once or whether the cardinalities are the other way round. For this reason, we consider the *multiset* of states reached at the children of a given node. However, our trees are also *unranked* and as the number of children grows, the number of possible state multisets that the automaton may encounter grows without limit. In order to use finite descriptions of the automata nevertheless, we introduce a *capping* operation: The automaton's transition relation is defined only for state multisets with some maximal multiplicity $m \in \mathbb{N}$ and whenever a state is reached at more than m children, the state is inserted into the state multiset only m times (multiplicities are capped at m).



As an example, suppose an automaton reaches the indicated states at the children of the root like in the example on the right. Then the multiset of states reached is $M = \{q_1, q_1, q_1, q_1, q_2, q_2, q_3\}$. If the multiplicity of the automaton is, say, 2, then the capped multiset is $M|_2 = \{q_1, q_1, q_2, q_2, q_3\}$. The state reached at the root is then determined by the entry of the transition relation for this particular multiset. In the following, these ideas are formalized.

Definition of Multiset Tree Automata. Multisets generalize sets by allowing elements to appear more than once. Formally, given a universe U (which is just a normal set), a *multiset M on U* is a function $\#_M : U \to \mathbb{N}$ that assigns a *multiplicity* to each element of U. We say that M *has multiplicity at most m* if $\#_M(e) \leq m$ holds for all $e \in U$. The *cardinality of M is* $|M| := \sum_{e \in U} \#_M(e)$. We write $\mathcal{P}_\omega(U)$ for the class of all multisets on U and we write $\mathcal{P}_m(U)$ for the class of all multisets on U of multiplicity at most m. Usual sets can be considered as multisets M with multiplicity at most 1 and $\mathcal{P}(U) := \mathcal{P}_1(U)$ is the usual power set of U. Given two multisets M and N over the same universe U, we write $M \subseteq N$ to indicate that $\#_M(e) \leq \#_N(e)$ holds for all $e \in U$. The *union* $M \cup N$ is the multiset with $\#_{M \cup N}(e) = \max\{\#_M(e), \#_N(e)\}$ for all $e \in U$. Similarly, the *intersection* $M \cap N$ has the property $\#_{M \cap N}(e) = \min\{\#_M(e), \#_N(e)\}$ and the *set difference* $M \setminus N$ has the property $\#_{M \setminus N}(e) = \max\{0, \#_M(e) - \#_N(e)\}$. We define two restriction operations on multisets $M \in \mathcal{P}_\omega(U)$. First, given a number $m \in \mathbb{N}$, let $M|_m$ be $\#_{M|_m}(e) := \min\{\#_M(e), m\}$ for $e \in U$. We call $M|_m$ the *capped version of M to multiplicity m.* Second, for a set $V \subseteq U$, let us write $M|_V$ for the *restriction of M to V*, defined by $\#_{M|_V}(e) := \#_M(e)$ for $e \in V$ and $\#_{M|_V}(e) := 0$ for $e \notin V$. Next, we formally define the notion of multiset tree automata and how they process labeled trees.

Definition 2.1 (Multiset Tree Automaton). A *nondeterministic (bottom-up) multiset tree automaton* is a tuple $A = (\Sigma, Q, Q_a, \Delta)$ consisting of an *alphabet* Σ, a *state set* Q, a set $Q_a \subseteq Q$ of *accepting states*, and a *state transition relation*

$$\Delta \subseteq \Sigma \times \mathcal{P}_m(Q) \times Q$$

for some constant $m \in \mathbb{N}$, the *multiplicity bound of A*. The automaton is *deterministic* if for every $\sigma \in \Sigma$ and every $M \in \mathcal{P}_m(Q)$ there is exactly one $q \in Q$ with $(\sigma, M, q) \in \Delta$; in this case we can view Δ as a *state transition function*

$$\delta \colon \Sigma \times \mathcal{P}_m(Q) \to Q \ .$$

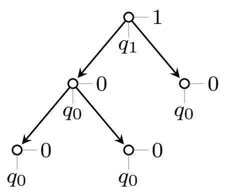
Definition 2.2 (Graph). A *(directed) graph G* is a tuple (V, E) consisting of a *vertex set V* and a *set of edges $E \subseteq V \times V$*, which we also denote by $V(G)$ and $E(G)$, respectively. A vertex $c \in V(G)$ is a *child* of another vertex $n \in V(G)$ in G if $(n, c) \in E(G)$ holds.

Definition 2.3 (Tree). A *tree* is a directed graph T that contains a distinguished *root* $r \in V(T)$, such that for every $v \in V(T)$ there exists exactly one path from r to v. We use the term *nodes* to refer to the vertices of a tree. The *leaves* of a tree are its nodes without children; all other nodes are *inner nodes.*

Definition 2.4 (Labeled Tree). Let Σ be an alphabet. A *labeled tree over* Σ is a pair (T, l) consisting of a tree T and a mapping $l \colon V(T) \to \Sigma$.

Definition 2.5 (Computation of a Multiset Tree Automaton). Let $A = (\Sigma, Q, Q_a, \Delta)$ be a multiset tree automaton and (T, l) be a labeled tree over Σ. A *computation* of A on (T, l) is a mapping $q \colon V(T) \to Q$, such that for every node $n \in V(T)$ we have $(l(n), M|_m, q(n)) \in \Delta$, where M is the multiset $\{q(c) \mid c$ is a child of n in $T\}$. A computation is *accepting*, if $q(r) \in Q_a$ holds for the root r of T. The *tree language* $L(A)$ contains all labeled trees over Σ for which there exists an accepting computation of A on (T, l).

Example 2.6 (A Multiset Tree Automaton for All-0 Labeled Trees). A multiset tree automaton that accepts exactly the labeled trees (T, l) over $\Sigma = \{0, 1\}$ where $l(n) = 0$ holds for each node $n \in V(T)$ is the following automaton (Σ, Q, Q_a, δ) with multiplicity bound $m = 1$: The automaton has the state set $Q := \{q_0, q_1\}$, where q_0 stands for the fact that the already processed part of the tree is only labeled by 0s, and q_1 stands for the opposite fact, that there is at least one node labeled by 1. Since we want to test whether the former property holds, we set $Q_a := \{q_0\}$. Let $\sigma \in \{0, 1\}$ and $M \subseteq Q$. For the transition function we set $\delta(\sigma, M) := q_1$ if $\sigma = 1$ or $q_1 \in M$ and $\delta(\sigma, M) := q_0$, otherwise. The example on the right shows a tree whose nodes are labeled by symbols from $\{0, 1\}$ and states that are assigned to them by the defined automaton. The whole tree is not accepted, while every proper subtree is accepted.



Closure Properties of Multiset Tree Automata. We now show that the class of tree languages accepted by multiset tree automata is closed under intersection, complement, and for every nondeterministic multiset tree automaton there is a deterministic automaton accepting the same tree language. These closure properties are crucial for the transformation of MSO-formulæ into multiset tree automata.

Lemma 2.7. *For all multiset tree automata A and B there is a multiset tree automaton C with $L(C) = L(A) \cap L(B)$.*

Proof. For this proof, we introduce the following operations on multisets: For universes U_1 and U_2, the *projections* $\pi_1\colon \mathcal{P}_\omega(U_1 \times U_2) \to \mathcal{P}_\omega(U_1)$ and $\pi_2\colon \mathcal{P}_\omega(U_1 \times U_2) \to \mathcal{P}_\omega(U_2)$ of multisets of pairs to their first and second components are defined as $\#_{\pi_1(M)}(e) := \sum_{f\in U_2} \#_M((e,f))$ and $\#_{\pi_2(M)}(f) := \sum_{e\in U_1} \#_M((e,f))$ for any $M \in P_\omega(U_1 \times U_2)$.

In the proof we use that for any multiset M and bounds $m, n, k \in \mathbb{N}$ with $m, n \leq k$ the equations

$$\begin{aligned} \pi_1(M)|_m &= \pi_1(M|_k)|_m \text{ and} \\ \pi_2(M)|_n &= \pi_2(M|_k)|_n \text{ hold.} \end{aligned} \tag{$*$}$$

Let $A = (\Sigma, Q, Q_a, \Delta_A)$ and $B = (\Sigma, P, P_a, \Delta_B)$ be multiset tree automata with multiplicity bounds m and n, respectively. The *intersection product automaton* of A and B is $C := (\Sigma, Q \times P, Q_a \times P_a, \Delta_C)$ with multiplicity bound $k := \max\{m, n\}$ and transition relation

$$\begin{aligned} \Delta_C := \big\{\big(\sigma, N, (q,p)\big) \mid \sigma \in \Sigma,\ N \in \mathcal{P}_k(Q \times P),\ q \in Q,\ p \in P, \\ (\sigma, \pi_1(N)|_m, q) \in \Delta_A,\ (\sigma, \pi_2(N)|_n, p) \in \Delta_B\big\}. \end{aligned}$$

We show by induction on the depth of the tree that the following is true for all labeled trees: The automata A and B can reach states q and p at the root, respectively, if, and only if, C can reach the state (q, p) at the root. Clearly, this implies the claim. Since there is nothing to prove for empty trees, we only need to prove the inductive step. Let c_1 to c_t be the children of the root. We need to prove two directions.

For the only-if-direction, let q_1 to q_t be states reached by A at the children c_1 to c_t, respectively, such that $(\sigma, \{q_1, \dots, q_t\}|_m, q) \in \Delta_A$ and let p_1 to p_t be states reached by B at these children, such that $(\sigma, \{p_1, \dots, p_t\}|_n, p) \in \Delta_B$. By the induction hypothesis, C can reach (q_1, p_1) to (q_t, p_t) at c_1 to c_t, respectively; let M denote the multiset of these state pairs. Since $m, n \leq k$, property $(*)$ implies the second equality in both

$$\begin{aligned} \{q_1, \dots, q_t\}|_m &= \pi_1(M)|_m = \pi_1(M|_k)|_m \text{ and} \\ \{p_1, \dots, p_t\}|_n &= \pi_2(M)|_n = \pi_2(M|_k)|_n\ . \end{aligned}$$

The definition of C immediately implies $(\sigma, N, (q, p)) \in \Delta_C$ for $N = M|_k$ and, thus, there exists a computation of C that assigns (q, p) to the root.

For the if-direction, let C reach (q_1, p_1) to (q_t, p_t) at the children and let $\big(\sigma, N, (q,p)\big) \in \Delta_C$ for $N = \{(q_1, p_1), \dots, (q_t, p_t)\}|_k$. By the induction hypothesis, A can reach the states q_1 to q_t at the children and B can reach p_1 to p_t. By the definition of Δ_C, we have $(\sigma, \pi_1(N)|_m, q) \in \Delta_A$ and $(\sigma, \pi_2(N)|_n, p) \in \Delta_B$. Since $(*)$ implies both

$$\begin{aligned} \pi_1(N)|_m &= \{q_1, \dots, q_t\}|_m \text{ and} \\ \pi_2(N)|_n &= \{p_1, \dots, p_t\}|_n\ , \end{aligned}$$

we have $(\sigma, \{q_1, \dots, q_t\}|_m, q) \in \Delta_A$ and $(\sigma, \{p_1, \dots, p_t\}|_n, p) \in \Delta_B$. Thus q and p are reachable by A and B, respectively, at the root. □

Lemma 2.8. *For every nondeterministic multiset tree automaton A there is a deterministic multiset tree automaton B with $L(A) = L(B)$.*

Proof. For this proof, we define the *choice relation* $\iota \subseteq \mathcal{P}_\omega(U) \times \mathcal{P}_\omega(\mathcal{P}(U))$ as follows:

$$(\{v_1, \dots, v_t\}, \{V_1, \dots, V_t\}) \in \iota \text{ whenever } v_i \in V_i \text{ holds for all } i \in \{1, \dots, t\} \ .$$

A key observation for the following construction is that for every m and every $W \in \mathcal{P}_\omega(\mathcal{P}(U))$, the following set (not multiset) equality holds:

$$\big\{V|_m \bigm| (V, W) \in \iota\big\} = \big\{V|_m \bigm| (V, W|_{m\cdot|U|}) \in \iota\big\}. \qquad (**)$$

To see that this holds, first let $(V, W) \in \iota$. We must find a multiset V' with $V'|_m = V|_m$ and $(V', W|_{m\cdot|U|}) \in \iota$. To this end, we construct a sequence of pairs (V_i, W_i), such that for all i we have

(1) $(V_i, W_i) \in \iota$,
(2) $V_i|_m = V|_m$, and
(3) $W_i|_{m\cdot|U|} = W|_{m\cdot|U|}$.

Setting $V_1 := V$ and $W_1 := W$, the properties clearly hold for $i = 1$. Each pair is obtained from the previous pair as follows, when possible: Choose an element $v \in V_i$ whose multiplicity in V_i exceeds m and that is an element of a set $X \in W_i$ of multiplicity exceeding $m \cdot |U|$ in W_i. Let $V_{i+1} := V_i \setminus \{v\}$ and $W_{i+1} := W_i \setminus \{X\}$. Clearly, each new pair still has the three properties. Furthermore, this process will not stop as long as W_i still contains an element X of multiplicity exceeding $m \cdot |U|$ since, then, at least one element of X must be present more than m times in V_i. This implies that for the last sets V' and W' constructed in this way we have $V'|_m = V|_m$ and $W' = W|_{m\cdot|U|}$ and by (a) this means $(V', W|_{m\cdot|U|}) \in \iota$ as claimed.

Second, let $(V, W|_{m\cdot|U|}) \in \iota$. We must construct a set V' with $V'|_m = V|_m$ and $(V', W) \in \iota$. Again, we construct a sequence of pairs satisfying the same properties as above, but start with $V_1 := V$ and $W_1 := W|_{m\cdot|U|}$. We choose an element v in V_i of multiplicity at least m and a set X in W_i with $v \in X$ of multiplicity is at least $m\cdot|U|$, but still less than $\#_W(X)$. We set $V_{i+1} := V \cup \{v\}$ and $W_{i+1} := W_i \cup \{X\}$. The three properties are still met by this construction. We claim that for the last sets V' and W' we have $W_i = W$. To see this, note that elements of W of multiplicity less than $m \cdot |U|$ have this multiplicity in all W_i and that for elements X of W of multiplicity at least $m \cdot |U|$ at least one element $v \in X$ must be present at least m times already in V.

We are now ready to construct the automaton B. Let $A = (\Sigma, Q, Q_a, \Delta)$ be a nondeterministic multiset tree automaton with multiplicity bound m. We call the elements of $P = \mathcal{P}_1(Q)$ *power states.* Let $B = (\Sigma, P, \{Q' \in P \mid Q' \cap Q_a \neq \emptyset\}, \delta)$ with multiplicity bound $k = m \cdot |Q|$ and transition function $\delta\colon \Sigma \times \mathcal{P}_k(P) \to P$ defined by

$$\delta(\sigma, W) := \{q \in Q \mid \exists\, V \in \mathcal{P}_k(Q) \text{ with } (V, W) \in \iota \text{ and } (\sigma, V|_m, q) \in \Delta\} \ .$$

We prove by induction on the depth of the tree that for every labeled tree the set S of states that the automaton A can reach at the root is exactly the power state S reached by B at the root on input of this tree. Given a tree whose root has children c_1 to c_t, assume that the set of states reached by A at child c_i is S_i for each i. By the induction hypothesis, B will reach the single power state S_i at child c_i. Let $W := \{S_1, \dots, S_t\}$. Now, by definition of the computation of multiset tree automata, the set S of states reached by A at the root is $\{q \in Q \mid \exists\, V \in \mathcal{P}_\omega(Q) \text{ with } (V, W) \in \iota \text{ and } (\sigma, V|_m, q) \in \Delta\}$. By equation ($**$), we can write S also as $\{q \in Q \mid \exists V \in \mathcal{P}_k(Q) \text{ with } (V, W|_k) \in \iota \text{ and } (\sigma, V|_m, q) \in \Delta\}$, which is exactly $\delta(\sigma, W|_k)$. □

LEMMA 2.9. *For every multiset tree automaton A there is a multiset tree automaton B accepting the complement of $L(A)$.*

PROOF. Make A deterministic, if necessary, using Lemma 2.8 and exchange accepting and rejecting states. □

2.2. Review of Monadic Second-Order Logic

In the present section we formally define MSO-logic and review a technique on how to eliminate first-order variables from MSO-formulæ.

Vocabularies and Structures. We define the notion of vocabularies with a finite set of symbols and structures with a finite universe as used, for example, by Libkin (2006).

DEFINITION 2.10 (Vocabulary). A *(relational) vocabulary* τ is a finite set of *relation symbols* together with a mapping ar that assigns an *arity* $\mathrm{ar}(R) \geq 1$ to each relation symbol R.

In slight abuse of notation, we write $R \in \tau$ to indicate that R is part of τ and $R^r \in \tau$ to additionally indicate that the arity of R in τ is r.

DEFINITION 2.11 (Structure). Let τ be a vocabulary. A *(finite) structure over τ*, or *τ-structure*, $\mathcal{A}$ consists of a nonempty, finite set A, the *universe* of $\mathcal{A}$, and for each relation symbol $R^r \in \tau$ a *relation* $R^{\mathcal{A}} \subseteq A^r$.

Like in the previous definition, we will often use the same letter for a structure and its universe, but a calligraphic font for the structure and a non-calligraphic font for the universe. Thus, the universes of the structures $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ are denoted by A, B, and C, respectively. A relation $R^{\mathcal{A}}$ and its symbol R are *monadic* if $\mathrm{ar}(R) = 1$.

Let $\mathcal{A}$ and $\mathcal{B}$ be two structures over the same vocabulary τ. The *union* of $\mathcal{A}$ and $\mathcal{B}$ is the τ-structure $\mathcal{A} \cup \mathcal{B}$ with universe $A \cup B$ and for every $R \in \tau$, we have $R^{\mathcal{A} \cup \mathcal{A}} = R^{\mathcal{A}} \cup R^{\mathcal{B}}$. We say that $\mathcal{B}$ is a *substructure* of $\mathcal{A}$ if $B \subseteq A$ and $R^{\mathcal{B}} \subseteq R^{\mathcal{A}}$ holds for every $R \in \tau$. For a subset $B \subseteq A$, the substructure of $\mathcal{A}$ that is *induced on B* has universe B and for all $R^r \in \tau$ we have $R^{\mathcal{B}} = R^{\mathcal{A}} \cap B^r$; we denote it by $\mathcal{A}[B]$.

While the theorems from the introduction make statements about structures over any fixed vocabulary, within their proofs we use relational structures whose forms are tailored to serve specific tasks:

We use structures to encode the adjacency relations of directed graphs.

DEFINITION 2.12 (Graph Structure). A *graph (structure)* is a structure $\mathcal{G} = (V, E^{\mathcal{G}})$ over the vocabulary $\tau_{\text{graph}} := \{E^2\}$, which consists of a binary *edge relation symbol.*

The concepts of a *subgraph* and a *subgraph that is induced* are inherited from general structures. We denote the set of edges (the universe) of a graph structure $\mathcal{G}$ also by $V(\mathcal{G})$ and its edge set also by $E(\mathcal{G})$.

To formulate a translation between MSO-formulæ and automata, we consider trees whose vertices are labeled using monadic relations.

Definition 2.13 (s-Tree Structure). For $s \in \mathbb{N}_0$, an *s-tree (structure)* is a structure $\mathcal{T} = (V, E^{\mathcal{T}}, R_1^{\mathcal{T}}, \ldots, R_s^{\mathcal{T}})$ over the vocabulary $\tau_{s\text{-tree}} = \tau_{\text{graph}} \cup \{R_1^1, \ldots, R_s^1\}$ where $(V, E^{\mathcal{T}})$ encodes the adjacency relation of a directed tree.

Formulæ and Definability. The formulæ of MSO-logic are all second-order formulæ where each variable is either a first-order variable, also called an *element variable*, or a monadic second-order variable, also called a *set variable*. We denote element variables by lowercase Latin letters like x, y, and z, and set variables by uppercase Latin letters like X, Y, and Z. Formally, MSO-formulæ are defined inductively as follows:

Definition 2.14 (MSO-Formula). Let τ be a vocabulary. An MSO-*formula over* τ is either

(1) an *atomic formula* of the form $x = y$, $X(z)$, or $R(x_1, \ldots, x_r)$, where $x, y, z, x_1, \ldots, x_r$ are element variables, X is a set variable, and $R^r \in \tau$, or

(2) a *composed formula* of the form $(\varphi) \wedge (\psi)$, $\neg(\varphi)$, $\exists x\, (\varphi)$, or $\exists X\, (\varphi)$ where φ and ψ are MSO-formulæ.

While the formal syntax of MSO-formulæ requires all subformulæ to be enclosed by brackets, for better readability we will sometimes omit some brackets that are implicitly known from the usual binding strength of the connectives and quantifiers. Beside the *Boolean connectives* $\wedge$ and $\neg$, the *existential element quantifier* $\exists x$, and the *existential set quantifiers* $\exists X$, which are formally part of the syntax of MSO-formulæ, we use the connectives $\vee$, $\rightarrow$, and $\leftrightarrow$, and the universal quantifiers $\forall x$ and $\forall X$ as shorthands. Moreover, we use $x \neq y$ as a shorthand for $\neg x = y$. *Bound* and *free* variables are defined as usual. We write $\varphi(x_1, \ldots, x_\ell, X_1, \ldots, X_k)$ to indicate that the formula φ has exactly the free element variables $x_1, \ldots, x_\ell$ and set variables $X_1, \ldots, X_k$.

Classically first-order logic is defined by using only element variables in the formula. We deviate from this and define first-order formulæ as MSO-formulæ without set quantifier, but free set variables. This allows to compare MSO- and first-order formulæ. This choice does not increase the expressive power of first-order formulæ since they are still not able to quantify over sets.

Definition 2.15 (First-Order Formula). Let τ be a vocabulary. A *first-order formula over* τ is an MSO-formula that does not contain set quantifiers, but may contain free set variables.

A *variable assignment* for a structure $\mathcal{A}$ is a mapping α whose domain is a finite set of element and set variables that maps each element variable x to an element $\alpha(x) \in A$ and each set variable X to a subset $\alpha(X) \subseteq A$. For an element variable x and $s \in A$, $\alpha[x \mapsto s]$ denotes the assignment that alters (or extends) α to map x to s; $\alpha[X \mapsto S]$ for a set variable X and a set $S \subseteq A$ is defined similarly.

Given a structure $\mathcal{A}$, an MSO-formula over the same vocabulary φ, and a variable assignment α that assigns a value to every free variable of φ, we write $(\mathcal{A}, \alpha) \models \varphi$ to indicate that $(\mathcal{A}, \alpha)$ *is a model of* φ, where the model relation is defined in the usual way. We also write $\mathcal{A} \models \varphi(s_1, \ldots, s_\ell, S_1, \ldots, S_k)$ for a formula $\varphi(x_1, \ldots, x_\ell, X_1, \ldots, X_k)$ if $s_1 = \alpha(x_1)$, …, $s_\ell = \alpha(x_\ell)$ and $S_1 = \alpha(X_1)$, …, $S_k = \alpha(X_k)$ hold.

Let τ be a vocabulary and φ a formula without free variables over τ. The set of τ-structures that is *defined* by φ contains exactly the τ-structures $\mathcal{A}$ with

$\mathcal{A} \models \varphi$. A set of τ-structures is *definable* in MSO-logic (in first-order logic) if it is defined by some MSO-formula (first-order formula). If φ is a formula with free element variables $\{x_1, \dots, x_\ell\}$ and set variables $\{X_1, \dots, X_k\}$, then it *defines* the set of pairs $(\mathcal{A}, \alpha)$ of τ-structures $\mathcal{A}$ and variable assignments α with domain $\{x_1, \dots, x_\ell, X_1, \dots, X_k\}$, such that $(\mathcal{A}, \alpha) \models \varphi$ holds. A set of such pairs is *definable* in MSO-logic (first-order logic) if there is an MSO-formula (a first-order formula) that defines it.

EXAMPLE 2.16 (Defining 3-Colorings and 3-Colorability). The set of *graphs with a valid 3-coloring* consists of all pairs of graphs $\mathcal{G} = (V, E^{\mathcal{G}})$ and sets $R \subseteq V$, $G \subseteq V$, and $B \subseteq V$, such that all vertices lie in some set and adjacent vertices lie in different sets. The pairs of such graphs and solution sets are exactly the ones that satisfy the first-order formula $\varphi_{\text{3-coloring}}(R, G, B) :=$

$$\forall v \Big(R(v) \lor G(v) \lor B(v) \land$$
$$\forall w \big(E(v,w) \rightarrow \neg(R(v) \land R(w)) \land \neg(G(v) \land G(w)) \land \neg(B(v) \land B(w)) \big) \Big) .$$

The set of *graphs that have valid 3-colorings* is defined by the MSO-formula $\exists R \, \exists G \, \exists B \, \varphi_{\text{3-coloring}}(R, G, B)$; this is the formula $\varphi_{\text{3-colorable}}$ from the introduction.

Monadic Second-Order Formulæ Without Element Variables. For some proofs it will be convenient to consider only MSO-formulæ that do not contain any first-order variables (free or bound). For this purpose, we transform MSO-formulæ into equivalent formulæ without element variables using different atomic formulæ.

DEFINITION 2.17 (MSO-Formula Without Element Variables). Let τ be a vocabulary. The formulæ over τ of MSO-*logic without element variables* are defined like the usual MSO-formulæ, but use a different set of atomic formulæ whose syntax and semantics is defined as follows:

(1) First, *empty*(X) is an atomic formula for every set variable X and semantically $\mathcal{A} \models$ *empty*(S) means $S = \emptyset$.

(2) Second, *elem*$(X_1, \dots, X_r, Z)$ is an atomic formula, where the X_i are set variables and Z is either a relation symbol from τ and $r = \text{ar}(Z)$, or Z is a set variable and $r = 1$. Semantically, $(\mathcal{A}, \alpha) \models$ *elem*$(X_1, \dots, X_r, Z)$ means $|\alpha(X_i)| = 1$ for each $i \in \{1, \dots, r\}$, and $\alpha(X_1) \times \cdots \times \alpha(X_r) \subseteq Z^{\mathcal{A}}$ when Z is a relation symbol and $\alpha(X_1) \subseteq \alpha(Z)$ when Z is a set variable.

Any MSO-formula that uses first-order variables can be transformed into an equivalent MSO-formula without first-order variables:

LEMMA 2.18. *For every* MSO*-formula $\varphi(y_1, \dots, y_\ell, X_1, \dots, X_k)$, there exists an* MSO*-formula without element variables $\psi(Y_1, \dots, Y_\ell, X_1, \dots, X_k)$ over the same vocabulary τ, such that for every τ-structure $\mathcal{A}$ with universe A, elements $s_1, \dots, s_\ell \in A$, and sets $S_1, \dots, S_k \subseteq A$, we have $\mathcal{A} \models \varphi(s_1, \dots, s_\ell, S_1, \dots, S_k)$ if, and only, if $\mathcal{A} \models \psi(\{s_1\}, \dots, \{s_\ell\}, S_1, \dots, S_k)$.*

PROOF. Let $\varphi(y_1, \dots, y_\ell, X_1, \dots, X_k)$ be any MSO-formula. We construct an MSO-formula without element variables $\psi(Y_1, \dots, Y_\ell, X_1, \dots, X_k)$ as follows:

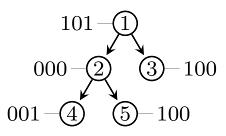
We start by introducing a fresh set variable for every element variable. Then we replace atomic formulæ that use element variables by atomic formulæ without element variables. Assume that X, Y, and X_1 to X_r are the set variables introduced for the element variables x, y, and x_1 to x_r, respectively. We replace every $x = y$ by $elem(X, Y) \wedge elem(Y, X)$, every $R(x_1, \ldots, x_r)$ for $R^r \in \tau$ by $elem(X_1, \ldots, X_r, R)$, and every $Z(x)$ where Z is a set variable by $elem(X, Z)$. The lemma holds for all atomic formulæ. To eliminate first-order variables from composed formulæ, we inductively replace every subformula $\exists x(\varphi')$ by the formula $\exists X(elem(X, X) \wedge \varphi')$. The first part of this formula expresses that we can only choose singleton sets, sets with exactly one element, for X. By induction, the second part ensures that choosing an element s for x has the same effect as choosing a singleton set $\{s\}$ for X. Thus, the formula constructed in this way proves the lemma. □

2.3. A Büchi–Elgot–Trakhtenbrot-type Theorem for Unordered Trees

In the following we prove a back and forth translation between MSO-formulæ on s-tree structures and multiset tree automata on labeled trees that preserves the sizes and number of solutions. This can be seen as extending the results of Büchi (1960), Elgot (1961), and Trakhtenbrot (1961) from strings and finite automata to (unranked and unordered) trees and multiset tree automata.

The translation from formulæ to automata, part (1) of Theorem 2.19, is used inside proofs from Chapter 4. While part (2) of Theorem 2.19 is not used in later chapters, we use one of its proof steps, Lemma 2.21 for moving from MSO-formulæ to equivalent first-order formulæ in case of bounded-depth trees, in Chapter 3.

To transform between labeled trees, on which automata work, and s-tree structures, for which MSO-formulæ define properties, we use the following definition: Given an s-tree structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ and sets $S_1, \ldots, S_k \subseteq V$, let us write $T(\mathcal{T}, S_1, \ldots, S_k)$ for the labeled tree over $\Sigma = \{0,1\}^{s+k}$ whose node set is V, whose edge set is $E^{\mathcal{T}}$, and whose labeling function maps each node $v \in V$ to $l_1 \ldots l_s x_1 \ldots x_k \in \{0,1\}^{s+k}$ with $l_i = 1 \Leftrightarrow v \in P_i^{\mathcal{T}}$ and $x_i = 1 \Leftrightarrow v \in S_i$. We write $T(\mathcal{T})$ in case $k = 0$. An example of this transformation is shown right for the 1-tree structure $\mathcal{T}$ with $V = \{1,2,3,4,5\}$, $E^{\mathcal{T}} = \{(1,2),(1,3),(2,4),(2,5)\}$, $P_1^{\mathcal{T}} = \{1,3,5\}$, $S_1 = \emptyset$, and $S_2 = \{1,4\}$.



THEOREM 2.19 (Back and Forth Between Formulæ and Automata on Unordered Trees). *Let $s, k \in \mathbb{N}_0$.*

(1) For every MSO*-formula $\varphi(X_1, \ldots, X_k)$ over $\tau_{s\text{-tree}}$ there is a multiset tree automaton A with alphabet $\{0,1\}^{s+k}$, such that for all s-tree structures $\mathcal{T}$ with universe V and $S_1, \ldots S_k \subseteq V$ we have $\mathcal{T} \models \varphi(S_1, \ldots, S_k)$ if, and only if, A accepts $T(\mathcal{T}, S_1, \ldots, S_k)$.*

(2) For every multiset tree automaton A with alphabet $\{0,1\}^{s+k}$ there is an MSO*-formula $\varphi(X_1, \ldots, X_k)$ over $\tau_{s\text{-tree}}$, such that for all s-tree structures $\mathcal{T}$ with universe V and $S_1, \ldots, S_k \subseteq V$ we have $\mathcal{T} \models \varphi(S_1, \ldots, S_k)$ if, and only if, A accepts $T(\mathcal{T}, S_1, \ldots, S_k)$.*

We start to prove Part (1) of Theorem 2.19.

Proof of Part (1) of Theorem 2.19. The proof follows ideas of Arnborg et al. (1991) that are modified to work for s-trees and multiset tree automata instead of degree-bounded trees and classical tree automata. We consider an MSO-formula φ without element variables from Definition 2.17 that is turned into an automaton by induction on its structure.

First, assume that φ is an atomic formula; that means it is either of the form $\mathit{empty}(X)$, $\mathit{elem}(X, Z)$ (where Z is a monadic relation symbol from $\tau_{s\text{-tree}}$ or a set variable), or $\mathit{elem}(X, Y, E)$. In each case, there exists a multiset tree automaton accepting exactly those trees satisfying the formula: (1) The automaton for $\mathit{empty}(X)$ has the same behaviour as the automaton from Example 2.6, which accepts exactly the labeled trees over $\{0, 1\}$ where all nodes are labeled by 0: It has two states q_0 and q_1 with only q_0 being accepting, and multiplicity $m = 1$. Working up from the leaves in state q_0, it switches to q_1 once it encounters a node that is an element of X and propagates this state to the root. The information of whether a node n is part of X can be looked up in its label. (2) For an atomic formula $\mathit{elem}(X, Z)$, we construct the intersection automaton from Lemma 2.7 for two automata: the first automaton tests whether X is a singleton set, and the second automaton tests that, if an element is part of X, it is also part of Z. (3) For the atomic formula $\mathit{elem}(X, Y, E)$, we combine three automata using Lemma 2.7. The first two automata test whether X and Y are singleton sets, respectively. The third automaton tests whether we have $\alpha(X) \times \alpha(Y) \subseteq E^{\mathcal{T}}$ if $|\alpha(X)| = |\alpha(Y)| = 1$. To verify this property, the automaton uses a state to signal each node that is an element of Y and tests whether each node is an element of X when one of its children has this state.

For the induction step, we must build an automaton for formulæ that are composed using negation, conjunction, and the existential set quantifier. These automata are constructed using the closure properties of multiset tree automata we proved earlier in Lemmas 2.7 to 2.9: When $\varphi = \varphi_1 \wedge \varphi_2$, we construct the intersection product automata from Lemma 2.7 of the automata for φ_1 and φ_2. When $\varphi = \neg\varphi'$, we take the complement automaton from Lemma 2.9. Finally, when $\varphi = \exists X(\varphi')$, we first transform the automaton for φ' into a nondeterministic automaton that makes its transition regardless of the relation X, using up- and down-projection of the alphabet, and then make it deterministic via the power set construction from Lemma 2.8. □

For the proof of Part (2) of Theorem 2.19, we first show that the transitions of multiset tree automata can be defined using first-order formulæ.

Lemma 2.20 (First-Order-Definable State Transitions). *Let $s, k \in \mathbb{N}_0$, and $A = (\Sigma, Q, Q_a, \Delta)$ be a multiset tree automaton with multiplicity bound m, alphabet $\Sigma = \{0, 1\}^{s+k}$ and states $Q = \{q_1, \dots, q_{|Q|}\}$. For ever $q_i \in Q$, there is a first-order formula $\varphi_{q_i}(X_1, \dots, X_k, x, Z_1, \dots, Z_{|Q|})$ over $\tau_{s\text{-tree}}$, such that for every s-tree structure $\mathcal{T}$ with universe V, sets $S_1, \dots, S_k \subseteq V$, $n \in V$, and a partition $Q_1, \dots, Q_{|Q|}$ of the children of n in $\mathcal{T}$, we have*

$$\mathcal{T} \models \varphi_{q_i}(S_1, \dots, S_k, n, Q_1, \dots, Q_{|Q|},) \text{ if, and only if, } (l(n), M|_m, q_i) \in \Delta\ ,$$

where M is the multiset $\{q_j \in Q \mid \text{ex. child } c \text{ of } n \text{ with } c \in Q_j\}$ and l is the labeling function of $T(\mathcal{T}, S_1, \dots, S_k)$.

Proof. For each $(\sigma, N, q_i) \in \Delta$ with $N = \{q_{i_1}, \dots, q_{i_n}\}$, we define the first-order formula $\varphi_{\sigma,N,q_i}(x, Z_1, \dots, Z_{|Q|}) :=$

$$\begin{aligned}\exists z_1 \dots \exists z_n \Big(& E(x, z_1) \wedge \dots \wedge E(x, z_n) \wedge \\ & \mathit{distinct}(z_1, \dots, z_n) \wedge Z_{i_1}(z_1) \wedge \dots \wedge Z_{i_n}(z_n) \wedge \\ & \forall z \big(E(x,z) \wedge \mathit{distinct}(z, z_1, \dots, z_n) \rightarrow \bigvee_{\substack{q_j \in Q \text{ s.t.} \\ \#_N(q_j) = m}} Z_j(z) \big) \Big) \ ,\end{aligned}$$

where *distinct* describes that distinct elements are assigned to its free variables. Given a partition $Q_1, \dots, Q_{|Q|}$ of the children of n, this formula tests whether the multiset M contains each state $q_i \in Q$ with $\#_N(q_i) < m$ exactly $\#_N(q_i)$ times and each state $q_i \in Q$ with $\#_N(q_i) = m$ at least $\#_N(q)$ times; this holds exactly if we have $M|_m = N$. A formula that tests whether the state q_i can be assigned to n provided that the symbol for n is σ is

$$\varphi_{\sigma,q_i}(x, Z_1, \dots, Z_{|Q|}) := \bigvee_{(\sigma,N,q_i) \in \Delta} \varphi_{\sigma,N,q_i}(x, Z_1, \dots, Z_{|Q|}) \ ,$$

the disjunction over all capped multisets of states that lead to valid transitions assigning q_i to n. Thus, the only thing left to do is to detect which symbol is present at n and apply the right formula. For this, we define the first-order formula $\varphi_{q_i}(X_1, \dots, X_k, x, Z_1, \dots, Z_{|Q|}) :=$

$$\bigwedge_{\sigma \in \Sigma} \left(\varphi_\sigma(X_1, \dots, X_k, x) \rightarrow \varphi_{\sigma,q_i}(x, Z_1, \dots, Z_{|Q|}) \right) \ ,$$

where φ_σ holds if σ is the label of n in $T(\mathcal{T}, S_1, \dots, S_k)$. This can easily be defined using a first-order formula that queries the monadic relations $R_i^{\mathcal{T}}$ of $\mathcal{T}$ and the set variables X_i. □

Proof of Part (2) of Theorem 2.19. The formula guesses a partition $Q_1, \dots, Q_{|Q|}$ of V using existential quantifiers that bind variables $Z_1, \dots, Z_{|Q|}$. Then it tests whether this assignment represents a valid computation of the automaton. In detail, it checks for every node n and $q_i \in Q$ whether $Z_i(n)$ implies $\varphi_{q_i}(S_1, \dots, S_k, n, Q_1, \dots, Q_{|Q|})$, which is a formula from Lemma 2.20. □

Using the transformation from MSO-formulæ to multiset tree automata, and the first-order definability of state transitions of multiset tree automata, we are already able to prove the following special case of Theorem 1.9 that first-order formulæ can simulate the behaviour of MSO-formulæ on trees of bounded depth. The *depth* of a node n in an s-tree structure $\mathcal{T}$ is the maximum number of vertices on a path from n to a leaf. The *depth* of $\mathcal{T}$ is the depth of its root.

Lemma 2.21. *Let $s, k \in \mathbb{N}_0$, $d \in \mathbb{N}$. For every* MSO*-formula $\varphi(X_1, \dots, X_k)$ over $\tau_{s\text{-tree}}$ there exists a first-order formula $\psi_d(X_1, \dots, X_k)$ over $\tau_{s\text{-tree}}$, such that for every s-tree structure $\mathcal{T}$ of depth d with universe V and $S_1, \dots, S_k \subseteq V$ we have $\mathcal{T} \models \varphi(S_1, \dots, S_k)$ if, and only if, $\mathcal{T} \models \psi_d(S_1, \dots, S_k)$.*

Proof. Let $A = (\Sigma, Q, Q_a, \delta)$ be the deterministic multiset tree automaton with $\Sigma = \{0,1\}^{s+k}$ from part (1) of Theorem 2.19, which is equivalent to φ.

We prove by induction over d for all $q \in Q$ that there is a first-order formula $\psi_{d,q}(X_1, \dots, X_k, x)$ with the following properties: For every s-tree $\mathcal{T}$

with universe V, node n of depth d from $\mathcal{T}$, and $S_1, \dots, S_k \subseteq V$, we have $\mathcal{T} = \psi_{d,q}(S_1, \dots, S_k, n)$ if, and only if, the state that is assigned to n in $T(\mathcal{T}, S_1, \dots, S_k)$ by A is q. Once this formula is available, we can prove the lemma using $\psi_d(X_1, \dots, X_k) :=$

$$\forall x \left(root(x) \to \bigvee_{q \in Q_a} \psi_{d,q}(X_1, \dots, X_k, x) \right) ,$$

where $root(x)$ says that the passed node is the root of the tree (the unique node without ingoing edges).

For $d = 1$, we use the formula $\psi_{1,q}(X_1, \dots, X_k, x) := \varphi'_q(X_1, \dots, X_k, x)$, where $\varphi'_q(X_1, \dots, X_k, x)$ is the formula we get from applying Lemma 2.20 to the automaton A and replacing every occurrence of a $Z_{q_i}(z)$ with a formula that is always false (like, for example, $\exists x \, x \neq x$). This formula has the claimed properties since n has no children and, thus, the multiset of states at the children is always empty. For the induction step, we use $\psi_{d,q}(X_1, \dots, X_k, x) := \varphi'_q(X_1, \dots, X_k, x)$ where, again, we get the formula $\varphi'_q(X_1, \dots, X_k, x)$ from applying Lemma 2.20 to A, but this time replacing each occurrence of a $Z_{q_i}(z)$ by

$$\bigwedge_{c=1}^{d-1} \left(depth_c(z) \to \psi_{c,q_i}(X_1, \dots, X_k, z) \right) ,$$

where $depth_c(z)$ is a first-order formula that tests whether the depth of the passed node in $\mathcal{T}$ is exactly c. Since the depth of the children of n in the tree is at most $d - 1$, by the induction hypothesis this formula correctly decides whether the state at a child of n is q_i. $\square$

CHAPTER 3

Bounded Tree Depth and First-Order Logic

Nešetřil and Ossona de Mendez (2006) generalized the notion of a depth bound for trees to graphs by defining the concept of bounded tree depth. In the present chapter we show that on such graphs, and logical structures, the expressive power of first-order logic and monadic second-order logic coincides. We prove the following theorem, which implies Theorem 1.9 from the introduction as a special case for $k = 0$, and Lemma 2.21 from the previous section for tree structures.

THEOREM 3.1. *For every $d \in \mathbb{N}$, and every* MSO*-formula $\varphi(X_1, \dots, X_k)$ over some vocabulary τ, there is a first-order formula $\psi(X_1, \dots, X_k)$ over τ, such that for all τ-structure $\mathcal{A}$ of tree depth at most d with universe A and all sets $S_1, \dots, S_k \subseteq A$, we have $\mathcal{A} \models \varphi(S_1, \dots, S_k)$ if, and only if, $\mathcal{A} \models \psi(S_1, \dots, S_k)$.*

In Section 3.4 we see that the preceding theorem still holds when MSO-logic is replaced by the more general guarded second-order logic (GSO-logic).

Tree depth is defined via a recursive process that deletes elements from disconnected parts of a structures in a parallel fashion. The minimum number of recursions that are needed in this process is the tree depth of a structure. Our first step towards proving Theorem 3.1 is to rigorously prove that this process can be implemented by first-order formulæ. In detail, we show that, for any $d \in \mathbb{N}$, the class of structures of tree depth at most d is definable using a first-order formula. For the proof of Theorem 3.1, we then extend this formula, such that it evaluates a fixed MSO-formula at the same time along the element deletion process. To achieve this, we combine the information of which MSO-formulæ hold in an unbounded number of substructure that arise during the recursive deletion process. Formally this is done by utilizing a newly developed Feferman–Vaught-type composition theorem that shows how to evaluate, using first-order formulæ, MSO-formulæ on structures that are partitioned into an unbounded number of substructure for which it is known which MSO-formulæ hold and which do not hold. The proof of the composition theorem, in turn, is based on Lemma 2.21 from the previous chapter.

A variant of Theorem 3.1 for graph structures was developed as part of a work of Elberfeld, Grohe, and Tantau (2012a) that presents graph-invariant-based characterizations of when first-order logic has the same expressive power as second-order logics like MSO and GSO, which are normally more expressive. In this paper it is shown that the property of having bounded tree depth is a necessary and sufficient condition for MSO-logic and first-order logic to have the same expressive power when considering classes of structures that are closed under taking substructures, and the same holds when MSO-logic is replaced by GSO-logic and closure under substructures by closure under induced substructures. A difference between Theorem 3.1 in the present dissertation and the same theorem in the paper lies under the hood, in the proof techniques: While the paper uses model-theoretic properties that are based on playing Ehrenfeucht–Fraïssé games in a non-constructive way to move from MSO- to first-order formulæ, the proof in my thesis is based on the effective construction of multiset tree automata for MSO-formulæ and their simulation using first-order formulæ in case of depth-bounded trees.

The present chapter is organized as follows: Section 3.1 contains the composition theorem. In Section 3.2, tree depth is formally defined and its first-order definability for any constant bound $d \in \mathbb{N}$ is shown. In Section 3.3 we combine the composition theorem with tree-depth-defining formulæ to prove Theorem 3.1. Section 3.4 shows the generalization to GSO-logic.

3.1. A Feferman–Vaught-type Theorem for Unbounded Partitions

The question answered by Feferman–Vaught-type composition theorems is the following: Suppose a logical structure $\mathcal{A}$ is the disjoint union of two structures $\mathcal{A}_1$ and $\mathcal{A}_2$ and suppose we wish to find out whether $\mathcal{A} \models \varphi$ holds; can we decide this solely based on knowing which formulæ hold in $\mathcal{A}_1$ and $\mathcal{A}_2$? Feferman and Vaught (1959) presented such a theorem for first-order formulæ. Intuitively, this should also be the case at least for logics like MSO where a formula cannot establish connections between the two disjoint parts of $\mathcal{A}$ and, indeed, a basic extension for MSO-logic states exactly this: For every MSO-formula φ we can decide $\mathcal{A} \models \varphi$ solely based on knowing for which formulæ φ' of at most the same quantifier rank we have $\mathcal{A}_1 \models \varphi'$ and for which we have the property $\mathcal{A}_2 \models \varphi'$. An elegant proof of this is based on determining the set of MSO-formulæ that hold in a structure, which is called the *type* of the structure, by playing an appropriate kind of Ehrenfeucht–Fraïssé game on it; since strategies for the individual structures $\mathcal{A}_1$ and $\mathcal{A}_2$ can be combined into a strategy for the whole structure $\mathcal{A}$, we get the claim as discussed, for example, by Makowsky (2004). One can think of extending the basic composition theorem for MSO-logic in three ways:

(1) First, one can consider structures that are not partitioned into two completely disjoint parts, but where the *parts are related* in some way like having a constant-size overlap or being the shores of a complete bipartite graph that is a substructure of the whole structure (Courcelle and Engelfried, 2012).

(2) Second, one can make explicit how to *construct* the type of $\mathcal{A}$ when the types of $\mathcal{A}_1$ and $\mathcal{A}_2$ are given as input. Constructive type composition theorems for MSO, like the one proved by Courcelle and Engelfried (2012), state that for each MSO-formula φ one can construct a propositional formula f_φ that has two propositional variables $p^1_{\varphi'}$ and $p^2_{\varphi'}$ for a finite number of MSO-formulæ φ'. When we set these propositional variables to true or false, depending on whether φ' hold in $\mathcal{A}_1$ and $\mathcal{A}_2$, the formula f_φ will evaluate to true if, and only if, $\mathcal{A} \models \varphi$. One can extend this idea to any fixed number of structures $\mathcal{A}_1, \dots, \mathcal{A}_k$ by introducing new propositional variables $p^3_{\varphi'}$ to $p^k_{\varphi'}$ as done by Makowsky (2004).

(3) Third, instead of considering only partitions into a fixed number of structures, one can consider partitions into an *unbounded* number of structures. The non-constructive theorem based on Ehrenfeucht–Fraïssé games still works in this setting, but the constructive version for a bounded number of parts in the partition that is based on propositional formulæ cannot be generalized

since a single propositional formula has only access to a constant number of propositional variables.

In the present section, we develop a type composition theorem that is both constructive and works for an unbounded number of parts with a constant-size overlap. The idea is to use first-order formulæ rather than propositional formulæ in order to define whether a formula holds in a structure $\mathcal{A}$ that is the union of an arbitrary number of substructures $\mathcal{A}_i$ for $i \in I$. Instead of having to introduce new propositional variables as the number of substructures increases, we simply enlarge the universe: We consider a structure $\mathcal{I}$ whose universe is the index set I, and, instead of using propositional variables $p^i_{\varphi'}$ inside a propositional formula, we use atomic formulæ $R_{\varphi'}(i)$ inside a first-order formula R_φ is a monadic relation symbol that tells us whether φ' holds in the structure $\mathcal{A}_i$. The result is a first-order formula that takes a structure as input that encodes which MSO-formulæ hold in the substructures $\mathcal{A}_i$ and outputs whether $\mathcal{A} \models \varphi$ holds. The composition theorem encompasses both the classical non-constructive theorem for unbounded index sets (the theorem shows that the mapping is first-order definable instead of non-constructively claiming that there is some unique mapping) and the constructive version for a fixed number of substructures (in case of fixed-size structures, one can transform the first-order formulæ into an equivalent propositional formula) as special cases. To state the theorem, we first review the notion of logical types and define type indicators that encode the types of substructures.

Types and Type Indicators. We say that two MSO-formulæ φ and ψ over the same vocabulary τ are *equivalent*, written $\varphi \equiv \psi$, if for every τ-structure $\mathcal{A}$ and every variable assignment α we have $(\mathcal{A}, \alpha) \models \varphi$ if, and only if, $(\mathcal{A}, \alpha) \models \psi$. The *(quantifier) rank* $\operatorname{qr}(\varphi)$ of an MSO-formula φ is the nesting depth of quantifiers in the formula. For a set Φ of formulæ, let us write $\Phi/{\equiv}$ for the set of all equivalence classes $[\varphi]_\equiv$ of formulæ from Φ with respect to the equivalence relation $\equiv$. Formulæ that have the free set variables X_1 to X_k and rank q are called *k-variable rank-q formulæ*. Variable assignments α with domain $\{X_1, \dots, X_k\}$ are called *k-variable assignments*.

FACT 3.2. *Let τ be a vocabulary and $k, q \in \mathbb{N}_0$. Let Φ be the set of all k-variable rank-q MSO-formulæ. Then $\Phi/{\equiv}$ contains finitely many equivalence classes.*

The fact can be proved by defining normal forms for MSO-formulæ as described, for example, by Libkin (2004). In such a proof, formulæ in normal form are used to build a *representative system* for the k-variable rank-q MSO-formulæ over some vocabulary τ, which is a finite set of formulæ that contains exactly one formula out of each class of equivalent formulæ. Let $\text{MSO}_{k,q}[\tau]$ denote such a finite representative system for the k-variable rank-q MSO-formulæ.

DEFINITION 3.3 (Type). Let τ be a vocabulary and $k, q \in \mathbb{N}_0$. Let $\mathcal{A}$ be a τ-structure and α a k-variable assignment. The *rank-q type of* $(\mathcal{A}, \alpha)$ is the set

$$\operatorname{tp}_q(\mathcal{A}, \alpha) := \{\varphi \in \text{MSO}_{k,q}[\tau] \mid (\mathcal{A}, \alpha) \models \varphi\} .$$

The *set of all rank-q types for τ-structures and k-variable assignments*, denoted by $\Theta_{k,q}[\tau]$, contains exactly the formula sets $\theta \subseteq \text{MSO}_{k,q}[\tau]$ with $\theta = \operatorname{tp}_q(\mathcal{A}, \alpha)$ for some τ-structure $\mathcal{A}$ and k-variable assignment α.

Since $\textsc{mso}_{k,q}[\tau]$ contains only finitely many (representative) formulæ, types are finite sets and, hence, $\Theta_{k,q}[\tau]$ is finite. As a consequence, sets of structures and variable assignments having a certain type can be defined using $\textsc{mso}$-formulæ:

LEMMA 3.4. *Let τ be a vocabulary and $k, q \in \mathbb{N}_0$. For every type $\theta \in \Theta_{k,q}[\tau]$ there is a formula $\beta_\theta \in \textsc{mso}_{k,q}[\tau]$, such that for every τ-structure $\mathcal{A}$ and every k-variable assignment α, we have*

$$(\mathcal{A}, \alpha) \models \beta_\theta \textit{ if, and only if, } \operatorname{tp}_q(\mathcal{A}, \alpha) = \theta\ .$$

PROOF. Let $\beta_\theta \in \textsc{mso}_{k,q}[\tau]$ be equivalent to the k-variable rank-q $\textsc{mso}$-formula

$$\bigwedge_{\varphi \in \theta} \varphi \wedge \bigwedge_{\varphi \in \textsc{mso}_{k,q}[\tau] \setminus \theta} \neg\varphi\ .$$

The formula β_θ holds for $(\mathcal{A}, \alpha)$ exactly if the formulæ from θ hold for $(\mathcal{A}, \alpha)$ and the representative formulæ not in θ do not hold for $(\mathcal{A}, \alpha)$. By definition, this is exactly the case when $\theta = \operatorname{tp}_{k,q}(\mathcal{A}, \alpha)$. ☐

To formulate the composition theorem, we define rooted partitions of structures and a structure that encodes information about the types of the parts of a rooted partition.

DEFINITION 3.5 (Rooted Structure). Let $w \geq \mathbb{N}_0$ and τ be a vocabulary. A *width-w rooted τ-structure* is a structure $\mathcal{A}$ over $\tau \cup \{B_1^1, \ldots, B_w^1\}$ (where the B_ℓ are fresh monadic relation symbols not present in τ), such that for each $\ell \in \{1, \ldots, w\}$ the set $B_\ell^{\mathcal{A}}$ is a singleton; $B(\mathcal{A}) = \bigcup_{\ell=1}^{w} B_\ell^{\mathcal{A}}$ is the *bag* of $\mathcal{A}$.

DEFINITION 3.6 (Rooted Partition). Let $\mathcal{A}$ be a rooted structure for some width $w \in \mathbb{N}_0$ and vocabulary τ. A *rooted partition* of $\mathcal{A}$ is a family $F = (\mathcal{A}_i)_{i \in I}$ of width-w rooted τ-structures, such that the following holds:

(1) The union of all $\mathcal{A}_i$ is exactly $\mathcal{A}$.
(2) Each $\mathcal{A}_i$ is a substructure of $\mathcal{A}$ induced by some set $A_i \subseteq A$.
(3) For all distinct $i, j \in I$, we have $A_i \cap A_j = B(\mathcal{A})$.

Note that in a width-0 rooted partition $(\mathcal{A}_i)_{i \in I}$, the structure $\mathcal{A}$ is the disjoint union of the $\mathcal{A}_i$. Let α be a variable assignment for a rooted structure $\mathcal{A}$ with rooted partition $(\mathcal{A}_i)_{i \in I}$. Then $\alpha_i(X) := \alpha(X) \cap A_i$ denotes the *restriction of α to the universe A_i of $\mathcal{A}_i$*. Let $\tau_{k,q,w}$ be the vocabulary that contains a monadic relation symbol R_θ^1 for each $\theta \in \Theta_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}]$.

DEFINITION 3.7 (Type Indicators). Let τ be a vocabulary and $k, q, w \in \mathbb{N}_0$. Let $F = (\mathcal{A}_i)_{i \in I}$ be a rooted partition of some width-w rooted τ-structure $\mathcal{A}$ and α be a k-variable assignment for $\mathcal{A}$. The *rank-q type indicator* of F and α is the $\tau_{k,q,w}$-structure $\mathcal{I} = \mathcal{I}_q(F, \alpha)$ with universe I and relation

$$R_\theta^{\mathcal{I}} := \{i \in I \mid \theta = \operatorname{tp}_q(\mathcal{A}_i, \alpha_i)\}$$

for every monadic relation symbol $R_\theta \in \tau_{k,q,w}$.

From knowing the types of the parts of a rooted partition that are encoded in a type indicator, we can determine which formulæ hold in them.

Lemma 3.8. *Let τ be a vocabulary and $k, q, w \in \mathbb{N}_0$. For every k-variable rank-q MSO-formula φ over $\tau \cup \{B_1^1, \ldots, B_w^1\}$, there exists a first-order formula $\gamma_\varphi(x)$ over $\tau_{k,q,w}$, such that for every rooted partition $F = (\mathcal{A}_i)_{i \in I}$ of some width-w rooted τ-structure $\mathcal{A}$, k-variable assignment α for $\mathcal{A}$, and $i \in I$, we have*

$$(\mathcal{A}_i, \alpha_i) \models \varphi \text{ if, and only if, } \mathcal{I}_q(F, \alpha) \models \gamma_\varphi(i) \ .$$

Proof. Let $\varphi' \in \text{MSO}_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}]$ with $\varphi' \equiv \varphi$. We set

$$\gamma_\varphi(x) := \bigvee_{\substack{\theta \in \Theta_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}] \\ \text{s.t. } \varphi' \in \theta}} R_\theta(x) \ .$$

$\square$

Formulation and Proof of the Composition Theorem. The composition theorem states that one can define the type of a structure based on knowing the types of the parts of a rooted partition for it. First, we prove that defining types can be done using MSO-formulæ in Theorem 3.9, then we show that the used MSO-formulæ can be replaced by equivalent first-order formulæ in Theorem 3.10.

Theorem 3.9 (Composing Types by Monadic Second-Order Formulæ). *For every type $\theta \in \Theta_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}]$ for some $k, q, w \in \mathbb{N}_0$ and vocabulary τ, there is an MSO-formula ρ_θ without free variables over $\tau_{k,q,w}$, such that for every rooted partition $F = (\mathcal{A}_i)_{i \in I}$ of a width-w rooted τ-structure $\mathcal{A}$, and every k-variable assignment α for $\mathcal{A}$, we have*

$$\text{tp}_q(\mathcal{A}, \alpha) = \theta \quad \text{if, and only if,} \quad \mathcal{I}_q(F, \alpha) \models \rho_\theta.$$

Proof. By Lemma 3.4, we can test whether $\theta \in \Theta_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}]$ is the type of $(\mathcal{A}, \alpha)$ using an MSO-formula $\beta_\theta \in \text{MSO}_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}]$. Thus, to prove the lemma, it is enough to show the following claim.

Claim. *For every formula $\varphi \in \text{MSO}_{k,q}[\tau \cup \{B_1^1, \ldots, B_w^1\}]$ for some $k, q, w \in \mathbb{N}_0$ and vocabulary τ, there is an MSO-formula ρ_φ without free variables over $\tau_{k,q,w}$, such that for every rooted partition $F = (\mathcal{A}_i)_{i \in I}$ of a width-w rooted τ-structure $\mathcal{A}$, and every k-variable assignment α for $\mathcal{A}$, we have*

$$(\mathcal{A}, \alpha) \models \varphi \text{ if, and only if, } \mathcal{I}_q(F, \alpha) \models \rho_\varphi \ .$$

For the proof of the claim, we consider MSO-formulæ without element variables from Definition 2.17.

Let $w \in \mathbb{N}_0$ be any width bound that is fixed throughout the proof. We prove the claim by induction on the structure of φ for all $k \in \mathbb{N}_0$

We start with atomic formulæ. For $\varphi = \mathit{empty}(X)$ we set $\rho_\varphi := \forall x\, (\gamma_\varphi(x))$, where $\gamma_\varphi(x)$ is the formula for φ from Lemma 3.8. We have $(\mathcal{A}, \alpha) \models \mathit{empty}(X)$ if, and only if, for all $i \in I$ we have $(\mathcal{A}_i, \alpha_i) \models \mathit{empty}(X)$, which is exactly what $\mathcal{I}_q(F, \alpha) \models \forall x\, (\gamma_{\mathit{empty}(X)}(x))$ means.

For $\varphi = \mathit{elem}(X_1, \ldots, X_r, R)$ with $R^r \in \tau$, we set

$$\rho_\varphi := \exists i \Big(\gamma_\varphi(i) \wedge \forall j \big(j \neq i \rightarrow \bigwedge_{m=1}^{r} \gamma_{\text{empty}(X_m)}(j) \vee \gamma_{\bigvee_{\ell=1}^{w} \mathit{elem}(X_m, B_\ell)}(j) \big) \Big) \ .$$

For the correctness proof first assume that $(\mathcal{A}, \alpha) \models \varphi$ holds. Then we know $|\alpha(X_m)| = 1$ for each $m \in \{1, \dots, r\}$ and $\alpha(X_1) \times \cdots \times \alpha(X_r) \subseteq R^{\mathcal{A}}$. Since $\mathcal{A}$ is the union of the $\mathcal{A}_i$, we have $\alpha(X_1) \times \cdots \times \alpha(X_r) \subseteq R^{\mathcal{A}_i}$ for some $\mathcal{A}_i$. Moreover, each singleton $\alpha(X_m)$ is either part of the bag $B(\mathcal{A})$, and we have $\alpha(X_m) \subseteq B_\ell^{\mathcal{A}_j}$ for some $\ell \in \{1, \dots, w\}$ and all $\mathcal{A}_j$, or it is not part of the bag, and $\alpha(X_m) \not\subseteq A_j$ holds for all $j \neq i$. For the other direction assume $\mathcal{I}_q(F, \alpha) \models \rho_\varphi$. The formula witnesses that there exists some i with $(\mathcal{A}_i, \alpha_i) \models \mathit{elem}(X_1, \dots, X_m, R)$ and for all other $\mathcal{A}_j$ and sets $\alpha(X_m)$, we have $A_j \cap \alpha(X_m) \subseteq B(\mathcal{A})$. From the definition of rooted partitions, we know $B(\mathcal{A}) \subseteq A_i$, which implies $\alpha(X_m) \subseteq A_i$ for each $m \in \{1, \dots, r\}$. Thus, $(\mathcal{A}, \alpha) \models \mathit{elem}(X_1, \dots, X_m, R)$ follows from $(\mathcal{A}_i, \alpha_i) \models \mathit{elem}(X_1, \dots, X_m, R)$.

For $\varphi = \mathit{elem}(X_1, Z)$, where Z is a set variable, the formula and correctness arguments are the same, except that we work with a set $\alpha(Z)$ that is assigned to Z instead of a relation $R^{\mathcal{A}}$ from the structure.

For the inductive step, we start with $\varphi = \neg\varphi'$. Here we can set $\rho_\varphi := \neg\rho_{\varphi'}$. Clearly, this has the required properties. Similarly, for $\varphi = \varphi_1 \wedge \varphi_2$, setting $\rho_\varphi := \rho_{\varphi_1} \wedge \rho_{\varphi_2}$ also has the desired properties. The proof is more involved for $\varphi = \exists X_{k+1}(\varphi')$. Let $\rho_{\varphi'}$ be the formula over $\tau_{k+1,q-1,w}$ from the induction hypothesis and let $\Theta_{k+1,q-1}[\tau \cup \{B_1^1, \dots, B_w^1\}] = \{\theta_1, \dots, \theta_n\}$.

For each $W \subseteq \{1, \dots, w\}$ and type θ_j, we consider the following k-variable rank-q MSO-formula over $\tau \cup \{B_1^1, \dots, B_w^1\}$:

$$\psi_{W,\theta_j} := \exists X_{k+1}\, \beta_{\theta_j} \wedge \bigwedge_{\ell \in W} \mathit{elem}(B_\ell, X_{k+1}) \quad \bigwedge_{\ell \in \{1,\dots,w\} \setminus W} \neg\mathit{elem}(B_\ell, X_{k+1}) \ ,$$

where β_{θ_j} is the $(k+1)$-variable rank-$(q-1)$ formula over τ from Lemma 3.4, which defines the set of pairs of $\tau \cup \{B_1^1, \dots, B_w^1\}$-structures and $(k+1)$-variable assignments of type θ_j.

We show that the following formula over $\tau_{k,q,w}$ has the claimed properties:

$$\begin{aligned} \rho_\varphi := \exists X_{\theta_1} \dots \exists X_{\theta_n} \Big(& \\ & \mathit{partition}(X_{\theta_1}, \dots, X_{\theta_n}) \wedge \rho'_{\varphi'} && (1) \\ \bigvee_{W \subseteq \{1,\dots,w\}} & \forall x \, \big(\bigwedge_{j=1}^{n} X_{\theta_j}(x) \to \gamma_{\psi_{W,\theta_j}}(x) \big) \Big) \ , && (2) \end{aligned}$$

where the formula *partition* tests whether the passed sets partition I, and $\rho'_{\varphi'}$ results from $\rho_{\varphi'}$ by substituting every relation symbol R_{θ_j} with X_{θ_j}.

We prove the correctness of this construction in two directions: First, assume $(\mathcal{A}, \alpha) \models \varphi$, which means that there exists a set $S \subseteq A$ with $(\mathcal{A}, \alpha[X_{k+1} \mapsto S]) \models \varphi'$. The rooted partition F of $\mathcal{A}$ and the assignment $\alpha[X_{k+1} \mapsto S]$ give rise to an indicator structure $\mathcal{I}_{q-1} = \mathcal{I}_{q-1}(F, \alpha[X_{k+1} \mapsto S])$ over $\tau_{k+1,q-1,w}$. By the induction hypothesis, we know $\mathcal{I}_{q-1} \models \rho_{\varphi'}$. To show $\mathcal{I}_q = \mathcal{I}_q(F, \alpha) \models \rho_\varphi$, we describe an assignment of sets to the variables X_{θ_j} and show that it satisfies the three parts of the formula. For the assignment, we put an index $i \in I$ into the set for X_{θ_j} if it lies in $R_{\theta_j}^{\mathcal{I}_{q-1}}$. Since the relations of $\mathcal{I}_{q-1}$ partition I, this also holds for the sets assigned to the variables X_{θ_j}. Moreover, since the sets for the variables X_{θ_j} are exactly the monadic relations $R_{\theta_j}^{\mathcal{I}_{q-1}}$, we know that $\rho'_{\varphi'}$ is satisfied. Thus, line (1) of ρ_φ is satisfied. To show that line (2) is satisfied, we consider $W := \{\ell \in \{1,\dots,w\} \mid B_\ell^{\mathcal{A}} \subseteq S\}$ and the part of the disjunction for this set. Let $i \in I$ be any index and let X_{θ_j} be a variable whose set contains i. Let $S_i = S \cap A_i$. By the construction of the set for X_{θ_j} we know $\mathrm{tp}_{q-1}(\mathcal{A}_i, \alpha_i[X_{k+1} \mapsto S_i]) = \theta_j$, which can be written as $(\mathcal{A}_i, \alpha_i[X_{k+1} \mapsto S_i]) \models \beta_{\theta_j}$ using Lemma 3.4. Since F is a rooted partition, we have $W = \{\ell \in \{1,\dots,w\} \mid B_\ell^{\mathcal{A}} \subseteq S_i\}$ and, therefore,

$$(\mathcal{A}_i, \alpha_i[X_{k+1} \mapsto S_i]) \models \bigwedge_{\ell \in W} \mathit{elem}(B_\ell, X_{k+1}) \bigwedge_{\ell \in \{1,\dots,w\}\setminus W} \neg\mathit{elem}(B_\ell, X_{k+1})$$

Putting both arguments together, we know $(\mathcal{A}_i, \alpha_i) \models \psi_{W,\theta_j}$ and, hence, $\mathcal{I}_q \models \gamma_{\psi_{W,\theta_j}}(i)$.

For the other direction assume $\mathcal{I}_q = \mathcal{I}_q(F,\alpha) \models \rho_\varphi$. Consider any assignment of sets to the variables X_{θ_j} that satisfy the inner part of the formula ρ_φ. Let $W \subseteq \{1,\dots,w\}$ be a set of indices for which the disjunction in line (2) holds. From line (1) we know that for each $i \in I$ there is exactly one X_{θ_j} whose set contains i, which implies that $\gamma_{\psi_{W,\theta_j}}(i)$ is true. We can equivalently write the last statement as $(\mathcal{A}_i, \alpha_i) \models \psi_{W,\theta_j}$. This means there exists a set $S_i \subseteq A_i$ that has the following two properties: $\mathrm{tp}_q(\mathcal{A}_i, \alpha_i[X_{k+1} \models S_i]) \models \theta_j$ and $W = \{\ell \in \{1,\dots,w\} \mid B_\ell^{\mathcal{A}} \subseteq S_i\}$. We combine the sets S_i, which agree on the bag $B(\mathcal{A})$, to a set $S := \bigcup_{i\in I} S_i$. Since $\rho'_{\varphi'}$ is satisfied, we have $\mathcal{I}_{q-1} = \mathcal{I}_{q-1}(F, \alpha[X_{k+1}) \mapsto S] \models \rho_{\varphi'}$. Applying the induction hypothesis, we know $(\mathcal{A}, \alpha[X_{k+1} \mapsto S]) \models \varphi'$ and, thus, $(\mathcal{A}, \alpha) \models \exists X_{k+1}\, \varphi'$. $\square$

The next theorem states that we can turn the MSO-formulæ from the previous theorem into first-order formulæ.

THEOREM 3.10 (Composing Types by First-Order Formulæ). *For every type $\theta \in \Theta_{k,q}[\tau \cup \{B_1^1,\dots,B_w^1\}]$ for some $k,q,w \in \mathbb{N}_0$ and vocabulary τ, there is a first-order formula ρ_θ without free variables over $\tau_{k,q,w}$, such that for every rooted partition $F = (\mathcal{A}_i)_{i\in I}$ of a width-w rooted τ-structure $\mathcal{A}$, and every k-variable assignment α for $\mathcal{A}$, we have*

$$\mathrm{tp}_q(\mathcal{A},\alpha) = \theta \quad \textit{if, and only if,} \quad \mathcal{I}_q(F,\alpha) \models \rho_\theta.$$

PROOF. Let $\rho_\theta^{(1)}$ be the MSO-formula over $\tau_{k,q,w}$ from Theorem 3.9. We turn $\rho_\theta^{(1)}$ into an equivalent first-order formula by using the transformation from MSO-formulæ to equivalent first-order formulæ in case of bounded-depth s-trees from Lemma 2.21. In order to apply the lemma, let $\mathcal{T}$ be the structure over the vocabulary $\{E^2\} \cup \tau_{k,q,w}$ that arises from $\mathcal{I} = \mathcal{I}_q(F,\alpha)$ by adding a new *root element* r to the universe and the relation $E^{\mathcal{T}}$ that contains a pair (r,i) for each $i \in I$. Moreover, adjust $\rho_\theta^{(1)}$ to a formula $\rho_\theta^{(2)}$, such that $\mathcal{I} \models \rho_\theta^{(1)}$ if, and only if, $\mathcal{T} \models \rho_\theta^{(2)}$; the new formula just ignores the existence of the root element r, which can be singled out during quantifications since it is the only element in $\mathcal{T}$ without ingoing edges. Since $\mathcal{T}$ is a depth-2 s-tree structure for some s that only depends on w, k, q, and τ, we are able to apply Lemma 2.21 to turn $\rho_\theta^{(2)}$ into an equivalent first-order formula $\rho_\theta^{(3)}$; that means $\mathcal{T} \models \rho_\theta^{(2)}$ if, and only if, $\mathcal{T} \models \rho_\theta^{(3)}$. The last step is to turn $\rho_\theta^{(3)}$ back into a first-order formula ρ_θ over $\tau_{k,q,w}$, such that $\mathcal{T} \models \rho_\theta^{(3)}$ if, and only if, $\mathcal{I} \models \rho_\theta$. Since the root r of $\mathcal{T}$ does not encode any information in the sense that it is connected to all other elements and is not part of any of the monadic relations, we can get rid of the relation symbol E in $\rho_\varphi^{(3)}$ and the element r in $\mathcal{T}$: Imagine we decide, for each element variable of $\rho_\varphi^{(3)}$, whether we choose r for it, or not. Then we can adjust $\rho_\varphi^{(3)}$ to a formula that has the desired property with respect to this choice by replacing atomic formulæ $E(x,y)$ with formulæ that are always true or always false, and leaving each $R_{\theta'}(x)$ either unchanged (if we decided to use elements from I for x) or replace it by a formula that is always false (if we decided to use r for x). A large disjunction over all choices proves the lemma. □

3.2. Getting Familiar with Tree Depth

In the present section, we define the notion of tree depth and learn enough about it to prove its first-order definability used in Section 3.3 to show Theorem 3.1.

Definition of Tree Depth. As discussed in the previous chapter, a graph can be represented by a relational structure $\mathcal{G} = (V, E^{\mathcal{G}})$ over the vocabulary $\tau_{\text{graph}} = \{E^2\}$. A *(simple) path* from a vertex $s \in V(\mathcal{G})$ to a vertex $t \in V(\mathcal{G})$ in a graph $\mathcal{G}$ is a sequence of distinct vertices $v_1, \ldots, v_\ell$ with $s = v_1$, $t = v_\ell$, and $(v_i, v_{i+1}) \in E(\mathcal{G})$ for all $i \in \{1, \ldots, m-1\}$; this *path's length* is $m - 1$. An *undirected graph* is a graph with a symmetric edge relation. An undirected graph is *connected* if there exists a path between any two of its vertices. The *(connected) components* of an undirected graph $\mathcal{G}$ are its maximal connected induced subgraphs.

The tree depth of undirected graphs was defined by Nešetřil and Ossona de Mendez (2006). We consider a generalized definition for relational structures based on their Gaifman graphs. The Gaifman graph of a structure is the following undirected graph that describes how the structure's elements are connected by relations.

Definition 3.11 (Gaifman Graph). Let $\mathcal{A}$ be a structure with universe A. The *Gaifman graph* of $\mathcal{A}$, denoted by $G(\mathcal{A})$, is the undirected graph (structure) whose vertex set is A and there is an edge $(a, a') \in E(G(\mathcal{A}))$ for $a, a' \in A$ if a relation $R^{\mathcal{A}}$ contains a tuple $(a_1, \ldots, a_{\text{ar}(R)}) \in R^{\mathcal{A}}$ with $a, a' \in \{a_1, \ldots, a_{\text{ar}(R)}\}$.

Let $\mathcal{A}$ be a relational structure and $(\mathcal{G}_i)_{i \in I}$ (for some finite set of indices I) be the components of its Gaifman graph $G(\mathcal{A})$. The *components* of $\mathcal{A}$ are the induced substructures $\mathcal{A}_i := \mathcal{A}[V(\mathcal{G}_i)]$ for $i \in I$; $\mathcal{A}$ is *connected* if $|I| = 1$.

Definition 3.12 (Tree Depth). Let $\mathcal{A}$ be a relational structure with universe A and $(\mathcal{A}_i)_{i \in I}$ be the components of $\mathcal{A}$.

The *tree depth* of $\mathcal{A}$ is

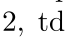
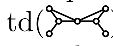
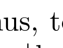
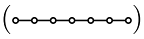
$$\text{td}(\mathcal{A}) := \begin{cases} 1 & \text{if } |A| = 1, \\ 1 + \min_{r \in A} \text{td}(\mathcal{A}[A \setminus \{r\}]) & \text{if } |A| > 1 \text{ and } |I| = 1, \\ \max_{i \in I} \text{td}(\mathcal{A}_i) & \text{otherwise.} \end{cases}$$

A class of structures C over some vocabulary τ has *bounded tree depth* (is *tree-depth-bounded*) if there exists a constant $d \in \mathbb{N}$, such that $\text{td}(\mathcal{A}) \leq d$ for every $\mathcal{A} \in C$.

The following fact follows immediately from the definition of tree depth and the definition of the components of a structure based on its Gaifman graph.

FACT 3.13. *For every structure $\mathcal{A}$, we have* $\operatorname{td}(\mathcal{A}) = \operatorname{td}(G(\mathcal{A}))$.

In light of the preceding fact, to get an intuition for the notion of tree depth, we have a look at some undirected graphs.

EXAMPLE 3.14. The notion of the tree depth of a graph can be seen as measuring the distance of a graph to being an *independent set*, a graph without any edges. Independent sets have tree depth 1, td( ) = 1, while all *stars* have tree depth 2, td() = 2, which follows from Definition 3.12 by deleting the *center vertex* that is connected to all other vertices of the graph, and producing an independent set. A slightly more complicated example is td() = 3, where the upper bound on the tree depth can be seen by deleting the vertex in the middle, which produces a graph whose components are stars. For any graph class C of bounded tree depth, Definition 3.12 states that graphs from C can be split into graphs of ever smaller tree depth by and deleting vertices from connected components using a constant number of parallel steps. This is not possible for graph classes whose tree depth is not bounded. Such graph classes of *unbounded tree depth* include the class of all *cliques*, the tree depth of a clique is the number of its vertices (thus, td() = 5), and the class of all *paths*, the tree depth of an n-vertex path is $\lfloor \log_2(n) \rfloor + 1$ (thus, td() = 3).

Alternative characterizations of when a class C of undirected graphs has bounded tree depth include: (a) There is a constant that bounds the length of the paths from the graphs of C. (b) The graphs of C have only bounded-depth depth-first search trees. In contrast, graphs with bounded-depth breath-first search trees are exactly the graphs with bounded diameter; like cliques, which have diameter 1, such graphs may have an unbounded tree depth. (c) A third alternative definition in terms of tree decompositions of bounded width whose underlying trees have bounded depth will be discussed and used in Section 4.2.

For defining tree depth using first-order formulæ, we use one direction of alternative definition (a). The maximum length of a path in a graph $\mathcal{G}$ is called its *longest path length* and denoted by $\operatorname{lpl}(\mathcal{G})$.

FACT 3.15 (Nešetřil and Ossona de Mendez (2008)). *For undirected graphs $\mathcal{G}$ we have* $\operatorname{lpl}(\mathcal{G}) \leq 2^{\operatorname{td}(\mathcal{G})} - 2$.

First-Order Definability of Tree Depth. In this section we prove that, for any vocabulary τ and $d \in \mathbb{N}$, the class of τ-structures of tree depth at most d is first-order definable. The formulæ we construct mimic the recursive definition of tree depth. In order to work with the components that arise during the element deletion process, we define descriptors: Let $\mathcal{A}$ be a structure and $w \in \mathbb{N}_0$ a *width bound*. A *width-w descriptor D in $\mathcal{A}$* consists of *w bag elements* $b_1, \dots, b_w \in A$ and a *(component) selector* $s \in A$. It *selects* the component $\mathcal{C}_D$ with universe C_D of $\mathcal{A}[A \setminus \{b_1, \dots, b_w\}]$ that contains s and *describes* $\mathcal{A}_D := \mathcal{A}[C_D \cup \{b_1, \dots, b_w\}]$ with universe $A_D := C_D \cup \{b_1, \dots, b_w\}$. If, in addition, a variable assignment α for $\mathcal{A}$ is given, we define the *restriction of α to $\mathcal{A}_D$* by $\alpha_D(X) := \alpha(X) \cap A_D$ for each set variable X of α's domain.

LEMMA 3.16 (Compare Components). *Let $\ell \in \mathbb{N}_0$ be a path length bound and $w \in \mathbb{N}_0$ a width bound. For every vocabulary τ, there is a first-order formula compare$_\ell(x_1, \dots, x_w, y_s, y_t)$ over τ, such that for all τ-structures $\mathcal{A}$ with $\operatorname{lpl}(G(\mathcal{A})) \leq \ell$, width-$w$ descriptors $D = (b_1, \dots, b_w, s)$ in $\mathcal{A}$ and elements $t \in A$, we have $\mathcal{A} \models$ compare$_\ell(b_1, \dots, b_w, s, t)$ if, and only if, $t \in C_D(b_1, \dots, b_w, s)$.*

Proof. The formula tests whether there is a path from s to t in the Gaifman graph $G(\mathcal{A})$ that does not go through the elements b_1 to b_w. Since the length of simple paths in $G(\mathcal{A})$ is bounded by the constant ℓ, this reachability test can be defined using first-order formulæ. The formula has access to $G(\mathcal{A})$ since, for any vocabulary τ, there is a first-order formula $\varphi_E(x, x')$ over τ that defines the Gaifman graph of τ-structures $\mathcal{A}$. That means, for every pair of elements $(a, a') \in A \times A$ we have $\mathcal{A} \models \varphi_E(a, a')$ exactly if (a, a') is an edge of $G(\mathcal{A})$. $\square$

The previous lemma can also be seen as testing, for two descriptors $D = (b_1, \dots, b_w, s)$ and $D' = (b_1, \dots, b_w, t)$ with the same bag elements, but possibly different selectors, whether $\mathcal{C}_D = \mathcal{C}_{D'}$ and, thus, $\mathcal{A}_D = \mathcal{A}_{D'}$ holds.

Lemma 3.17 (Define Tree Depth for Selected Components). *Let $\ell \in \mathbb{N}_0$ be a path length bound and τ be a vocabulary. For every tree depth bound $d \in \mathbb{N}$ and width bound $w \in \mathbb{N}_0$, there is a first-order formula $\varphi_d(x_1, \dots, x_w, y_s)$ over τ, such that for all τ-structures $\mathcal{A}$ with $\operatorname{lpl}(G(\mathcal{A})) \leq \ell$ and width-w descriptors $D = (b_1, \dots, b_w, s)$ in $\mathcal{A}$, we have $\mathcal{A} \models \varphi_d(b_1, \dots, b_w, s)$ if, and only if, $\operatorname{td}(\mathcal{C}_D) \leq d$.*

Proof. Consider any vocabulary τ and path length bound $\ell \in \mathbb{N}_0$ that are fixed throughout the proof. We prove the lemma by induction over d for all w. The formula for $d = 1$ is $\varphi_d(x_1, \dots, x_w, y_s) := \neg\exists y_t(y_s \neq y_t \land \mathit{compare}_\ell(x_1, \dots, x_w, y_s, y_t))$, which tests whether C_D contains only the single element s. For $d > 1$, we use $\varphi_d(x_1, \dots, x_w, y_s) :=$

$$\exists x_{w+1}\Big(\mathit{compare}_\ell(x_1, \dots, x_w, y_s, x_{w+1}) \land$$
$$\forall y_t\big(\mathit{compare}_\ell(x_1, \dots, x_w, y_s, y_t) \land x_{w+1} \neq y_t \to \varphi_{d-1}(x_1, \dots, x_{w+1}, y_t)\big)\Big)\ ,$$

where $\mathit{compare}_\ell(x_1, \dots, x_w, y_s, x_{w+1})$ and $\mathit{compare}_\ell(x_1, \dots, x_w, y_s, y_t)$ require that the element b_{w+1} that is chosen for x_{w+1} and the element t that is chosen for y_t are from $\mathcal{C}_{(b_1,\dots,b_w,s)}$. The first-order formula $\varphi_{d-1}(x_1, \dots, x_{w+1}, y_t)$ from the induction hypothesis correctly decides whether $\mathcal{C}_{(b_1,\dots,b_w,b_{w+1},t)} \leq d-1$ holds. Correctness for the whole formula follows from the observation that it exactly mimics the recursive definition of the tree depth of connected structures: A connected structure, in our case the selected component $\mathcal{C}_{(b_1,\dots,b_w,s)}$, has tree depth at most d exactly if there exists an element, in our case b_{w+1}, whose deletion produces components, in our case the selected components $\mathcal{C}_{(b_1,\dots,b_w,b_{w+1},t)}$, of tree depth at most $d - 1$. $\square$

Lemma 3.18 (First-Order Definability of Tree Depth). *For every vocabulary τ and every $d \in \mathbb{N}$, there is a first-order formula $\varphi_{\mathrm{td} \leq d}$ over τ, such that for every τ-structure $\mathcal{A}$ we have $\mathcal{A} \models \varphi_{\mathrm{td} \leq d}$ if, and only if, $\operatorname{td}(\mathcal{A}) \leq d$.*

Proof. The formula first tests if all paths in $G(\mathcal{A})$ have length at most $2^d - 2$. If this does not hold, then, by Fact 3.13 and Fact 3.15, the tree depth of $\mathcal{A}$ exceeds d and we are done. Otherwise, consider the formula $\varphi_d(x_s)$ for the width bound $w = 0$ and path length bound $\ell = 2^d - 2$ from Lemma 3.17. The lemma is proved by the formula $\varphi_{\mathrm{td} \leq d} := \forall x_s\, \varphi_d(x_s)$, which tests whether all components of $\mathcal{A}$, the components that are selected by any width-0 descriptor $D = (s)$ in $\mathcal{A}$, have tree depth at most d. $\square$

3.3. Evaluating Monadic Second-Order Formulæ

If we could just translate tree-depth-bounded structures into equivalent depth-bounded trees using first-order formulæ, we would be able to simulate any MSO-formula by a first-order formula using Lemma 2.21. Unfortunately, this is not possible using first-order formulæ: Defining, for example, a unique root element for a tree from a graph cannot be done with first-order formulæ since there may be multiple nodes having the same properties with respect to first-order formulæ. An example of such graphs are cliques: Every first-order formula that is constructed with the intention to define a unique root element is true either for all or none of the vertices in a clique. In the present section, we work around this problem and, instead of constructing trees, evaluate MSO-formulæ on top of the element deletion process from the previous section that defines tree depth.

We start with a technical lemma that shows how to test first-order properties for substructures that are described. Then, we use this lemma to prove Theorem 3.1 for described structures and, finally, prove Theorem 3.1, itself.

LEMMA 3.19 (From Structures to Described Structures). *Let $\ell \in \mathbb{N}_0$ be a path length bound and $w \in \mathbb{N}_0$ a width bound. For every first-order formula $\varphi(X_1, \dots, X_k)$ over some vocabulary τ, there is a first-order formula $\varphi_\ell(x_1, \dots, x_w, y_s, X_1, \dots, X_k)$, such that for all τ-structures $\mathcal{A}$ with $\mathrm{lpl}(G(\mathcal{A})) \leq \ell$, width-$w$ descriptors $D = (b_1, \dots, b_w, s)$ in $\mathcal{A}$, and variable assignments α with domain $\{X_1 \dots, X_k\}$, we have*

$$(\mathcal{A}, \alpha[x_1 \mapsto b_1] \dots [x_w \mapsto b_w][y_s \mapsto s]) \models \varphi_\ell \text{ if, and only if, } (\mathcal{A}_D, \alpha_D) \models \varphi .$$

PROOF. The formula φ_ℓ needs to ensure that the evaluation of φ only depends on the elements of $\mathcal{A}_D$. For this, we inductively replace every subformula $\exists y\, \psi$ of φ by $\exists y((y = x_1 \vee \dots \vee y = x_w \vee \mathit{compare}_\ell(x_1, \dots, x_w, y_s, y)) \wedge \psi)$ where $\mathit{compare}_\ell$ is from Lemma 3.16. With this modification all valid interpretations of y are elements of $\mathcal{A}_D$. □

LEMMA 3.20 (Define Type of Described Structures). *Let $\ell \in \mathbb{N}_0$ be a path length bound. For every tree depth bound $d \in \mathbb{N}$, width bound $w \in \mathbb{N}_0$, and type $\theta \in \Theta_{k,q}[\tau]$ for some $k, q \in \mathbb{N}_0$ and vocabulary τ, there is a first-order formula $\varphi_{d,\theta}(x_1, \dots, x_w, y_s, X_1, \dots, X_k)$ over τ, such that for every τ-structure $\mathcal{A}$ with $\mathrm{lpl}(G(\mathcal{A})) \leq \ell$, variable assignment α with domain $\{X_1, \dots, X_k\}$ for $\mathcal{A}$, and every width-w descriptor $D = (b_1, \dots, b_w, s)$ in $\mathcal{A}$ with $\mathrm{td}(\mathcal{C}_D) \leq d$, we have*

$$\mathrm{tp}_q(\mathcal{A}_D, \alpha_D) = \theta \text{ if, and only if, } (\mathcal{A}, \alpha[x_1 \mapsto b_1] \dots [x_w \mapsto b_w][y_s \mapsto s]) \models \varphi_{d,\theta} .$$

PROOF. Let $\ell \in \mathbb{N}_0$. The proof is by induction over the tree depth bound d for all width bounds w and types θ.

For the induction start consider $d = 1$. Let $w \in \mathbb{N}_0$, and $\theta \in \Theta_{k,q}[\tau]$. By Lemma 3.4, we can define the set of structures and variable assignments of type θ using an MSO-formula. Since $\mathcal{C}_D$ contains exactly one element in this case, which is s, the universes of the considered structures $\mathcal{A}_D$ have at most $w{+}1$ elements, which is a constant. When considering only structures of constant size, MSO-formulæ can be transformed into equivalent first-order formulæ via replacing set quantifier by equivalent sequences of element quantifiers. Using Lemma 3.19, we know that $\varphi_{1,\theta}$ exists.

For the induction step, let $d > 1$. The formula we construct computes the type of $(\mathcal{A}_D, \alpha_D)$ based on recursively computing types for substructures and combining them with the composition theorem. We take the formula ρ_θ from Theorem 3.10 and adjust it to work on $\mathcal{A}$ instead of a type indicator. For $w \in \mathbb{N}_0$ and $\theta \in \Theta_{k,q}[\tau]$, we set $\varphi_{d,\theta}(x_1, \dots, x_w, y_s, X_1, \dots, X_k) :=$

$$\begin{aligned}\exists x_{w+1}\big(&\mathit{compare}_\ell(x_1, \dots, x_w, y_s, x_{w+1}) \land \\ &\forall y_t\big(\mathit{compare}_\ell(x_1, \dots, x_w, y_s, y_t) \land x_{w+1} \neq y_t \to \varphi_{d-1}(x_1, \dots, x_{w+1}, y_t)\big) \land \\ &\eta_{d,\theta}(x_1, \dots, x_{w+1}, y_s, X_1, \dots, X_k)\big)\,.\end{aligned}$$

The first part of the formula tests whether there is an element $b_{w+1} \in C_D$ whose removal makes $\mathcal{C}_D$ having tree depth at most $d - 1$. Since $\mathcal{C}_D$ is a connected structure of tree depth at most d, we know that such an element exists.

To describe the construction of the formula $\eta_{d,\theta}$ and prove its correctness, we use the following notations: For any b_{w+1} that is chosen for x_{w+1}, we let $\mathcal{B}$ be the rooted structure that arises from $\mathcal{A}_D$ by adding the monadic relations $B_1^{\mathcal{B}} = \{b_1\}$ to $B_{w+1}^{\mathcal{B}} = \{b_{w+1}\}$ to it. Let F be the rooted partition of $\mathcal{B}$ whose parts are exactly the structures that can be described by some descriptor $D' = (b_1, \dots, b_{w+1}, t)$ with $t \in C_D$. Together with an assignment α, let $\mathcal{I}$ be the corresponding indicator structure.

The formula $\eta_{d,\theta}(x_1, \dots, x_{w+1}, y_s, X_1, \dots, X_k)$ arises by rewriting the formula ρ_θ over $\tau_{k,q,w+1}$ from Theorem 3.10 into a formula over τ. For this, we start to replace the atomic formulæ of ρ_θ by atomic formulæ over τ: We replace every occurrence of $x = y$ by the formula $\mathit{compare}_\ell(x_1, \dots, x_{w+1}, x, y)$ from Lemma 3.16. The original formula $x = y$ is true if the indices from $\mathcal{I}$ that are assigned to x and y are the same and, hence, $\mathcal{A}_x = \mathcal{A}_y$. The replaced formula tests the same property based on elements from the parts of F. It is true for two elements t_i and t_j from $\mathcal{C}_{(b_1,\dots,b_w,s)}$ that are not b_{w+1} if we have $\mathcal{A}_{(b_1,\dots,b_w,b_{w+1},t_i)} = \mathcal{A}_{(b_1,\dots,b_w,b_{w+1},t_j)}$. Next, we replace every atomic formula $R_{\theta'}(x)$ by the formula $\varphi_{d-1,\theta'}(x_1, \dots, x_{w+1}, x, X_1, \dots, X_k)$ from the induction hypothesis; instead of accessing the type of some part of F via an index and the type indicator $\mathcal{I}$, this formula defines the type on the structure itself. Finally, recursively replace every subformula $\exists y\, \varphi$ by $\exists y(\mathit{compare}_\ell(x_1, \dots, x_w, y_s, y) \land x_{w+1} \neq y \land \varphi)$; with this adjustment we transfer quantifications over indices of the type indicator $\mathcal{I}$ that address components of F to quantifications of elements from $\mathcal{C}_D$ that the formula uses to select components. Correctness follows from the induction hypothesis, that the types of the parts of F are defined correctly, and Theorem 3.10, that the types of the parts are combined correctly. □

LEMMA 3.21. *For every tree depth bound $d \in \mathbb{N}$, and type $\theta \in \Theta_{k,q}[\tau]$, there is a first-order formula $\varphi_{d,\theta}(X_1, \dots, X_k)$ over τ, such that for every τ-structure $\mathcal{A}$ and variable assignment α with domain $\{X_1, \dots, X_k\}$ for it we have*

$$\mathrm{tp}_q(\mathcal{A}, \alpha) = \theta \textit{ if, and only if, } (\mathcal{A}, \alpha) \models \varphi_{d,\theta}.$$

PROOF. The formula $\varphi_{d,\theta}(X_1, \dots, X_k)$ is constructed like the formula from the previous lemma, except that we do not use a variable x_{w+1} that binds an element b_{w+1} whose removal decreases the tree depth. Instead, we take the formula ρ_θ for $w = 0$ from the composition theorem and rewrite it, such that it defines the types of the components of $\mathcal{A}$ using formulæ $\varphi_{d,\theta'}(y_s, X_1, \dots, X_k)$, which work on the structure $\mathcal{A}$, instead of using predicates $R_{\theta'}(x)$, which work on an indicator structure $\mathcal{I}$. $\square$

Proof of Theorem 3.1. We set $\psi(X_1, \dots, X_k) :=$

$$\bigvee_{\theta \in \Theta_{k,q}[\tau] \text{ s.t. } \varphi \in \theta} \varphi_{d,\theta}(X_1, \dots, X_k)$$

where the formulæ $\varphi_{d,\theta}(X_1, \dots, X_k)$ are from the previous lemma. $\square$

3.4. Application to Evaluating Guarded Second-Order Formulæ

In this section we extend Theorem 3.1 from MSO-logic to guarded second-order logic (GSO-logic). We first review the definition of GSO-logic and, then, discuss how results from the literature can be used to generalize Theorem 3.1.

Getting Familiar with Guarded Second-Order Logic. Like monadic second-order logic, guarded second-order logic can be defined by considering second-order formulæ of a restricted syntax. *Guarded second-order formulæ (GSO-formulæ)* are all second-order formulæ where free and bound second-order variables of arity higher than 1 are guarded. For any vocabulary τ and second-order variable X with $\mathrm{ar}(X) \geq 2$, we define

$$\mathit{guard}(X) := \forall x_1, \dots, x_{\mathrm{ar}(X)} \Big(X(x_1, \dots, x_{\mathrm{ar}(X)}) \rightarrow \bigvee_{R \in \tau} \exists y_1, \dots, y_{\mathrm{ar}(R)} \Big(R(y_1, \dots, y_{\mathrm{ar}(R)}) \wedge \bigwedge_{i=1}^{\mathrm{ar}(X)} \bigvee_{j=1}^{\mathrm{ar}(R)} x_i = y_j \Big) \Big) .$$

Each quantification of a second-order variable X with $\mathrm{ar}(X) \geq 2$ in a guarded second-order formulæ must be of the form $\exists X(\mathit{guard}(X) \wedge \varphi)$, and formulæ with free second-order variables X_1 to X_k of arity at least 2 must be of the form $\bigwedge_{i \in \{1,\dots,k\}} \mathit{guard}(X_i) \wedge \varphi$. The formula $\mathit{guard}(X)$ restricts the interpretation of the variable X in the following way: Whenever a tuple of elements $(a_1, \dots, a_{\mathrm{ar}(X)})$ is part of the relation that is assigned to X, then the elements a_1 to $a_{\mathrm{ar}(X)}$ are present in a single tuple that is part of a relation of the input structure; such interpretations of X are called *guarded relations*. Equivalently, GSO-logic can be defined by taking all formulæ of second-order logic without restricting the syntax, but defining the semantics via *guarded variable assignments* that assign only guarded relations to free and bound second-order variables (Grädel et al., 2002).

GSO-logic restricted to graph structures over $\tau_{\text{graph}} = \{E^2\}$ is sometimes denoted by MSO_2-logic, and MSO-logic for graph structures by MSO_1-logic, to indicate that GSO-formulæ are able to quantify over 2-ary subsets of edges, while MSO-formulæ can quantify only over 1-ary subsets of the vertices. The following examples show GSO-formulæ that define matchings and paths in graphs.

Example 3.22 (Defining Matchings with GSO-formulæ). We define matchings in undirected graphs $\mathcal{G} = (V, E^{\mathcal{G}})$. Let M be a 2-ary second-order variable, for which guarded relations are always subsets of the edges $E^{\mathcal{G}}$. To define that M is a guarded set of undirected edges we use the formula $\mathit{undirected}(M) := \mathit{guard}(M) \wedge \forall x \forall y\, (M(x,y) \leftrightarrow M(y,x))$. Then the formula $\varphi_{\text{matching}}(M) := \mathit{undirected}(M) \wedge \forall x \forall y \forall z\, (M(x,y) \wedge M(x,z) \rightarrow y = z)$ defines matchings; that means sets of edges with pairwise disjoint end vertices. This formula can be extended to define perfect matchings, sets of edges that partition the vertices, using the formula $\varphi_{\text{perfect-matching}}(M) := \varphi_{\text{matching}}(M) \wedge \forall x \exists y\, M(x,y)$. Finally $\exists M\, \varphi_{\text{perfect-matching}}(M)$ is a GSO-formula without free variables that defines the undirected graphs with a perfect matching.

EXAMPLE 3.23 (Defining Paths with GSO-formulæ). We build GSO-formulæ that define paths in graphs. Let W be a set variable and F be a 2-ary second-order variable. Let $\varphi_{\text{path}}(W,F) :=$

$$\begin{aligned}
&\mathit{guard}(F) \wedge \\
&\forall v \left(\exists w\, (F(v,w) \vee F(w,v)) \leftrightarrow W(v) \right) \wedge && (1)\\
&\exists s\, \exists t\, \Big(W(s) \wedge W(t) \wedge \neg \exists w\, F(w,s) \wedge \neg \exists w\, F(t,w) \wedge \\
&\quad \forall v \left(W(v) \to (v \neq s \to \exists! w\, F(w,v)) \wedge (v \neq t \to \exists! w\, F(v,w)) \right) \wedge && (2)\\
&\quad \forall C \left(C(s) \wedge \forall x \forall y\, (C(x) \wedge F(x,y) \to C(y)) \to \mathit{subset}(W,C) \right) \Big) \ , && (3)
\end{aligned}$$

where the formula $\mathit{subset}(W,C)$ tests whether W is a subset of C. The formula $\varphi_{\text{path}}(W,F)$ defines all paths inside graphs that are not cycles. That means, for a given graph $\mathcal{G} = (V, E^{\mathcal{G}})$ and assignment α, we have $(\mathcal{G}, \alpha) \models \varphi_{\text{path}}(W,F)$ exactly if there is a path with vertex set $\alpha(W)$ and edge set $\alpha(F)$ in $\mathcal{G}$. This is due to the following reasons: Part (1) of the formula states that $\alpha(W)$ is exactly the set of vertices that are present in the tuples of $\alpha(F)$. Thus, $(\alpha(W), \alpha(F))$ is a subgraph of $\mathcal{G}$. Part (2) states that $(\alpha(W), \alpha(F))$ contains a vertex s with in-degree 0 and out-degree 1 and a vertex t with in-degree 1 and out-degree 0, and all other vertices have in-degree 1 and out-degree 1. Thus, using the parts (1) and (2) in conjunction, we know that $(\alpha(W), \alpha(F))$ is the disjoint union of a single path that starts in s and ends in t, and cycles. Together with part (3), we know that $\alpha(W)$ equals the set of vertices that are reachable from s using edges from $\alpha(F)$. As a result, cycles are not possible anymore and, thus, $(\alpha(W), \alpha(F))$ is a path from s to t.

We can play around with $\varphi_{\text{path}}(W,F)$ to define paths of various kind. For example, we can define Hamiltonian paths using the formula

$$\varphi_{\text{ham-path}} := \varphi_{\text{path}}(W,F) \wedge \forall w\, W(w) \ ,$$

which requires that all vertices are present in $\alpha(W)$. By extending it to the formula $\exists W \exists F\, \varphi_{\text{ham-path}}(W,F)$ without free variables, we can define the set of graphs that contain Hamiltonian paths.

We can also use the basic path-defining formula to define paths between given vertices. For this, we consider structures $\mathcal{G} = (V, E^{\mathcal{G}}, S^{\mathcal{G}}, T^{\mathcal{G}})$ over the vocabulary $\tau_{s\text{-}t\text{-graph}} = \tau_{\text{graph}} \cup \{S^1, T^1\}$ with $|S^{\mathcal{G}}| = |T^{\mathcal{G}}| = 1$. For such structures, we can build formulæ that make statements about paths that start in the unique vertex $s \in S^{\mathcal{G}}$ and end in the unique vertex $t \in S^{\mathcal{G}}$. The formula

$$\begin{aligned}
\varphi_{s\text{-}t\text{-path}}(W,F) := \varphi_{\text{path}}(W,F) \wedge \forall s \left(S(s) \to W(s) \wedge \neg \exists w\, F(w,s) \right) \wedge \\
\forall t \left(T(t) \to W(t) \wedge \neg \exists w\, F(t,w) \right)
\end{aligned}$$

defines exactly the paths from $s \in S^{\mathcal{G}}$ to $t \in S^{\mathcal{G}}$.

From Guarded to Monadic Second-Order Logic. As shown in the previous examples, GSO-formulæ can define graphs with perfect matchings and graphs with Hamiltonian paths. These properties are not definable by MSO-formulæ in general (see the book of Libkin (2006) for detailed arguments), but become MSO-definable when formulæ have access to incidence representations of graphs. The following definition is adapted from Grädel et al. (2002) and Blumensath et al. (2007): Let τ be a vocabulary. The *incidence representation* of a τ-structure $\mathcal{A}$ is another structure $\mathcal{A}_{\text{inci}}$ that is defined as follows: The universe of $\mathcal{A}_{\text{inci}}$ contains all elements $a \in A$ and an element $t_{(a_1,\dots,a_r)}$ for each tuple $(a_1,\dots,a_r) \in R^{\mathcal{A}}$ with $R^r \in \tau$. The relations are as follows: It has two monadic relations $U^{\mathcal{A}_{\text{inci}}}$ and $T^{\mathcal{A}_{\text{inci}}}$ to distinguish between the elements from the universe and the tuples from the relations. That means, $U^{\mathcal{A}_{\text{inci}}} := A$ and $T^{\mathcal{A}_{\text{inci}}}$ contains all elements from the universe of $\mathcal{A}_{\text{inci}}$ that are not in A. Moreover, it has binary relations $R_1^{\mathcal{A}_{\text{inci}}}$ to $R_r^{\mathcal{A}_{\text{inci}}}$ for each $R^r \in \tau$ with $(a, t_{(a_1,\dots,a_r)}) \in R_i^{\mathcal{A}_{\text{inci}}}$ exactly if $(a_1,\dots,a_r) \in R^{\mathcal{A}_{\text{inci}}}$ and $a = a_i$. Variable assignments α for $\mathcal{A}$ are adjusted accordingly to assignments α_{inci} for $\mathcal{A}_{\text{inci}}$: $\alpha_{\text{inci}}(X)$ is just $\alpha(X)$ if X is a set variable and the set $\{t_{(a_1,\dots,a_{\text{ar}(X)})} \mid (a_1,\dots,a_{\text{ar}(X)}) \in \alpha(X)\}$ if $\text{ar}(X) \geq 2$. The difference between usual structures and their incidence representation is best seen for graphs: While a graph structure $\mathcal{G} = (V, E^{\mathcal{G}})$ encodes the *adjacency relation among vertices*, $\mathcal{G}_{\text{inci}}$ encodes the *incidence relation between vertices and edges*. MSO-formulæ with access to the incidence representation of graph structures are able to define matchings and paths.

EXAMPLE 3.24 (Defining Matchings and Paths with MSO-formulæ). The GSO-formulæ on graphs $\mathcal{G} = (V, E^{\mathcal{G}})$ from the Examples 3.22 and 3.23 can be turned into MSO-formulæ on incidence representations $\mathcal{G}_{\text{inci}}$ that define the same properties: For this, we replace every 2-ary second-order variable X by a 1-ary second-order variable X' and replace each subformula *guard*(X) by the formula *subset*(X', T), which ensures that only edge-representing elements can be part of the set for X'. Moreover, each query $Z(x, y)$ to a 2-ary second-order variable or relation is replaced by $\varphi_Z(x, y) := \exists t\,(Z(t) \land E_1(x, t) \land E_2(y, t))$. Using these replacements, every GSO-formula φ over graph structures is transformed into an equivalent MSO-formula φ' over incidence representations of graphs. For example, for every graph $\mathcal{G}$ and assignment α, we have $(\mathcal{G}, \alpha) \models \varphi_{\text{matching}}$ if, and only if, $(\mathcal{G}_{\text{inci}}, \alpha_{\text{inci}}) \models \varphi'_{\text{matching}}$. In the same way, we can define all other properties from the Examples 3.22 and 3.23 by MSO-formula that access the incidence representations of graphs.

The phenomenon discussed in the previous example can be generalized to structures and GSO-formulæ over any vocabulary, as formally proved by Grädel, Hirsch, and Otto (2002):

FACT 3.25. *For every* GSO*-formula* $\varphi(X_1,\dots,X_k)$ *over structures of some vocabulary* τ*, there exists an* MSO*-formula* $\varphi(X_1',\dots,X_k')$ *over incidence representations of* τ*-structures, such that for every* τ*-structure* $\mathcal{A}$ *and variable assignment* α *we have* $(\mathcal{A}, \alpha) \models \varphi(X_1,\dots,X_k)$ *if, and only if,* $(\mathcal{A}_{\text{inci}}, \alpha_{\text{inci}}) \models \varphi(X_1',\dots,X_k')$.

For evaluating GSO-formulæ by logspace Turing machines or uniform circuit families, we can always restrict to prove evaluating MSO-formulæ, since each of these computing devices can easily turn the encoding of a given structure into the encoding of its incidence representation and, then, consider the equivalent MSO-formula from the previous fact on such structures. Moreover, the tree depth (tree width) of an incidence representation $\mathcal{A}_{\text{inci}}$ is bounded by 1 plus the tree depth (tree width) of its original structure $\mathcal{A}$. Thus, all theorems that are proved in the upcoming Chapters 4 to 6 for MSO-logic directly generalize to GSO-logic. In contrast, it is far from being obvious how to translate structures into their incidence representation when the only device available is first-order logic without any build-in ordering or arithmetic predicates. Fortunately, we are able to plug-in the following result from the literature for this, which is proved for graphs by Courcelle (2003) and for structures over any vocabulary by Blumensath, Colcombet, and Löding (2007) via defining the incidence representation of a structure from the structure itself using MSO-formulæ. A structure $\mathcal{A}$ with universe A is *uniformly s-sparse* for $s \in \mathbb{N}$ if the following holds: for each subgraph $\mathcal{G}$ of the Gaifman graph $G(\mathcal{A})$ we have $|E(\mathcal{G})| \leq s\,|V(\mathcal{G})|$.

FACT 3.26. *For every vocabulary τ, every $s \in \mathbb{N}$, and every GSO-formula $\varphi(X_1, \dots, X_k)$ over τ, there is an MSO-formula $\psi(X_1, \dots, X_k)$ (where the X_1 to X_k may be higher-ary second-order variables) over τ, such that for every uniformly s-sparse τ-structure $\mathcal{A}$ and guarded variable assignment α, we have*

$$(\mathcal{A}, \alpha) \models \varphi \textit{ if, and only if, } (\mathcal{A}, \alpha) \models \psi \ .$$

Since structures of tree depth d are uniformly $(d-1)$-sparse (which is a well known fact that can be proved by induction over the tree depth), we can apply Theorem 3.1 to the previous fact and get the following:

THEOREM 3.27. *For every vocabulary τ, every tree depth bound $d \in \mathbb{N}$, and every GSO-formula $\varphi(X_1, \dots, X_k)$ over τ, there is a first-order formula $\psi(X_1, \dots, X_k)$ (where the X_1 to X_k may be higher-ary second-order variables) over τ, such that for every τ-structure $\mathcal{A}$ of tree depth at most d and guarded variable assignment α, we have*

$$(\mathcal{A}, \alpha) \models \varphi \textit{ if, and only if, } (\mathcal{A}, \alpha) \models \psi \ .$$

Elberfeld, Grohe, and Tantau (2012a) prove Theorem 3.27 for graph structures using a composition theorem that is based on types with respect to GSO-logic, instead of using the results of Courcelle (2003) and Blumensath, Colcombet, and Löding (2007).

CHAPTER 4

Bounded Tree Depth and Constant-Depth Circuits

In this chapter we prove the theorems from the introduction that are related to tree-depth-bounded structures and constant-depth circuits. We will show how to solve MSO-definable decision, counting, and optimization problems on tree-depth-bounded structures using constant-depth DLOGTIME-uniform Boolean, arithmetic, and threshold circuits, respectively.

In Section 4.1, we review the definition of DLOGTIME-uniform constant-depth circuit families and their equivalent definition in terms of first-order formulæ that have access to ordering and arithmetic predicates defined on input strings. Since relational structures can be extracted from their string encodings using such first-order computations, Theorem 1.6—solving MSO-definable decision problems on tree-depth-bounded structures using constant-depth Boolean circuits—follows from Theorem 1.9.

Section 4.2 starts to define the notion of tree decompositions. Then we show how tree decompositions of bounded width whose underlying trees have bounded depth are computed using first-order computations. Once tree decompositions are available, we can adjust the original MSO-formula to an equivalent formula on the computed tree. This allows us to replace the task of computing solution histograms for MSO-formulæ on structures over any vocabulary by the more manageable problem of computing solution histograms for MSO-formulæ on trees.

Section 4.3 contains the proof of how to reduce computing the number of ways in which multiset tree automata accept an input tree to evaluating arithmetic circuits. Since counting solution histograms for MSO-formulæ on trees can be reduced to the problem of evaluating multiset tree automata, this leads to the proof of Theorems 1.7 and 1.8, computing number and string representations, respectively, of histograms on tree-depth-bounded structures. In the course of Section 4.3, we address the problem of how histograms can be encoded as numbers. As we will see, by using an appropriate encoding, we may assume that the formulæ in the theorems from the introduction are all of the form $\varphi(X_1, \dots, X_k)$. That is, we may assume that no variables Y_i are present. This is why the lemmas and theorems of the present chapter, and Chapters 5 and 6, are all formulated without references to any Y_j.

In Section 4.4, we apply the theorems to concrete problems. In particular, it is shown that integer linear equation systems with any constant number of equations whose coefficients are encoded in unary can be solved in DLOGTIME-uniform TC^0. This implies the TC^0-completeness of such well studied problems like SUBSETSUM and KNAPSACK for input numbers that are given in unary.

4.1. Review of Uniform Circuits

In the present section, we first review the definition of uniform circuit complexity classes inside logspace. Beside constant-depth circuits that are used in the present chapter, we define logarithmic-depth circuits that are used in Chapter 5. Then we have a look at used reducibility notions. Finally, we discuss how to encode relational structures as strings.

Circuit Classes. In the present and the next chapter, we are concerned with uniform circuit complexity classes that are inside logspace. The most widely known *language classes* we consider are NC^1, the class of decision problems $L \subseteq \{0,1\}^*$ that are decidable by Boolean circuit families of logarithmic depth and polynomial size with input gates $\{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}$ and bounded fan-in inner gates from $\{\wedge, \vee, 0, 1\}$; AC^0, the class of decision problems $L \subseteq \{0,1\}^*$ decidable by Boolean circuit families of constant depth and polynomial size with input gates $\{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}$ and unbounded fan-in inner gates from $\{\wedge, \vee, 0, 1\}$; and TC^0, whose definition is the same as AC^0, except that we also allow unbounded fan-in gates that decide $\textsc{majority} := \{y_1 \dots y_n \in \{0,1\}^* \mid \sum_{i \in \{1,\dots,n\}} y_i > n/2\}$. Note that we do not allow $\neg$-gates to be located in the inner part of the circuit; they are only applied to input gates directly, but this does not restrict the computational power of the uniform circuit classes we consider. We also consider the *function class* variants of AC^0, TC^0, and NC^1, known as FAC^0, FTC^0, and FNC^1, respectively. Each function class is defined in the same way as its corresponding language class, but with Boolean circuit families computing general mappings $f\colon \{0,1\}^* \to \{0,1\}^*$ instead of characteristic functions $\chi_L\colon \{0,1\}^* \to \{0,1\}$ of languages $L \subseteq \{0,1\}^*$. Circuits whose inputs are strings over non-binary alphabets Σ access atomic predicates $[x_i = \sigma]$ for $\sigma \in \Sigma$, instead of the x_i directly; such a predicate evaluates to 1 if $x_i = \sigma$ holds and to 0, otherwise (Vollmer, 1999).

The classes AC^0 and NC^1 are *arithmetized* by replacing, in the definition of each class, the admitted input gates by $\{x_1, \dots, x_n, 1 - x_1, \dots, 1 - x_n\}$ and the Boolean inner gates by arithmetic gates $\{\times, +, 0, 1\}$. This leads to circuit families that compute functions $f\colon \{0,1\}^* \to \mathbb{N}_0$; the resulting classes are called $\#\mathrm{AC}^0$ and $\#\mathrm{NC}^1$. If we also allow the constant -1 as an inner gate, the circuit families compute functions $f\colon \Sigma^* \to \mathbb{Z}$; the corresponding classes are known as GapAC^0 and GapNC^1. Vollmer (1999) calls these circuits *counting arithmetic circuits* and the classes *counting arithmetic classes* due to their equivalent definitions in terms of counting the number of accepting proof trees of Boolean circuits. To study the relation between language classes defined via Boolean circuits and their arithmetic variants, the following language classes are known from the literature: PNC^1 is the class of all languages $L \subseteq \{0,1\}^*$, such that there exists a function $f \in \mathrm{GapNC}^1$ with $x \in L$ if, and only if, $f(x) > 0$ for all $x \in \{0,1\}^*$. The corresponding variant of AC^0, the class PAC^0, is defined in the same way, but with respect to GapAC^0. Note that, if we would define such classes with respect to arithmetic circuits from $\#\mathrm{AC}^0$ and $\#\mathrm{NC}^1$, we get AC^0 and NC^1, respectively, again; the question of whether a $\#\mathrm{NC}^1$-circuit has a nonzero output can be answered by an NC^1-circuit that arises from it via replacing $\times$- and $+$-gates by $\wedge$- and $\vee$-gates, respectively. We use this fact in Chapter 5 where we first prove Theorem 1.5 for $\#\mathrm{NC}^1$ and, then, get Theorem 1.4 for NC^1 as a corollary.

Uniformity. We use the above complexity classes in their DLOGTIME-uniform variants, as defined by Mix Barrington et al. (1990). For constant-depth circuit families with gates of unbounded fan-in, this corresponds to the notion that their *direct connection languages*, a language of tuples that describe the type of gates and their adjacencies, can be decided by random-access logarithmic-time deterministic Turing machines (Mix Barrington et al., 1990). For logarithmic-depth circuit families with bounded fan-in gates this corresponds to the fact that their *extended connection languages*, a language that extends the direct connection language to include tuples describing paths between gates in the circuits, can be decided by random-access logarithmic-time alternating Turing machines (Ruzzo, 1981; Buss, 1987; Mix Barrington et al., 1990).

The equality $\mathrm{PAC}^0 = \mathrm{TC}^0$ (and $\mathrm{GapAC}^0 = \mathrm{FTC}^0$) was first shown in the P-uniform setting (Agrawal et al., 2000; Ambainis et al., 1998), and later refined to also hold in the DLOGTIME-uniformity setting (Hesse et al., 2002). The classes $\#\mathrm{NC}^1$, GapNC^1, and PNC^1 where introduced by Caussinus et al. (1998) who showed $\mathrm{PNC}^1 \subseteq \mathrm{L}$ and $\mathrm{FNC}^1 \subseteq \#\mathrm{NC}^1$. The classes of functions can be arranged into the following chain of inclusions:

$$\mathrm{FAC}^0 \subsetneq \#\mathrm{AC}^0 \subseteq \mathrm{GapAC}^0 = \mathrm{FTC}^0 \subseteq \mathrm{FNC}^1 \subseteq \#\mathrm{NC}^1 \subseteq \mathrm{GapNC}^1 \subseteq \mathrm{FL} \ ,$$

where $\mathrm{GapNC}^1 \subseteq \mathrm{FL}$ follows from the work of Chiu et al. (2001), see also (Hesse et al., 2002). Note that, in order to state inclusion relations between classes of functions that compute numbers and classes of functions that compute strings, binary string representations of numbers are considered. Each of the above function classes is closed under composition. The language classes that are defined via Boolean and counting arithmetic circuits are related as follows:

$$\mathrm{AC}^0 \subsetneq \mathrm{PAC}^0 = \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{PNC}^1 \subseteq \mathrm{L} \ .$$

For a general introduction to the field of uniform circuit complexity, we refer to the book of Vollmer (1999) and the survey article of Allender (2004).

Reducibility Notions and Descriptive Complexity. We will use reducibility notions based on DLOGTIME-uniform FAC^0- and FTC^0-circuit families to compare the complexity of problems and state intermediate algorithmic steps. In terms of stating hardness of a problem for a complexity class, we use DLOGTIME-uniform AC^0-Turing-reductions for TC^0, and their many-one version for logarithmic-depth circuits and L. Intermediate algorithmic steps are stated in terms of functions from DLOGTIME-uniform FAC^0 and FTC^0 in the present and the next chapter.

Mix Barrington et al. (1990) proved that the DLOGTIME-uniform version of AC^0 equals the class of problems that can be defined by first-order formulæ with build-in ordering and bit predicates (DLOGTIME-uniform $\mathrm{AC}^0 = \mathrm{FO}$), and an analog characterization holds for TC^0 by additionally using MAJORITY-quantifiers that decide whether the majority of assignments of elements to a variable make a formula satisfy a given input structure (DLOGTIME-uniform $\mathrm{TC}^0 = \mathrm{FOM}$). The same holds for circuits that compute functions: The functions in DLOGTIME-uniform FAC^0 are exactly the FO-*computable functions*, used, for example, by Immerman (1999); the functions in DLOGTIME-uniform FTC^0 are exactly the FOM-*computable functions.* They are used, for example, by Lohrey (2001) and Gottlob et al. (2005). While the theorems of this thesis are stated in terms of DLOGTIME-uniform circuit classes, we use this *descriptive complexity* framework as a convenient tool for stating intermediate algorithmic results; that means, we argue that some function or the circuit it computes, is FO-computable, instead of talking about DLOGTIME-computations.

It is important to not mix up first-order formulæ as used in the preceding chapters and the concept of FO-computations: While the former is used to define properties of relational structures (and can only access the relations that are part of the structure), the later is used as a model of computation that works on the string input of a computational problem (and has access to build-in predicates whose power corresponds to the DLOGTIME preprocessing done to verify circuits). The input string might, as in the case of this thesis, contain, or exactly be, the encoding of some relational structure.

Encoding Relational Structures as Strings. In order to consider relational structures as inputs and outputs of Turing machines and circuits, we encode them as strings. We do this in the usual way as, for example, described by Immerman (1999): Without loss of generality, we only consider finite structures $\mathcal{A}$ whose universes are sets $A = \{1, \dots, n\}$ for some $n \in \mathbb{N}$. Let $\mathcal{A} = \{A, R_1^{\mathcal{A}}, \dots, R_m^{\mathcal{A}}\}$ be a structure over $\{R_1^{r_1}, \dots, R_m^{r_m}\}$. The *string encoding* of $\mathcal{A}$ is the string $\mathrm{str}(\mathcal{A}) = \mathrm{str}(R_1^{\mathcal{A}}) \dots \mathrm{str}(R_m^{\mathcal{A}})$ over the alphabet $\{0, 1\}$ where each relation $R_i^{\mathcal{A}}$ is represented by a bitstring $\mathrm{str}(R_i^{\mathcal{A}})$ of length $|A|^{r_i}$ whose jth bit is 1 if, and only if, the jth r_i-tuple (in lexicographic order) of elements of the universe is an element of the relation. Extracting the structure $\mathcal{A}$ from $\mathrm{str}(\mathcal{A})$ is FO-computable (Immerman, 1999).

For some problems, the input and output structures are given along with numbers that encode, for example, a solution size. Unless stated otherwise, we assume these numbers to be encoded in binary. For example, an input $(\mathrm{str}(\mathcal{G}), s)$ to DOMINATING-SET consists of the string encoding of a graph $\mathcal{G}$ and a solution size s given in binary.

PROOF OF THEOREM 1.6. Since extracting a relational structure from its string encoding is FO-computable, definability of a property of structures in terms of first-order formulæ translates to the FO-computability of deciding whether the structure encoded in an input string has this property. Thus, Theorem 1.6 follows from Theorem 1.9. □

4.2. Getting Familiar with Tree Decompositions

The first step toward our goal of proving Theorem 1.7 is to turn structures of bounded tree depth and MSO-formulæ on them into equivalent depth-bounded s-trees and MSO-formulae on s-trees. The construction of s-trees and equivalent formulæ is done in two steps: First, tree decompositions of bounded width whose underlying trees have bounded depth are constructed using FO-computations. Then tree decompositions are turned into s-trees and formulæ are adjusted accordingly. We start with the definition of tree decompositions.

Definition of Tree Decompositions. Robertson and Seymour (1986) define the concept of tree width of graphs through tree decompositions; we use the following generalized definition of tree decompositions that is used, for example, by Flum and Grohe (2006). It applies to relational structures of any signature and, thus, also graphs.

DEFINITION 4.1 (Tree Decomposition). A *tree decomposition* (T, B) of a structure $\mathcal{A}$ over some vocabulary τ is a tree T together with a labeling function $B\colon V(T) \to \mathcal{P}(A)$, where $\mathcal{P}(A)$ is the power set of A, that satisfies the following two properties:

– *Connectedness condition:* For all $a \in A$, the induced subtree $T\bigl[\{n \in V(T) \mid a \in B(n)\}\bigr]$ is nonempty and connected.

– *Cover condition:* For every symbol $R^r \in \tau$ and every tuple $(a_1, \dots, a_r) \in R^{\mathcal{A}}$, there is an $n \in V(T)$ with $\{a_1, \dots, a_r\} \subseteq B(n)$.

The sets $B(n)$ are called the *bags* of the tree decomposition. The *width* of the tree decomposition is $\max_{n \in V(T)} |B(n)| - 1$.

Since a tuple of r elements of a structure gives rise to a clique of size r in its Gaifman graph and in a tree decomposition every clique is completely contained in some bag, the following holds:

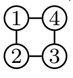
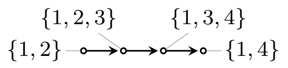
FACT 4.2. *Let $\mathcal{A}$ be a structure. Every tree decomposition of the Gaifman graph $G(\mathcal{A})$ is also a tree decomposition of $\mathcal{A}$, and vice versa.*

DEFINITION 4.3 (Tree Width). The *tree width* $\operatorname{tw}(\mathcal{A})$ of a structure $\mathcal{A}$ is the minimum width of a tree decomposition for it.

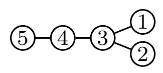
A class C of structures over some vocabulary τ has *bounded tree width* (is *tree-width-bounded*) if there exists a constant $w \in \mathbb{N}$, such that $\operatorname{td}(\mathcal{A}) \leq w$ for every $\mathcal{A} \in C$.

FACT 4.4. *For every structure $\mathcal{A}$, we have* $\operatorname{tw}(\mathcal{A}) = \operatorname{tw}(G(\mathcal{A}))$.

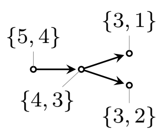
EXAMPLE 4.5 (Tree Decompositions and Tree Width).

While tree depth can be seen as measuring the distance of a graph to a star graph and grows for graphs that have long paths, tree width measures the distance of a graph to trees and grows for graphs with cycles that are intertwined. All trees have tree width 1, like the undirected tree $\mathcal{T}$ on the right for which a tree decomposition (T, B) of width 1, which witnesses this fact, is given. An undirected cycle like  has tree decompositions like  of width 2. Note how vertex 1 is carried along the tree decomposition to satisfy the cover condition for its edges to 2 and 4, and, at the same time, satisfy the connectedness condition for it.

Tree $\mathcal{T}$



(T, B)



A graph class of *unbounded tree width* is the class of all cliques. Every n-vertex clique has tree width $n - 1$ (thus, tw( ) = 4). Another example are $n \times n$ grids, which have tree width n (thus, tw( ) = 3). In fact, Robertson and Seymour (1984) showed that a class of graphs has unbounded tree width exactly if its graphs contain grids of any size as minors.

The graph problem TREE-WIDTH $= \{(\operatorname{str}(\mathcal{G}), w) \mid \operatorname{tw}(\mathcal{G}) \leq w\}$, which is the same problem as deciding whether embeddings into k-trees exist, is NP-complete in general (Arnborg et al., 1987). In contrast, for any constant w, the problem TREE-WIDTH-w $= \{\operatorname{str}(\mathcal{G}) \mid \operatorname{tw}(\mathcal{G}) \leq w\}$ is shown to be L-complete in Section 6.3. While in Chapter 6, we construct tree decompositions of any constant width in logspace, for the results of the present chapter it is enough to construct tree decompositions of bounded width whose underlying trees have bounded depth. It turns out that this task has a much lower complexity; it is FO-computable (Lemma 4.6). Just deciding whether an input structure has some constant tree depth $d \in \mathbb{N}$, that means solving $\textsc{tree-depth-}d = \{\mathrm{str}(\mathcal{G}) \mid \mathrm{td}(G) \leq d\}$, can be done in DLOGTIME-uniform AC^0 by extracting $\mathcal{G}$ from $\mathrm{str}(\mathcal{G})$ and applying Lemma 3.18. In contrast, the problem $\textsc{tree-depth} = \{(\mathrm{str}(\mathcal{G}), d) \mid \mathrm{td}(\mathcal{G}) \leq d\}$, where d is part of the input, is NP-complete (Bodlaender et al., 1995; Nešetřil and Ossona de Mendez, 2006).

Computing Width- and Depth-Bounded Tree Decompositions. In the present section, we show how to construct tree decompositions of bounded width whose underlying trees have bounded depth for structures whose tree depth is bounded by some constant. We already worked with structures of bounded tree depth in Chapter 3. What prevented us from defining trees based on graphs in Chapter 3 was the fact that a first-order formula, which has only access to the graph structure itself, cannot distinguish between vertices that have the same properties with respect to first-order formulæ. Thus, they cannot single out a root element that serves as a starting point to build a tree. FO-computations are different: In addition to the expressive power of first-order logic on the encoded structure, they are able to compute properties based on ordering and arithmetic predicates defined on the input string. For example, if there are many candidates for the root element of a tree, then an FO-computation just chooses the one with the smallest index. In the following, whenever we talk about the fact that computing devices (like FO-computations, circuits, or Turing machines) *choose* an element, tuple, or relation, we mean that a uniquely determined one among all candidates is chosen, like the one with the smallest index.

LEMMA 4.6. *Let τ be a vocabulary and $d \in \mathbb{N}$. There is a* FO*-computable function that, on input of the encoding* $\mathrm{str}(\mathcal{A})$ *of a τ-structure $\mathcal{A}$ of tree depth at most d, outputs a tree decomposition (T, B) for $\mathcal{A}$ of width at most $d-1$ where T has depth at most $d+1$.*

PROOF. On input of $\mathcal{A}$, we first construct the first-order definable Gaifman graph $\mathcal{G} = G(\mathcal{A})$. By Fact 3.13, we know that the tree depth of $\mathcal{G}$ is bounded by $d \in \mathbb{N}$. We proceed to build a tree decomposition (T, B) of width at most $d-1$ and depth at most $d+1$ for $\mathcal{G}$; by Fact 4.2, this is also a tree decomposition for the input structure $\mathcal{A}$. We first prove the existence of the decomposition and, then, discuss how to implement its construction using FO-computations.

Existence of the tree decomposition: We start to prove the existence of decompositions for graphs that are described by descriptors and, then, generalize to graphs of any kind. First, we recall the definition of descriptors from the previous chapter: A width-w descriptor D in $\mathcal{G}$ consists of w bag vertices b_1 to b_w, and a component selector vertex s; D selects the component $\mathcal{C}_D$ of $\mathcal{G}[V(\mathcal{G}) \setminus \{b_1, \dots, b_w\}]$ that contains s and describes $\mathcal{G}_D = \mathcal{G}[C_D \cup \{b_1, \dots, b_w\}]$.

CLAIM. *Let $d \in \mathbb{N}$ and $w \in \mathbb{N}_0$. For every graph $\mathcal{G}$ and width-w descriptor $D = (b_1, \dots, b_w, s)$ in $\mathcal{G}$ with $\mathrm{td}(\mathcal{C}_D) \leq d$, there exists a tree decomposition (T, B) for $\mathcal{G}_D$ of width at most $w + d - 1$, where T has depth at most d, and $\{b_1, \dots, b_w\} \subseteq B(r)$ holds for the root r of T.*

We prove the claim by induction over d for all w. For $d = 1$, $\mathcal{C}_D$ contains only a single element, which is s. We build a tree decomposition with a single node r whose bag is $B(r) = \{b_1, \dots, b_w, s\}$; the size of r's bag is $w + 1$ and T has depth 1. For $d > 1$, we know that the connected graph $\mathcal{C}_D$ contains a vertex b_{w+1} whose removal produces a graph of tree depth at most $d - 1$. Let $(\mathcal{C}_i)_{i\in I}$ be the family of components of $\mathcal{C}_D$ without b_{w+1}. Each component $\mathcal{C}_i$ can be selected by a descriptor $D_i = (b_1, \dots, b_w, b_{w+1}, s_i)$. Since the tree depth of each $\mathcal{C}_i$ is at most $d - 1$, we know from the induction hypothesis that there exists a tree decomposition (T_i, B_i) for $\mathcal{G}_{D_i}$ of width at most $w + 1 + d - 2 = w + d - 1$ where T_i has depth at most $d-1$, and whose root bag $B_i(r_i)$ contains the vertices b_1 to b_{w+1}. To construct a tree decomposition for $\mathcal{G}_D$, we add a new root node r with bag $\{b_1, \dots, b_{w+1}\}$ and connect it to the nodes $(r_i)_{i\in I}$. The width bound remains the same and the depth bound increases by one. The resulting tree with the bag labeling is a tree decomposition for $\mathcal{G}_D$: The connectedness condition is satisfied for all vertices that only live inside a single $\mathcal{G}_{D_i}$ by induction. The only vertices that are shared by multiple $\mathcal{G}_{D_i}$ are the vertices b_1 to b_{w+1}. The connectedness condition is satisfied for them because they are part of the bags $B_i(r_i)$, and part of r's bag. Each edge of $\mathcal{G}_D$ is part of some $\mathcal{G}_{D_i}$ and, thus, the cover condition is satisfied for it by the induction hypothesis.

To prove the existence of tree decompositions for a graph $\mathcal{G}$ that is not described, we build a tree decomposition (T_i, B_i) of width $d - 1$ and depth d for each component $\mathcal{G}_i$ of $\mathcal{G}$ using width-0 descriptors $D_i = (s_i)$ with $\mathcal{G}_{D_i} = \mathcal{G}_i$. Then we put the decompositions together using a new root node r with bag $B(r) = \emptyset$ that is connected to the root nodes of the decompositions (T_i, B_i). With this construction, the width bound remains the same and the depth increases by one. The combined decomposition satisfies connectedness and cover conditions since the $\mathcal{G}_i$ are connected components.

FO-*computability:* Considering the recursion from the proof of the claim, the task of choosing a unique vertex b_{w+1} is FO-computable with the help of the first-order definition of tree depth for selected components from Lemma 3.17, and Fact 3.15; the length of paths in tree-depth-bounded graphs is bounded. The task of putting depth-bounded tree decompositions together and appending a new root node is also FO-computable. Finally, since we only need a constant number of recursions, we connect a constant number of FO-computations for (1) dividing the input graph into components, and (2) combining the output of the recursions. This computes a tree decomposition of bounded width whose underlying tree has bounded depth for described graphs. To construct decompositions for an input graph that is not described, we first partition it into its connected components, which are FO-computable since the longest path length in the graph is bounded. Then we construct tree decompositions for the components and, finally, put the decompositions together. □

From Structures With Tree Decompositions to Trees. Using the previous lemma, we can construct tree decompositions (T, B) of bounded width where T has bounded depth for logical structures $\mathcal{A}$ of bounded tree depth using FO-computations. Our aim is to work with these tree decompositions rather than the original structures. However, the formula φ for which we wish to compute a histogram refers to $\mathcal{A}$, not to (T, B). Thus, our objective is to transform the formula φ into an equivalent formula ψ that refers to a tree structure $\mathcal{T}$ based on this tree decomposition rather than the structure $\mathcal{A}$. Proofs of transforming structures into trees with the help of tree decompositions are used a lot in the literature; see, for example, (Arnborg et al., 1991). The proof presented in this thesis is adjusted to (1) ensure that the depth of the tree only grows by a constant additive factor, (2) the solution histogram remains the same, and (3) the transformation is FO-computable.

Note that the lemma does not restrict the depth of the tree that underlies the input decomposition. In fact, while in the present chapter we will only consider input decompositions whose underlying trees have bounded depth, in Chapter 5 we will also translate tree decompositions of any depth into equivalent s-trees.

LEMMA 4.7. *Let $\varphi(X_1, \dots, X_k)$ be an* MSO*-formula over some vocabulary τ and $w \in \mathbb{N}$ a width bound. There is an $s \in \mathbb{N}_0$, and an* MSO*-formula $\psi(X_1, \dots, X_k)$ over $\tau_{s\text{-tree}}$, and a* FO*-computable function that, on input of any τ-structure $\mathcal{A}$ with universe A and a width-w tree decomposition (T, B) for $\mathcal{A}$, produces an s-tree structure $\mathcal{T}$, such that*

(1) the depth of $\mathcal{T}$ equals the depth of T plus 1, and

(2) we have $\text{histogram}(\mathcal{A}, \varphi)[s] = \text{histogram}(\mathcal{T}, \psi)[s]$ *for all indices $s \in \{0, \dots, |A|\}^k$ and all other entries in the array* $\text{histogram}(\mathcal{T}, \psi)$ *are 0.*

PROOF. The node set of $\mathcal{T}$ is the union of two disjoint sets V_B and V_E of nodes, which we call the *bag nodes* and the *element nodes*, respectively. The set V_B is exactly $V(T)$. The set V_E is the disjoint union of the sets $\{e_1^n, \dots, e_{r_n}^n\}$ for $n \in V(T)$ with attached bag $B(n) = \{e_1, \dots, e_{r_n}\}$, where some ordering is chosen for each bag. For an element node $x = e_i^n$, we write $n(x)$ for the node $n \in V(T)$, we write $i(x)$ for the index i, and we write $e(x)$ for the element $e_i \in A$. The edges of $\mathcal{T}$ are as follows: All edges of T are also present in $\mathcal{T}$. Additionally, for each $x \in V_E$ there is an edge from $n(x)$ to x.

The unary predicates of $\mathcal{T}$ fall into the following four groups:

(1) *Node type predicates:* We define predicates $P_B^{\mathcal{T}} := V_B$ and $P_E^{\mathcal{T}} := V_E$.

(2) *Element ordering predicates:* We use $w + 1$ predicates $P_1^{\mathcal{T}}, \dots, P_{w+1}^{\mathcal{T}}$ to record the chosen total ordering for the element nodes of each bag: That means, for each bag node n with attached element nodes $\{x_1, \dots, x_r\}$ we set $x_1 \in P_{i(x_1)}^{\mathcal{T}}$ to $x_r \in P_{i(x_r)}^{\mathcal{T}}$.

(3) *Structure predicates:* These predicates are used to represent the relations from the structure $\mathcal{A}$. To represent a relation $R^{\mathcal{A}}$ of arity r of $\mathcal{A}$, we introduce new predicates $P_{i_1,\dots,i_r}^{\mathcal{T}}$ for all $i_1, \dots, i_r \in \{1, \dots, w+1\}$. They locally encode the tuples of $R^{\mathcal{A}}$ at the bags with $(i_1, \dots, i_r)$ being the local indices of the elements of a tuple of $R^{\mathcal{T}}$: For every tuple $(x_1, \dots, x_r) \in V_E^r$ with $n(x_1) = \dots = n(x_r)$ and $(e(x_1), \dots, e(x_r)) \in R^{\mathcal{A}}$, we let $x_j \in P_{i(x_1),\dots,i(x_r)}^{\mathcal{T}}$ for each $j \in \{1, \dots, r\}$. Since a tree decomposition puts the elements of a tuple completely into at least one bag, for all tuples $(e_1, \dots, e_r) \in R^{\mathcal{A}}$ there are element nodes $x_1, \dots, x_r$ with $n(x_1) = \dots = n(x_r)$ and $x_1 \in P_{i(x_1),\dots,i(x_r)}^{\mathcal{T}} \wedge x_1 \in P_{i(x_1)}^{\mathcal{T}}$, $\dots$, $x_r \in P_{i(x_1),\dots,i(x_r)}^{\mathcal{T}} \wedge x_r \in P_{i(x_r)}^{\mathcal{T}}$.

(4) *Equivalence predicates:* These predicates are used to relate element nodes that stand for the same element in the structure. We introduce $w + 1$ predicates $N_1^{\mathcal{T}}, \dots, N_{w+1}^{\mathcal{T}}$ and put an element node x into $N_j^{\mathcal{T}}$ if $e(x) = e(y)$ for some y with $i(y) = j$ and $n(y)$ is the parent node of $n(x)$ in $\mathcal{T}$. If an element

node is in none of the predicates $N_i^{\mathcal{T}}$, we insert it into the predicate $P_{\text{repr}}^{\mathcal{T}}$, the set of *representative element nodes*. By this construction, there is a one-to-one correspondence between $P_{\text{repr}}^{\mathcal{T}}$ and the universe of $\mathcal{A}$. For each element $e \in A$, the unique element node x with $e = e(x)$ that is nearest to the root of $\mathcal{T}$ is part of $P_{\text{repr}}^{\mathcal{T}}$. All steps of the reduction are FO-computable.

The formula ψ is build on top of the following subformulæ:

(a) The MSO-formula $\psi_{\text{equ}}(x, y)$ over $\tau_{s\text{-tree}}$. It is true if x and y are element nodes with $e(x) = e(y)$. This formula quantifies over element nodes that are attached to the bag nodes on the path between x and y and uses the element ordering predicates and the equivalence predicates to make sure that all chosen element nodes stand for the same element of the universe of $\mathcal{A}$.

(b) For every $R^r \in \tau$, the MSO-formula $\psi_R(x_1, \dots, x_r)$ over $\tau_{s\text{-tree}}$, where the x_i are *first-order* variables. It is true for representative element nodes x_i if, and only if, $(e(x_1), \dots, e(x_r)) \in R^{\mathcal{A}}$: The formula tests whether there are element nodes $y_1, \dots, y_r$ such that $\psi_{\text{equ}}(x_1, y_1) \wedge \dots \wedge \psi_{\text{equ}}(x_r, y_r)$, the y_i are children of the same bag node, and there is an index tuple $(i_1, \dots, i_r)$ with $P_{i_1,\dots,i_r}(y_1) \wedge y_1 \in P_{i_1} \wedge \dots \wedge P_{i_1,\dots,i_r}(y_r) \wedge y_r \in P_{i_r}$.

To build ψ, we first extend the formula φ such that only elements and subsets of $P_{\text{repr}}^{\mathcal{T}}$ are permissible for the free and bounded variables. Second, substitute $x = y$ with $\psi_{\text{equ}}(x, y)$. Third, substitute every $R(x_1, \dots, x_r)$ with the formula $\psi_R(x_1, \dots, x_r)$. □

Lemma 4.6 and Lemma 4.7 together provide a transformation from evaluating MSO-formulæ on structures of bounded tree depth to s-tree structures of bounded depth.

4.3. From Automata Evaluation to Arithmetic Circuit Evaluation

Theorem 2.19 on page 23 establishes a link between formulæ and multiset tree automata that is solution-preserving in the sense that there is a one-to-one correspondence between satisfying assignments to the free variables of the formulæ and labelings of the trees that make the automaton accept. Using this theorem, we can translate the task of computing a solution histogram for a formula on s-trees to the task of evaluating automata. In the present section we pick up the later problem and solve it as follows: First, we replace the evaluation of multiset tree automata by the evaluation of convolution circuits, see Lemma 4.10, such that the outputs of the circuits are the sought solution histograms. Then we reduce the evaluation of convolution circuits to the evaluation of arithmetic circuits.

From Formula Histograms to Multicoloring Histograms. In order to talk more easily about the number of labelings that makes an automaton accept a tree, we define the notion of *multicolorings*. An $[r]^k$*-array* is an k-dimensional array of integers where all indices $s = (s_1, \dots, s_k)$ are elements of the index set $[r]^k = \{0, \dots, r-1\}^k$. We call r the *range*. Given a set S, a *multicoloring* of S is a tuple $(S_1, \dots, S_k)$ of subsets $S_j \subseteq S$ for $j \in \{1, \dots, k\}$. Given a set X of multicolorings of S, let histogram(X) denote the $[|S|+1]^k$-array whose entry at index $s = (s_1, \dots, s_k)$ is the number of multicolorings $(S_1, \dots, S_k) \in X$ with $|S_1| = s_1$, …, $|S_k| = s_k$. For instance, for $S = \{1, 2\}$ and $X = \{(\{1\}, \{1\}), (\{2\}, \{2\}), (\{2\}, \emptyset)\}$, we have $\text{histogram}(X) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$.

The connection between multicolorings and tree automata is as follows: Given a multiset tree automaton $A = (\{0,1\}^{s+k}, Q, Q_a, \Delta)$, $P \subseteq Q$, and an s-tree structure $\mathcal{T}$ with universe V, let us write $S_A(\mathcal{T}, P)$ for the set of tuples $(S_1, \dots, S_k)$ with $S_i \subseteq V$ for which A reaches a state $q \in P$ at the root of $T(\mathcal{T}, S_1, \dots, S_k)$. Clearly, $S_A(\mathcal{T}, P)$ is a set of multicolorings of V. In particular, for the automaton A constructed in Theorem 2.19 for a formula φ we have $\text{histogram}(\mathcal{T}, \varphi) = \text{histogram}(S_A(\mathcal{T}, Q_a))$. This means that all we have to do in the following is to devise a way of computing $\text{histogram}(S_A(\mathcal{T}, Q_a))$.

Before we proceed, it will be useful to define some simple operations on sets of multicolorings and see how these operations change their histograms. First, for two disjoint sets of multicolorings X_1 and X_2 of the same set S, we have $\text{histogram}(X_1 \cup X_2) = \text{histogram}(X_1) + \text{histogram}(X_2)$ where the addition of arrays is just the component-wise addition. Second, given two disjoint sets S and U and sets of multicolorings X and Y of S and U, respectively, let us write $X \otimes Y$ for the set of multicolorings $\{(S_1 \cup U_1, \dots, S_k \cup U_k) \mid (S_1, \dots, S_k) \in X, (U_1, \dots, U_k) \in Y\}$. To understand its effect, consider the case where $k = 1$. Then $X \otimes Y = \{S \cup U \mid S \in X, U \in Y\}$ and consider, say, $\text{histogram}(X \otimes Y)[3]$. This is number of ways we can choose sets $S \in X$ and $U \in Y$ with $|S \cup U| = 3$. It is not hard to see that this is exactly $\text{histogram}(X)[0] \cdot \text{histogram}(Y)[3] + \text{histogram}(X)[1] \cdot \text{histogram}(Y)[2] + \text{histogram}(X)[2] \cdot \text{histogram}(Y)[1] + \text{histogram}(X)[3] \cdot \text{histogram}(Y)[0]$. This sum is also known as the convolution of the two histogram arrays at position 3. In general, given two arrays B and C with the same dimension k and ranges r and s, respectively, their *convolution* is the $[r+s-1]^k$-array $D = B * C$ with

$$D[\ell] = \sum_{i \in [r]^k, j \in [s]^k \text{ with } \ell = i+j} B[i]C[j].$$

With these definitions, $\text{histogram}(X \otimes Y) = \text{histogram}(X) * \text{histogram}(Y)$.

Turning Automata Computations Into Convolution Circuits. Our ultimate goal is to compute (number encoded) solution histograms using arithmetic circuits. We postpone the problem of computing number encodings for the moment, leaving us with the computation of histograms. To this end, we now introduce convolution circuits, which instead of passing around Boolean values (like AC^0 circuits) or numbers (like GapAC^0 circuits) pass around whole number arrays (that is, histograms).

Definition 4.8 (Convolution Circuit). A *convolution circuit* is a circuit C where each inner gate is labeled with $+$, $-$, or $*$. The addition and convolution gates have unbounded fan-in, the subtraction gates have fan-in 2 and their inputs are ordered. Constant gates can be labeled with arbitrary arrays. A convolution circuit without subtraction gates is *positive*.

In slight abuse of notation, when describing the structure of circuits, we will sometimes just write down formulæ involving addition and convolution operators. For instance, $B * C + D$ denotes the circuit starting with an addition gate at the top, one convolution gate as a child, and the three leaves B, C, and D. When a gate has many input gates $C_1, \dots, C_n$, we use the notation $\sum_i C_i$ for addition gates and $\prod_i C_i$ for convolution gates.

Definition 4.9 (Computation of a Convolution Circuit). The input for a convolution circuit C with n inputs is a sequence $(B_1, \dots, B_n)$ of arrays.

The *value* $\mathrm{val}(g, C[B_1, \dots, B_n])$ of a gate g is the component-wise addition, component-wise difference, or convolution of the arrays that are the values of the gate's inputs. For the ith input gate, $\mathrm{val}(g, C[B_1, \dots, B_n])$ is B_i; while for constant gates its value is the constant attached to it. The array produced at the output gate will be denoted $\mathrm{val}(C[B_1, \dots, B_n])$ or, if there are no input gates, just $\mathrm{val}(C)$.

We are now ready to state a lemma that shows how the histograms of multiset tree automata on labeled trees can be computed using convolution circuits. In its proof addition gates are used to unite histograms that arise from disjoint sets of solutions: For instance, suppose that for some deterministic multiset tree automaton A we have found a way to compute $h_q = \mathrm{histogram}(S_A(\mathcal{T}, \{q\}))$ for all states $q \in Q_a$. Then the sum over all of these histograms will be the histogram of all multicolorings that make A accept. Convolution gates are used to combine solution histograms for different child trees: Suppose the root of a tree has exactly two children c_1 and c_2 and suppose the only way to reach a state q at the root is to reach q_1 at c_1 and q_2 at c_2. Then the set of multicolorings $S_A(\mathcal{T}, \{q\})$ is exactly $S_A(\mathcal{T}_1, \{q_1\}) \otimes S_A(\mathcal{T}_2, \{q_2\})$, where $\mathcal{T}_1$ and $\mathcal{T}_2$ are the subtrees rooted at c_1 and c_2, because every multicoloring of $\mathcal{T}_1$ that makes A reach $\{q_1\}$ can be combined with every multicoloring of $\mathcal{T}_2$ that makes A reach $\{q_2\}$. But then, as we saw above, the desired solution histogram $\mathrm{histogram}(S_A(\mathcal{T}, \{q\}))$ is given by the convolution of the subtrees histograms. If there are different ways to reach q, we sum them up. Subtraction gates come into play if the degree of the tree is too high to enumerate all possible ways in which the automaton can reach a state.

LEMMA 4.10. *Let $A = (\{0,1\}^{s+k}, Q, Q_a, \delta)$ be a deterministic multiset tree automaton with multiplicity bound $m \in \mathbb{N}$. Then there is an* FO*-computable function that maps every s-tree structure $\mathcal{T} = (V, P_1^{\mathcal{T}}, \dots, P_s^{\mathcal{T}})$ to a convolution circuit C such that*

(1) $\mathrm{val}(C) = \mathrm{histogram}(S_A(\mathcal{T}, Q_a))$,
(2) the depth of C is bounded by a function that depends on A and linearly on the depth of $\mathcal{T}$, and
(3) the fan-in of C is bounded by a function that depends on A and linearly on the degree of $\mathcal{T}$.

Furthermore, if the degree of $\mathcal{T}$ is less than m, then C is positive.

PROOF. Let $A = (\{0,1\}^{s+k}, Q, Q_a, \delta)$ be a deterministic multiset automaton with multiplicity bound $m \in \mathbb{N}$. Let $\mathcal{T} = (V, P_1^{\mathcal{T}}, \dots, P_s^{\mathcal{T}})$ be an s-tree structure. We will first describe a transformation that turns $\mathcal{T}$ and A into a convolution circuit C by locally transforming the root of every subtree $\mathcal{T}'$ into a subcircuit whose inputs are the outputs of the subcircuits for the child trees of $\mathcal{T}'$. Then we will prove that C satisfies the claimed properties.

Before we go into the details of the construction, we introduce some new terminology regarding multisets of states (multisets and related set-theoretic terms are defined on page 16). Given a multiset $M \in \mathcal{P}_\omega(Q)$ of states, let us say that a state $q \in Q$ is *rare in M*, if $\#_M(q) < m$ and let us say that it is *plentiful in M* if $\#_M(q) \geq m$. Let us write $\mathrm{rare}(M)$ and $\mathrm{plenty}(M)$ for the set of rare and plentiful states, respectively.

Construction and its complexity: For every subtree $\mathcal{T}' = (V', P_1^{\mathcal{T}'}, \dots, P_s^{\mathcal{T}'})$ of $\mathcal{T}$ we build two groups of subcircuits, called the *transition circuits* C_{trans}, which depend on the transition function, and the *multiset circuits* C_{multi}, which combine histograms for specific multisets of states at the child trees:

(1) For the transition circuits, let $z \in \{0,1\}^s$ be the label of the root of $T(\mathcal{T}')$. Then for every state set $P \subseteq Q$ let $C_{\text{trans}}(\mathcal{T}', P)$ have the form

$$\sum_{\substack{x \in \{0,1\}^k,\ M \in \mathcal{P}_m(Q) \text{ s.t.} \\ |M| \leq n \text{ and } \delta(zx, M) \in P}} C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M) * \chi(x),$$

where $\chi(x)$ is the k-dimensional array with a 1-entry at position x and 0-entries at all other positions. The $C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M)$ are the multiset circuits defined next. Each C_{trans} has depth 2.

(2) Let $\mathcal{T}_1, \dots, \mathcal{T}_n$ be the child subtrees of $\mathcal{T}'$ and consider any $M \in \mathcal{P}_m(Q)$ with $|M| \leq n$. Then the circuit $C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M)$ is of the form $C_{\text{uncap}} - C_{\text{correct}}$, where C_{uncap} is the *uncapped circuit* and C_{correct} is the *correction circuit.*

The uncapped circuit C_{uncap} starts with an addition gate that sums up a large number of subcircuits. There is one subcircuit for each function

$$f \colon \{1, \dots, n\} \to \{\{q\} \mid q \in \text{rare}(M)\} \cup \{\text{plenty}(M)\}\ ,$$

with

$$M|_{\text{rare}(M)} = \bigcup_{\substack{i \in \{1,\dots,n\} \text{ s.t.} \\ f(i) \neq \text{plenty}(M)}} f(i)\ .$$

In other words, for each rare state $q \in M$, the state must be present in M exactly as often as f maps some i to $\{q\}$; in contrast, f can map an arbitrary number of i to a plentiful state. The subcircuit has the form $\prod_{i=1}^{n} C_{\text{trans}}(\mathcal{T}_i, f(i))$.

The correction circuit also starts with an addition gate. This gate is directly connected to all circuits $C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, N)$ where $N \subsetneq M$ and $M|_{\text{rare}(M)} = N|_{\text{rare}(M)}$. (In other words, N is obtained from M by deleting some elements from states that used to be plentiful in M and by leaving rare states untouched.) Note that, since N is a proper subset of M, this definition is not cyclic.

Subtraction gates: If the number n of children is less than the multiplicity bound of m, we have $|M| \leq n < m$. In this case, M contains only rare states and there is no $N \subsetneq M$ with $M|_{\text{rare}(M)} = N|_{\text{rare}(M)}$. This, in turn, means that the correction circuit is empty and can be left out. Hence, the circuit is positive.

The output of the whole circuit C is $C_{\text{trans}}(\mathcal{T}, Q_a)$. Since we transfer $\mathcal{T}$ into C by making only local changes that depend on the fixed automaton A, the construction is FO-computable.

Correctness: To show the first property of the lemma, we prove the following two claims:

Claim (Correctness of Transition Circuits). *For each $P \subseteq Q$ we have* $\text{val}(C_{\text{trans}}(\mathcal{T}', P)) = \text{histogram}(S_A(\mathcal{T}', P))$.

Claim (Correctness of Multiset Circuits). *For each $M \in \mathcal{P}_m(Q)$ with $|M| \leq n$ we have*

$$\text{val}(C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M)) = \sum_{\substack{q_1,\dots,q_n \in Q \text{ s.t.} \\ \{q_1,\dots,q_n\}|_m = M}} \prod_{i=1}^{n} \text{val}(C_{\text{trans}}(\mathcal{T}_i, \{q_i\}))\ .$$

The proof of these claims is based on two nested inductions. The outer induction is an induction over the structure of the tree $\mathcal{T}'$. Thus, in this outer induction we assume that both claims have already been proved for the child trees $\mathcal{T}_i$ of $\mathcal{T}'$ and we must show that they both hold for $\mathcal{T}'$. The second inner induction is over the size of the multisets M. Recall that the definition of the correction circuits C_{correct} refers to the circuits $C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, N)$ for proper subsets N of M. We show that the second claim holds for a given M under the assumption that it holds for all $N \subsetneq M$.

For the outer inductions, first observe that if the claims hold for the child trees $\mathcal{T}_1$ to $\mathcal{T}_n$ of a subtree $\mathcal{T}'$, then the first claim holds for this particular subtree: By the second claim and the outer induction hypothesis, we know that the array $\text{val}(C_{\text{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M))$ stores the number of multicolorings making A reach a particular capped multiset $M \in \mathcal{P}_m(Q)$ at the children of the root of $\mathcal{T}'$. Note that for each pair of different multisets the underlying sets of multicolorings are distinct. But, then, the circuit $C_{\text{trans}}(\mathcal{T}', P)$ sums up exactly over those histograms that contribute to reaching a state from P at the root. The convolution with $\chi(x)$ ensures that the histogram for the children is shifted to accommodate for the contribution of the root's label x to the sizes of the multicolorings.

Next, to prove the second claim, in addition to the outer induction hypothesis we have the inner induction hypothesis that the second claim holds for all proper subsets $N \subsetneq M$. For the empty M there is nothing to prove. We give names to sequences $(q_1, \dots, q_n)$ of states $q_i \in Q$: Let us say that the sequence is *perfect* if $\{q_1, \dots, q_n\}|_m = M$. Let us call it *good* if $\{q_1, \dots, q_n\}|_{\text{rare}(M)} = M|_{\text{rare}(M)}$. Clearly, every perfect sequence is good, but not necessarily the other way round, namely when a plentiful state of M is present less than m times in the sequence. Let us call a sequence *superfluous* if it is good, but not perfect. Note that the second claim states that we can express the value of the multiset circuits as a sum over all perfect sequences. In the following, we show that the uncapped circuit C_{uncap} computes the sum over all good sequences while the correction circuit computes the sum over all superfluous sequences. Then, since the multiset circuit is just $C_{\text{uncap}} - C_{\text{trans}}$, we get the second claim.

The uncapped circuit C_{uncap} computes, by definition, the first line of the following equation and we claim that it can be rewritten as the second line:

$$\begin{aligned}
&\sum_{\substack{f:\{1,\dots,n\}\to\{\{q\}\mid q\in\text{rare}(M)\}\cup\{\text{plenty}(M)\} \text{ s.t.} \\ M|_{\text{rare}(M)}=\cup_{i\in\{1,\dots,n\} \text{ s.t. } f(i)\neq\text{plenty}(M)} f(i)}} \prod_{i=1}^{n} \text{val}(C_{\text{trans}}(\mathcal{T}_i, f(i))) \\
= &\sum_{\substack{(q_1,\dots,q_n)\in Q^n \text{ s.t.} \\ (q_1,\dots,q_n) \text{ is good}}} \prod_{i=1}^{n} \text{val}(C_{\text{trans}}(\mathcal{T}_i, q_i)) \ .
\end{aligned}$$

To prove this equality, first fix a function f. Let us say that a sequence $(q_1, \dots, q_n)$ of states is *good for f*, if $\{q_i\} = f(i)$ for all i with $f(i) \neq \text{plenty}(M)$ and $q_i \in \text{plenty}(M)$ for all i with $f(i) = \text{plenty}(M)$. Observe that for the latter kind of i, the set of sequences good for f will range over all possible combinations of states $q_i \in \text{plenty}(M)$ for these positions. By construction of the transition circuits, we have $C_{\text{trans}}(\mathcal{T}_i, f(i)) = \sum_{q \in f(i)} C_{\text{trans}}(\mathcal{T}_i, \{q\})$. This implies that we can rewrite $\prod_{i=1}^{n} \mathrm{val}(C_{\mathrm{trans}}(\mathcal{T}_i, f(i)))$ as follows:

$$\sum_{\substack{(q_1,\dots,q_n)\in Q^n \text{ s.t.}\\ (q_1,\dots,q_n) \text{ is good for } f}} \prod_{i=1}^{n} \mathrm{val}(C_{\mathrm{trans}}(\mathcal{T}_i, \{q_i\})) \ .$$

Since every sequence $(q_1, \dots, q_n)$ is good for exactly one f and since

$$\{q_1, \dots, q_n\}|_{\mathrm{rare}(M)} = \bigcup_{\substack{i\in\{1,\dots,n\} \text{ s.t.}\\ f(i)\neq \mathrm{plenty}(M)}} f(i)$$

holds for this f, we get the claimed equality.

Let us now analyse the correction circuit C_{correct}. By definition, it computes the following value:

$$\sum_{\substack{N\subsetneq M \text{ s.t.}\\ M|_{\mathrm{rare}(M)}=N|_{\mathrm{rare}(M)}}} \mathrm{val}(C_{\mathrm{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, N)) \ .$$

By applying the inner induction hypothesis to N for the second claim, we can rewrite this as

$$\begin{aligned}
&\sum_{\substack{N\subsetneq M \text{ s.t.}\\ M|_{\mathrm{rare}(M)}=N|_{\mathrm{rare}(M)}}} \ \sum_{\substack{(q_1,\dots,q_n)\in Q^n \text{ s.t.}\\ \{q_1,\dots,q_n\}|_m=N}} \prod_{i=1}^{n} \mathrm{val}(C_{\mathrm{trans}}(\mathcal{T}_i, \{q_i\}))\\
= &\sum_{\substack{(q_1,\dots,q_n)\in Q^n \text{ s.t.}\\ \text{ex. } N\subsetneq M \text{ s.t.}\\ M|_{\mathrm{rare}(M)}=N|_{\mathrm{rare}(M)} \text{ and}\\ \{q_1,\dots,q_n\}|_m=N}} \prod_{i=1}^{n} \mathrm{val}(C_{\mathrm{trans}}(\mathcal{T}_i, \{q_i\})) \ .
\end{aligned}$$

Consider the set of sequences $(q_1, \dots, q_n)$ for which there exists an $N \subsetneq M$ with $M|_{\mathrm{rare}(M)} = N|_{\mathrm{rare}(M)}$ and $\{q_1, \dots, q_n\}|_m = N$. This is exactly the set of superfluous sequences: These are the sequences where the number of rare states is the same as in M, but where at least one plentiful state of M is not present m times in the sequence. This concludes the correctness proof.

Depth and Fan-in: The second and third properties of the lemma follow since the construction of the circuit makes only local changes to the tree that only depend on the automaton. □

Turning Convolution Circuits Into Arithmetic Circuits. We now tackle the problem of evaluating convolution circuits using GapAC^0 circuits and, at the same time, address the problem of how histograms are encoded as numbers.

In order to represent a one-dimensional histogram h for a structure $\mathcal{A}$ using a single number, we imagine h to be stored in computer memory. Then $\mathrm{num}_b(h) = \sum_{i\in\{0,\dots,|A|\}} h[i]b^i$, where $\log_2 b$ is the word size of the memory, is a single number that represents the whole of the memory contents. For sufficiently large b, the bit representation of each $h[i]$ can be retrieved easily from a bitstring representation of $\mathrm{num}_b(h)$. For multidimensional histograms, we use a vector $b = (b_1, \dots, b_k)$ of bases and define the *number encoding of* h *with respect* *to b* as

$$\operatorname{num}_b(h) := \sum_{(i_1,\dots,i_k)\in\{0,\dots,|A|\}^k} h[i_1,\dots,i_k] b_1^{i_1}\cdots b_k^{i_k} \ .$$

The *bitstring representation of h with respect to b*, denoted by $\operatorname{str}_b(h)$, arises from viewing $\operatorname{num}_b(h)$ as a binary number that starts with the most significant bit. By choosing ever larger bases b_i, namely $b_1 = 2^{|A|}$, $b_2 = 2^{|A|^2}$, $b_3 = 2^{|A|^3}$, all histogram entries can be retrieved from $\operatorname{str}_b(h)$. However, we can also set a b_i to just 1. If we do so for all b_i, then the number $\operatorname{num}_b(h)$ is just the sum over all entries of the histogram. Setting the last ℓ many b_i to 1 while setting the first k many to ever larger values, we get a number that encodes for each index into the first k dimensions of the histogram the sum over all entries for the last ℓ dimensions.

Given two k-dimensional arrays C and D with ranges r and s, respectively, we have $\operatorname{num}_b(C+D) = \operatorname{num}_b(C) + \operatorname{num}_b(D)$ and also $\operatorname{num}_b(C-D) = \operatorname{num}_b(C) - \operatorname{num}_b(D)$. More importantly, we also have $\operatorname{num}_b(C*D) = \operatorname{num}_b(C)\cdot\operatorname{num}_b(D)$ from the derivation

$$\begin{aligned}
&\operatorname{num}_b(C*D)\\
&= \textstyle\sum_{\ell_1,\dots,\ell_k\in[r+s-1]} (C*D)[\ell_1,\dots,\ell_k] b_1^{\ell_1}\cdots b_k^{\ell_k}\\
&= \textstyle\sum_{\ell_1,\dots,\ell_k\in[r+s-1]} \ \sum_{i_1+j_1=\ell_1,\dots,i_k+j_k=\ell_k} C[i_1,\dots,i_k] D[j_1,\dots,j_k] b_1^{\ell_1}\cdots b_k^{\ell_k}\\
&= \textstyle\sum_{i_1,\dots,i_k\in[r] \text{ and } j_1,\dots,j_k\in[s]} C[i_1,\dots,i_k] D[j_1,\dots,j_k] b_1^{i_1+j_1}\cdots b_k^{i_k+j_k}\\
&= \textstyle\sum_{i_1,\dots,i_k\in[r]} C[i_1,\dots,i_k] b_1^{i_1}\cdots b_k^{i_k} \sum_{j_1,\dots,j_k\in[s]} D[j_1,\dots,j_k] b_1^{j_1}\cdots b_k^{j_k}\\
&= \operatorname{num}_b(C)\cdot\operatorname{num}_b(D).
\end{aligned}$$

With these observations, we can turn a convolution circuit into an arithmetic circuit for given bases, as stated by the following lemma:

LEMMA 4.11. *There is a* FO*-computable function that gets a convolution circuit C with n input gates as input and outputs an arithmetic circuit D with $n+k$ input gates, such that for all $b=(b_1,\dots,b_k)$ the following holds:*

$$\operatorname{num}_b\big(\operatorname{val}(C[B_1,\dots,B_n])\big) = \operatorname{val}(D[\operatorname{num}_b(B_1),\dots,\operatorname{num}_b(B_n),b_1,\dots,b_k]).$$

The circuit D will have the same topology as C, except that each constant gate gets replaced by a circuit of constant depth with $O(r^k \log m)$ gates where r is the range of the constant gate's arrays and m is the largest number in these arrays.

PROOF. The circuit D is obtained from C by replacing every addition gate of the convolution circuit by a normal addition gate in the arithmetic circuit, replacing every convolution gate by a multiplication gate, and replacing every constant gate with the constant array X attached to it by an arithmetic circuit evaluating the formula $\operatorname{num}_b(X) = \sum_{\ell_1,\dots,\ell_k\in[r]} X[\ell_1,\dots,\ell_k] b_1^{\ell_1}\cdots b_k^{\ell_k}$. The above circuit has constant depth and each number $X[\ell_1,\dots,\ell_k]$ can be expressed in constant depth using $\log m$ gates. □

To prove Theorem 1.7 (and, hence, Theorem 1.8) from the introduction we prove the following more general theorem:

THEOREM 4.12. *For every* MSO*-formula $\varphi(X_1, \dots, X_k)$ over some vocabulary τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* GapAC^0*-circuit family that, on input of a τ-structure $\mathcal{A}$ of tree depth at most d and a vector $b \in \mathbb{N}^k$ of bases, outputs* $\mathrm{num}_b(\mathrm{histogram}(\mathcal{A}, \varphi))$*.*

PROOF. Let us recapitulate the sequence of transformations introduced in this chapter: Starting with an MSO-formula φ over a vocabulary τ and $d \in \mathbb{N}$, by Lemmas 4.6 and 4.7 and Theorem 2.19 we can turn φ into a fixed multiset tree automaton A and we can turn (using an FO-computation) any logical structure $\mathcal{A}$ of tree depth at most d into an s-tree structure $\mathcal{T}$ of constant depth with $\mathrm{histogram}(\mathcal{A}, \varphi) = \mathrm{histogram}(S_A(\mathcal{T}, Q_a))$. By Lemma 4.10, we can turn the s-tree structure into a convolution circuit whose output is exactly the desired histogram. This circuit will have constant depth and polynomial size and, furthermore, all constants are from $[2]^k = \{0, 1\}^k$. By the preceding lemma, we can turn the convolution circuit into an arithmetic circuit that takes bases as (additional and only) inputs. The resulting arithmetic circuit has constant depth. In particular, it can be evaluated in GapAC^0 as claimed. □

4.4. Applications to Solving Number Problems

The theorems proved in the present section put problems into constant-depth circuit classes by using MSO-based definitions and considering their restriction to structures of bounded tree depth, or reductions to such problems.

Finding Matchings. In Example 3.24 we used an MSO-formula $\varphi'_{\mathrm{matching}}$ to define matchings on incidence representations of graphs. That means, for the incidence representations $\mathcal{G}_{\mathrm{inci}}$ of a graph and a variables assignment α_{inci} for it, we have $(\mathcal{G}_{\mathrm{inci}}, \alpha_{\mathrm{inci}}) \models \varphi'_{\mathrm{matching}}$ if, and only, if $\alpha_{\mathrm{inci}}(M)$ is a set of elements from the universe of $\mathcal{G}_{\mathrm{inci}}$ that stands for a set of edges that is a matching. By restricting the admissible inputs to have bounded tree depth, we can apply Theorem 1.7 to show that the problem of counting the number of perfect matchings in tree-depth-bounded graphs lies in DLOGTIME-uniform GapAC^0. Moreover, deciding whether a given graph of bounded tree depth has a matching of some given size is TC^0-complete; in contrast, the same problem on tree-width-bounded graphs is L-complete (Section 6.4).

THEOREM 4.13. *For every $d \in \mathbb{N}$, the language*

$$\text{EXACT-MATCHING-TREE-DEPTH-}d := \{(\mathcal{G}, s) \mid \mathrm{td}(\mathcal{G}) \leq d \text{ and } \mathcal{G} \text{ has a matching of size } s\}$$

is TC^0*-complete under* AC^0*-Turing-reductions.*

PROOF. For proving containedness, the TC^0-circuit first checks whether the tree depth of $\mathcal{G}$ is at most d, which can even be done in AC^0. Next, the circuit computes the incidence representation of $\mathcal{G}$ and computes the solution histogram h of $\mathcal{G}_{\mathrm{inci}}$ and $\varphi'_{\mathrm{matching}}$. If $h[s] > 0$, it accepts and rejects, otherwise.

We are only left to prove TC^0-hardness under AC^0-Turing-reductions; we will do this by a transformation from MAJORITY. For an input $x_1 \dots x_n \in \{0, 1\}^n$ to MAJORITY, build a graph $\mathcal{G}$ that is the union of n components $\mathcal{G}_1$ to $\mathcal{G}_n$. Each component G_i is either just a single vertex, if $x_i = 0$, or the graph $\circ\!\!-\!\!\circ$, if $x_i = 1$. The graph contains a matching of size $\lfloor n/2 \rfloor + 1$ exactly if more than half of the entries of $x_1 \dots x_n$ are 1. □

From Numbers to Structures. Even when problems are not directly definable in MSO, we may use reductions to problems that are in order to place them in uniform constant-depth circuit classes by using theorems related to tree-depth-bounded input structures. An example are problems whose inputs are sequences of numbers and we ask whether we can add up the numbers in a certain way. In the introduction we saw that MAJORITY can be reduced to a problem that fits to Theorem 1.8. Similar arguments work for the unary versions of the subset sum problem, UNARY-SUBSETSUM :=

$$\{1^{a_1}0\,1^{a_2}0\ldots 1^{a_n}00\,1^b \mid \exists I \subseteq \{1,\ldots,n\} \text{ with } \textstyle\sum_{i\in I} a_i = b\}\ ,$$

and the knapsack problem, UNARY-KNAPSACK :=

$$\{1^{w_1}0\,1^{w_2}0\ldots 1^{w_n}00\,1^{v_1}0\,1^{v_2}0\ldots 1^{v_n}00\,1^{w}00\,1^{v} \mid \\ \exists I \subseteq \{1,\ldots,n\} \text{ with } \sum_{i\in I} w_i \le w \text{ and } \sum_{i\in I} v_i \ge v\}\ .$$

The quest of resolving the complexity of these problems has a long history: If the input numbers are encoded in binary, both problems are known to be NP-complete. They become polynomial time solvable using dynamic tables if the numbers are encoded in unary as above; in fact, they can be shown to lie in NL in this setting via solving a reachability problem that is related to the tables. Inspired by a conjecture of Cook (1985) that "a problem in NL which is probably not complete is the knapsack problem with unary weights," a line of research began to capture the complexity of UNARY-SUBSETSUM and UNARY-KNAPSACK with specialized complexity classes lying between L and NL (Ibarra et al., 1988; Cho and Huynh, 1988; Jenner, 1995), see also the earlier work of Monien (1980). Elberfeld, Jakoby, and Tantau (2010) proved UNARY-SUBSETSUM $\in$ L, which was independently also shown by Kane (2010). While the later paper presents a direct algebraic approach for solving UNARY-SUBSETSUM, the earlier paper maps an instance $1^{a_1}0\,1^{a_2}0\ldots 1^{a_n}00\,1^b$ of UNARY-SUBSETSUM to a forest $\mathcal{F} = (V, E^{\mathcal{F}})$ consisting of n stars where the ith star has a_i vertices and uses an MSO-formula $\varphi(X)$ that forces solution sets $S \subseteq V$ to cover each star either completely or not at all. Since this reduction is FO-computable and the forest has tree depth 2, we can apply Theorem 1.8 to get UNARY-SUBSETSUM $\in \mathrm{TC}^0$; we have $1^{a_1}0\,1^{a_1}0\ldots 1^{a_n}00\,1^b \in$ UNARY-SUBSETSUM exactly if $\mathrm{histogram}(\mathcal{F},\varphi)[b] > 0$. UNARY-SUBSETSUM is also hard for TC^0 since asking whether the majority of entries of a binary string of length n are set to 1 is equivalent to ask whether we can single out a subset of positions whose entries add up to $\lfloor n/2 \rfloor + 1$.

Number problems like SUBSETSUM can be phrased as the task of solving a constant number of linear equations where the solution values that are assigned to the variables are bounded by some threshold: For example, SUBSETSUM is the problem of deciding, for a given vector $a \in \{0,1\}^n$ and a target value $b \in \mathbb{N}_0$ whether there exists a vector $s \in \{0,1\}^n$ with $a^T s = b$. The application of Theorem 1.8 to UNARY-SUBSETSUM can be adjusted to solve systems of a constant number of linear equations where the entries of the solution vector are bounded by a number given in the input. Formally, the input to this problem, which we call ℓ-INTEGER-LINEAR-EQUATIONS, is a linear equation system with integer coefficients $(A, b) \in \mathbb{Z}^{\ell\times m} \times \mathbb{N}_0^{\ell}$ and an upper bound $t \in \mathbb{N}$. The problem asks whether there is an $s \in \{0,\ldots,t\}^m$ with $As = b$. Like SUBSETSUM, this problem is well known to be NP-complete for any ℓ if the input numbers are encoded in binary and solvable in polynomial time if the input numbers are encoded in unary (this fact is discussed, for example, by Papadimitriou (1994)). Its unary version, proven to lie in L by Stockhusen (2011), is TC^0-complete.

THEOREM 4.14. *For each $\ell \in \mathbb{N}$, ℓ-INTEGER-LINEAR-EQUATIONS with input numbers given in unary is complete for TC^0 under AC^0-Turing-reductions.*

PROOF. For the upper bound, reduce ℓ-INTEGER-LINEAR-EQUATIONS with coefficients that are given in unary to an MSO-definable problem on structures of tree depth 4, which can be solved in TC^0 by applying Theorem 1.8. We first discuss the case that the coefficients of the equation system are from $\mathbb{N}_0$ and later extend this to coefficients from $\mathbb{Z}$.

Given an equation system $(A, b) \in \mathbb{N}_0^{\ell \times m} \times \mathbb{N}_0^{\ell}$ and a bound $t \in \mathbb{N}$, we build a forest $\mathcal{F}$ of m trees $\mathcal{T}_1, \ldots, \mathcal{T}_m$. Each tree $\mathcal{T}_i$ has a root r_i to which we attach $t+1$ child nodes $v_{i,0}, \ldots, v_{i,t}$. Choosing a node $v_{i,j}$ will later correspond to choosing the value j for the variable x_i. To each $v_{i,j}$ we attach ℓ children $a_{i,j,1}, \ldots, a_{i,j,\ell}$. We establish ℓ unary predicates $P_1^{\mathcal{F}}, \ldots, P_\ell^{\mathcal{F}}$ and put each node $a_{i,j,k}$ into the predicate $P_k^{\mathcal{F}}$. To each $a_{i,j,k}$, we attach $j \cdot A[k,i]$ leaf nodes.

To define the problem, we use an MSO-formula $\varphi(X_1, \ldots, X_\ell)$ whose solutions $S_1, \ldots, S_\ell$ must satisfy the following properties: There exists a set of vertices $\{v_{1,j_1}, \ldots, v_{m,j_m}\}$ (that means, a value for each variable), such that for each $k \in \{1, \ldots, \ell\}$ (each equation with index k), exactly the leaves below the nodes $a_{i,j_1,k}, \ldots, a_{m,j_m,k}$ are in X_k. The last property can be defined in MSO using the predicates $P_k^{\mathcal{F}}$. As a result, the number of elements in each set S_k equals the value of the k's equation when evaluated for the assignment of values $x_1 = j_1 \ldots x_m = j_m$ to the variables. Thus, $\mathrm{histogram}(\mathcal{F}, \varphi)[b] > 0$ exactly if $(A, b) \in \ell$-INTEGER-LINEAR-EQUATIONS.

To solve the general problem with integer coefficients (and not only positive integers) we first consider the absolute value of all coefficients and construct the structure as described above. Then we establish two unary predicates $P_+^{\mathcal{F}}$ and $P_-^{\mathcal{F}}$ that are used to label the $a_{i,j,k}$ nodes that stand for positive and negative coefficients, respectively. This means, if $A[k,i]$ is a positive coefficient, we set $a_{i,j,k} \in P_+^{\mathcal{F}}$ for all $j \in \{0, \ldots, t\}$, and $a_{i,j,k} \in P_-^{\mathcal{F}}$ otherwise. We extend the previous formula to the formula $\varphi(X_{1,+}, \ldots, X_{\ell,+}, X_{1,-}, \ldots, X_{\ell,-})$ that forces the same requirements, except that we put leaves below nodes from $P_+^{\mathcal{F}}$ into the $X_{k,+}$ sets and leaves below nodes from $P_-^{\mathcal{F}}$ into the $X_{k,-}$ sets. The equation system has a solution if there exists an index $i = i_1, \ldots, i_\ell, i_{\ell+1}, \ldots, i_{2\ell}$ with $\mathrm{histogram}(\mathcal{F}, \varphi)[i] > 0$ and $b[1] = i_1 - i_{\ell+1}, \ldots, b[\ell] = i_\ell - i_{2\ell}$.

Hardness follows by encoding UNARY-SUBSETSUM instances as 1-INTEGER-LINEAR-EQUATIONS instances. □

By using AC^0-Turing-reductions to query for the existence of a solution in the histogram that satisfies the optimization criterion imposed by the knapsack problem, the previous theorem makes it easy to prove UNARY-KNAPSACK $\in \mathrm{TC}^0$.

Papadimitriou (1981) showed that the dynamic-programming-based technique of solving a constant number of linear equations with integer coefficients that are given in unary still works if there is no bound t on the entries of the solution vector. His key argument states that if (A, b) is solvable, then also by a vector whose entries are polynomial in the length of the unary encoding of (A, b). By equipping inputs with this bound and applying the previous theorem, we can also solve this problem variant in TC^0.

CHAPTER 5

Tree Decompositions as Terms and Logarithmic-Depth Circuits

In the present chapter we prove the theorems that involve circuits of logarithmic depth and structures that are given together with width-bounded tree decompositions whose underlying trees are encoded as terms. In the context of research on logarithmic-depth circuits, trees in term representation are a natural form of input because many problems on trees, including problems studied in the present thesis, become L-complete if the input tree is given as a pointer structure, but are solvable in logarithmic depth if it is given as a term. A problem of this kind is evaluating Boolean sentences; it is L-complete if the tree underlying the input sentence is encoded as a pointer structure and NC^1-complete if it is given as a term.

In many works on logarithmic-depth circuits (including the present), a central problem is that a logarithmic-depth circuit cannot work on the term representation directly when the encoded tree has large depth. Instead, some sort of balancing must be done. To deal with this issue, the most common approach is to recursively divide the tree into parts of almost the same size, compute intermediate values for the different parts, and put them together. Finding appropriate recursive separators of the tree in a uniform manner is a highly involved problem; and even when the separators have been found, it is difficult to implement the recursion in such a way that intermediate values are passed around in the correct way. This way of *simultaneously balancing and processing the input* was developed by Buss, Cook, Gupta, and Ramachandran (1992) and applied to the problem of evaluating Boolean and arithmetic sentences. It was later extended by Krebs, Limaye, and Mahajan (2010) to count the number of accepting computations of visible pushdown automata in $\#\mathrm{NC}^1$. While Krebs et al. left the balancing approach of Buss et al. untouched, they adjusted the way of how partial solutions are combined based on automaton-dependent algebraic operations. Instead of trying to push this approach even further to the needs of solving MSO-definable problems, we will attack the balancing issue at an earlier stage: We balance the tree decomposition before evaluating the problem-defining MSO-formula. For this we adapt an NC-approach of Bodlaender (1989) that is based on the classical tree contraction method and show that it can be implemented in FTC^0 in case of trees that are given as terms. This has the advantage that it *separates the balancing from the actual problem-solving step* and, thus, results in simpler proofs.

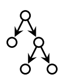
The proofs of Theorems 1.4 and 1.5 almost follow along the same lines as those of the previous chapter, which involved the following steps: (1) Compute tree decompositions of the input structures. (2) Move from formulæ on the input structures to formulæ on trees. (3) Move from formulæ on trees to equivalent tree automata. (4) Move from the evaluation of tree automata to evaluating (arithmetic) circuits. Clearly, the first step is no longer necessary in the setting of the present section since the tree decomposition is already part of the input, but instead of constructing a tree decomposition, we balance an existing one. All of the other steps are still possible when the depth of the tree is no longer bounded. Moreover, since they increase the depth and degree of the considered trees only by a constant factor, we will get the desired Boolean and arithmetic circuits of logarithmic depth.

The present chapter is organized as follows: In Section 5.1 we review term representations. In Section 5.2 we prove the technical result on how a balanced width-3 tree decomposition of any tree can be computed in FTC^0, and how this can be used to balance tree decompositions. At the end of this section we prove Theorems 1.4 and 1.5. In Section 5.3 we describe how the theorems can be used to simulate computations of different kinds of automata.

5.1. Representing the Ancestor Relation of Trees by Terms

Up to now, the details of how tree decompositions are encoded as strings was not important; indeed, in the context of bounded tree depth almost any encoding of the input graph and of tree decompositions will work since they can easily be transformed into one another. In the context of logarithmic-depth circuits, however, it is well known that it is crucial that the ancestor relation of the tree is made accessible to the circuits, rather than just a pointer structure or an adjacency matrix. There are two different ways of encoding this relation: Explicitly as a list of pairs or implicitly as a bracket structure. The two representations can be transformed into one another using TC^0-circuits and we will use both of them.

Definition of Ancestor and Term Representations. In the following, let T be a tree like the one shown right. The set of *term representations* of T is the set of all strings over the two-letter alphabet $\{\,[\,,\,]\,\}$ that can be obtained recursively as follows: If the subtrees rooted at the children of T's root are $T_1, \dots, T_n$ (in some arbitrary order) and if t_1 to t_n are term representations of T_1 to T_n, respectively, then $[t_1 \dots t_n]$ is a term representation of T. For instance, the tree right has the two term presentations [[ ] [[ ] [ ]]] and [[[ ] [ ]] [ ]].



An *ancestor representation* of a tree T whose nodes are encoded as bitstrings is a strings over the alphabet $\{\,(\,,\,)\,,0,1,\#\}$ that is a concatenation of all strings $(u\#v)$ where u is an ancestor of v in T. For the tree above with nodes labeled with bitstrings a to e according to an in-order traversal (so b is the root), a possible ancestor representation is $(b\#a)(b\#c)(b\#d)(b\#e)(d\#c)(d\#e)$.

In order to encode a tree decomposition using either term representations or ancestor representations, we must also encode the bags. For term representations, this can be done, for instance, by first encoding individual bags in some sensible way as strings, and then encoding the bag function B as a string of blocks, such that the ith block encodes the bag $B(n)$ exactly if the ith symbol of the term representation is the opening bracket of the node n. (The contents of blocks at positions of closing brackets are arbitrary and ignored.) For ancestor representations, we use the same encoding, only i is now the number of bitstrings different from n that precede n in the ancestor representation.

Converting Representations. In a term representation t, the elements of the set $V(T)$ are not explicitly encoded anywhere; we only access them indirectly through the fact that each node n has a unique position $L(n)$ in t were the opening bracket of this node is located. For this reason, for convenience we may assume that $V(T) = \{1, \dots, q\}$ is just a set of numbers and we may assume that L is a monotone function. For a node n with exactly two children, let us call the child of n that comes first in the term representation the *left* child and the one that comes second the *right* child. For each node n, let $R(n)$ be the position of the *closing* bracket of its representation in t. For instance, if $t = [\,[\,[\,[\,]\,[\,]\,]\,[\,]\,]$, for the first child c of the root we have $L(c) = 2$ and $R(c) = 7$. It is well known that L and R are computable in FTC^0; see, for example, (Gottlob et al., 2005): The number $L(n)$ for $n \in \{1, \dots, q\}$ is the position of the opening bracket for which there are exactly $n - 1$ opening brackets to its left. The number $R(n)$ is the first position to the right of $L(n)$ such that between $L(n)$ and $R(n)$ the number of opening and closing brackets is balanced (that is, equal). Given a node $n \in V(T)$, using $L(n)$ and $R(n)$ we can easily determine whether a node n is a left or a right child or the root. Provided they exist, we can also compute its *parent* node $p(n)$, its *grandparent node* $g(n) = p(p(n))$, and also its *sibling node* $s(n)$ in FTC^0. We can also decide the *ancestor relation* for two nodes n and m by testing whether $[L(m), R(m)]$ is contained in $[L(n), R(n)]$. The *least common ancestor* $\mathrm{lca}(n, m)$ is the least node (with respect to the ancestor relation) that is an ancestor of both n and m. These observations imply Lemma 5.1; also the other direction is possible (Lemma 5.2).

LEMMA 5.1. *There is a function in* DLOGTIME-*uniform* FTC^0 *that maps every term representation of a tree T to an ancestor representation of T (for an appropriate naming of the nodes of T).*

LEMMA 5.2. *There is a function in* DLOGTIME-*uniform* FTC^0 *that maps every ancestor representation of a tree T to a term representation of T.*

PROOF. It suffices to show that a TC^0-circuit can compute, for every node $n \in V(T)$, two positions $L(n)$ and $R(n)$ such that the string t that has an opening bracket at each $L(n)$ and a closing bracket at each $R(n)$ is a term representation of T. Let us order the nodes of T as follows: For each node n let the children of n be ordered according to the order in which they first appear in the ancestor representation. This induces a specific term representation t of T and we will compute this particular representation. Observe that with respect to this ordering, the position $L(n)$ of a node n can be expressed as follows: Consider the path $n_1, n_2, \dots, n_k$ from the root to the parent of n. For each node on this path there is one opening bracket to the left of $L(n)$. Furthermore, for every sibling s of any n_i that comes before n_i with respect to the ordering we fixed earlier, every node in the subtree rooted at s contributes an opening and a closing bracket to the left of $L(n)$. For the computation of $L(n)$ using FTC^0-circuits, first observe that given a node $n \in V(T)$ we can compute the number d_n of nodes of T that are descendants of n. Next, we can also compute the parent node of n in T and, thus, also the set of all of its siblings and also which siblings are before n with respect to the ordering. For a node n, let p_n be the number of nodes in the subtrees rooted at the siblings of n that precede n in the ordering of the siblings. Then $L(n)$ is the sum of the numbers $2p_{n_i}+1$, where the n_i are all ancestors of n, plus 1 for the opening bracket of n. The number $R(n)$ is given by $L(n) + 2d_n + 1$. $\square$

5.2. Balancing Tree Decompositions in Constant Depth

A tree is *binary* if every inner node has *exactly* two children. In a binary tree, we may wish to distinguish between the *left* and the *right* child of a node. In such a case, we call T a *binary tree with distinguished left and right children.* Formally, T is a labeled tree where for every inner node exactly one of the children has the label “is left child.” A tree is *balanced* if all root-to-leaf paths have the same length. Note that balanced binary trees have depth $\lfloor \log_2(|V(T)|) \rfloor + 1$.

In the present section we prove the following lemma:

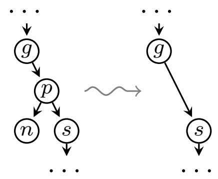
Lemma 5.3. *Let $w \in \mathbb{N}$ be any tree width bound. There is a* DLOGTIME*-uniform* TC^0*-circuit family that gets a term-represented width-w tree decomposition (T,B) of a graph G as input and outputs a term-represented width-$(4w+3)$ tree decomposition (T',B') of G where T' is a binary and balanced tree.*

In order to prove this lemma, we adapt the NC-approach of Bodlaender (1989) for computing balanced tree decompositions for trees and show that it can be implemented using constant-depth threshold circuits if the input tree is given as a term. The approach of Bodlaender is based on the classical method of tree contraction (Miller and Reif, 1985; Abrahamson et al., 1989). Thus, we first review the tree contraction method and discuss its complexity, before we use it to balance trees.

Contracting Trees in Constant Depth. The tree contraction method is commonly used in the context of PRAM algorithms to solve problems on tree structures in logarithmic parallel time. We review this method and show how to implement it using FTC^0-circuits for input trees that are given in term representation. The tree contraction method starts with a binary input tree T and iteratively proceeds as follows: Consider every second leaf (with respect to the left-to-right ordering induced by the term representation of T). Among these leaves, (conceptually) build two sets L^{left} and L^{right} of leaves that are left and right children, respectively. Starting with, say, L^{left}, apply the following *prune-and-bypass operation* (also known as the *rake operation*) in parallel to all its members.

Definition 5.4. Let T be a tree and let n be a leaf of T. Let p be its parent, s its sibling, and g its grandparent. We call the tuple (n,s,p,g) the *contraction tuple* of n. The tree resulting from the *prune-and-bypass* operation applied to this tuple is the tree in which we remove both the node n (this is called *pruning*) and its parent node p, making the sibling s the new child of g, taking the place of p (this is called *bypassing*).

An example of the prune-and-bypass operation is shown right. Note that, because only every second leaf is considered and since all leaves are left children, the contractions can be applied in parallel to all members of L^{left}. Next, apply the prune-and-bypass operation to all leaves in L^{right} in parallel. The tree that results obviously has half the number of leaves as T used to have and we can reapply the contraction operation. After a logarithmic number of steps, the tree will have shrunk to a single node.



Let T be a tree with $m = 2^k$ leaves. Number the leaves of T from left to right as $\{l_1,\ldots,l_m\}$. The leaves to be deleted in round i are all that have an even number among those that remain. Thus, in the first round, $\{l_2, l_4, l_6, \dots, l_m\}$ will be removed, in the second round $\{l_3, l_7, l_{11}, l_{15}, \dots, l_{m-1}\}$, and so on. In general, in round ith we remove the leaves $\{l_{2^{i-1}+1+j2^i} \mid j \in \{0, \dots, 2^{k-i}-1\}\}$.

For each set S of leaves to be removed, we first compute the contraction tuples of all leaves in S that are left children and apply the contraction step to them. In the resulting tree, we compute the contraction tuples in S that are right children and, again, apply the contraction step to all of them. Thus, it actually takes two rounds to halve the number of leaves. Let us call the sequence of trees generated in this way T_1, T_2, T_3, …, T_t, where $T_1 = T$ and in T_t there are exactly two leaves, and in every second tree the number of leaves has exactly halved. For a leaf l, let rank(l) be the maximal number i such that l is still an element of T_i. Clearly, this rank function is computable in FTC^0. An example is given in Figure 5.1.

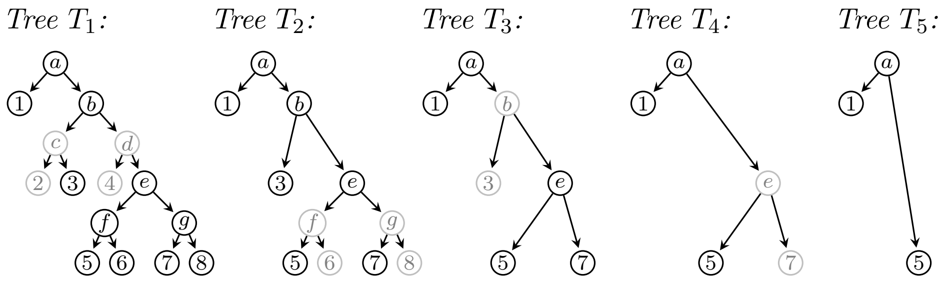
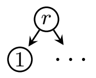


FIGURE 5.1. Example of a tree $T = T_1$ and the trees T_2, T_3, T_4 and T_5. In T_1 and T_2 the prune-and-bypass operation is applied to the even-numbered leaves that are left children (2 and 4) or right children (6 and 8) of their parents in T_1, respectively. In T_3 and T_4 the operation is applied to the even-numbered leaves that are left children (3) or right children (5) of their parents in T_3, respectively. The gray nodes indicate the nodes that will be pruned and bypassed in each tree.

For technical reasons, in the following we require that T has the following properties: (a) Every inner node has exactly two children, (b) the number of leaves is $m = 2^k$ for some k, (c) the left child of the root is a leaf, like shown on the right. This ensures that the root's left child comes first in the leaf ordering and that it will never be pruned. Thus, all pruning is done on the right subtree of the root. This implies, however, that in all trees up to T_t, the root is not the parent of a pruned node. Thus, by stopping our construction at T_t, we ensure that $g(l)$ is always well-defined for all pruned leaves l.



The following observations concerning the trees T_i are due to Buss (1993).

(1) The leaves of each T_i are precisely the leaves of T with rank at least i.
(2) If x and y are nodes in T_i, then their least common ancestor in T_i is the same as their least common ancestor in T.
(3) An inner node of T is a node of T_i if, and only if, it is the least common ancestor of two leaves of rank at least i.

Buss infers from these properties that the T_i can be computed in NC^1. However, since computing ranks and least common ancestors can even be done in FTC^0, as we argued earlier, the T_i can actually be computed in FTC^0.

Balancing Trees via Tree Contractions. Before applying the tree contraction method to balance trees, we prove two technical lemmas that help us to make trees of bounded degree and logarithmic depth binary and balanced.

For this, we define embeddings of trees into trees: An *embedding* of a tree T into a tree T' is an injective mapping $\iota\colon V(T) \to V(T')$ such that for every pair of nodes $a, b \in V(T)$ there is a path from a to b in T if, and only if, there is a path from $\iota(a)$ to $\iota(b)$ in T', and the root of T is mapped to the root of T'. Given two embeddings $\iota\colon V(T) \to V(T')$ and $\kappa\colon V(T') \to V(T'')$, note that the composition $\kappa \circ \iota\colon V(T) \to V(T'')$ is also an embedding. Given an embedding $\iota\colon V(T) \to V(T')$, we call a node $w \in V(T')$ a *white* node if there is no node $n \in V(T)$ with $\iota(n) = w$. An example of an embedding is shown in Figure 5.2.

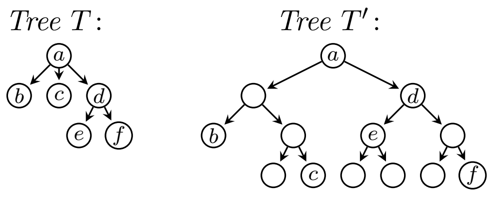


FIGURE 5.2. The embedding maps each node of the left tree to the node with the same label in the right tree. The unlabeled nodes are exactly the white nodes.

LEMMA 5.5. *There is a function in* DLOGTIME-*uniform* FTC^0 *that maps the ancestor representation of any tree T of bounded degree to an ancestor representation of a tree T' together with an encoding of an embedding $\iota\colon T \to T'$ such that T' is binary.*

PROOF. For every node n of too high degree $d > 2$, introduce $d - 2$ new white nodes and connect them so that they form a path starting at n and make the children of n children of these nodes in such a way that n and all new nodes have degree 2. For nodes of degree 1, just add a second child that is white. □

LEMMA 5.6. *For every c, there is a function in* DLOGTIME-*uniform* FTC^0 *that maps the ancestor representation of any tree T of degree c and depth $c \log_2 |V(T)|$ to an ancestor representation of a tree T' together with an encoding of an embedding $\iota\colon T \to T'$ such that T' is binary and balanced.*

PROOF. First, using Lemma 5.5, we may assume that the degree of all inner nodes of T is 2. We compute the height h of the tree (the maximum length of a path from the root to a leaf). Now, for every leaf l of T with distance $h' < h$ to the root, we add a new binary balanced tree of depth $h - h'$ with l as its root. All added nodes are white. The added trees themselves do not depend on the input and they inherit their ancestors from the leaf l. □

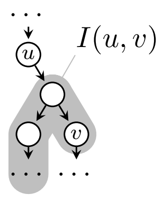
The upcoming lemma follows almost immediately from the FTC^0 computation of tree contractions observed above in conjunction with ideas from Bodlaender (1989) who showed how to compute logarithmic-depth tree decompositions with the help of the tree contraction method in NC. Thus, the proof of the following lemma is mostly added to give a complete picture of why the stated claims hold, rather than presenting new techniques.

Lemma 5.7. *There is a function in* DLOGTIME-*uniform* FTC^0 *that, on input of a term representation of a tree T, outputs a term representation of a width-3 tree decomposition (S, B) of T where S is a balanced binary tree.*

Proof. Our input is a term representation t of a tree T. As a preprocessing step, we compute an embedding of T into another tree T' such that the three technical properties from above are satisfied: (a) Every inner node has exactly two children, (b) the number of leaves is $m = 2^k$ for some k, (c) the left child of the root r is a leaf. To achieve this, we apply Lemma 5.5 and add some additional white nodes as needed. Clearly, this can be done using TC^0-circuits. Note that T' now has exactly $2m - 1$ nodes.

Our objective is to compute a width-3 tree decomposition (S, B) of T that is balanced. In the following, we only describe how we can compute a width-3 tree decomposition of T' that has bounded degree and logarithmic depth. However, Lemma 5.6 allows us to embed S into a balanced tree and extending the bag labeling to the white nodes w of this balanced tree can be done, for instance, by setting $B(w) = B(n)$ for the first non-white ancestor n of w. Thus, we must now compute an ancestor representation of a width-3 tree decomposition (S, B) of T of constant degree and logarithmic depth.

We start with some observations concerning how contraction tuples can be related. Given two nodes $u, v \in V(T')$ such that u is an ancestor of v let $I(u, v)$ be the set of nodes $n \in V(T')$, such that u is an ancestor of n, but not v; an example of $I(u, v)$ is shown on the right. For every contraction tuple $c = (n, s, p, g)$, observe that the sets $I(g, p)$, $I(p, n)$, and $I(p, s)$ are disjoint because p is the least common ancestor of n and s in T. Furthermore, because n is a leaf, $I(g, s) = I(g, p) \cup I(p, n) \cup I(p, s)$. Let us also write $I(c)$ for this disjoint union, see Figure 5.3 for an example.



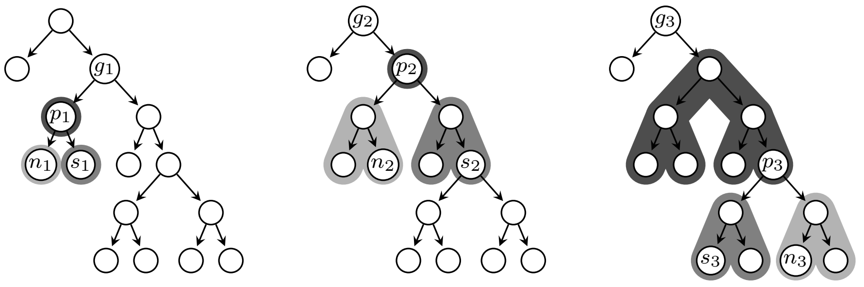


Figure 5.3. Three contraction tuples $c_1 = (n_1, s_1, p_1, g_1)$ to $c_3 = (n_3, s_3, p_3, g_3)$ of the tree T from Figure 5.1. Note how in each tree the sets $I(p_i, n_i)$, $I(p_i, s_i)$, and $I(g_i, p_i)$ are disjoint and that their union $I(c_i)$ is exactly one of the sets for the next tuple.

When moving from T_i to T_{i+1}, each edge is either copied or contracted. This implies that for all $i \leq j$ and all edges $(u, v) \in E(T_i)$ there is a unique edge $(x, y) \in E(T_j)$ such that $I(u, v) \subseteq I(x, y)$. Given two different contraction tuples $c = (n, s, p, g)$ and $c' = (n', s', p', g')$, the sets $I(c)$ and $I(c')$ are either disjoint or one is contained in the other. To see this, let $r = \mathrm{rank}(n)$, $r' = \mathrm{rank}(n')$, and assume $r < r'$. Then $I(c) = I(g, s)$ and as we just argued there is a unique edge (x, y) in $E(T_{r'})$ with $I(g, s) \subseteq I(x, y)$. If $(x, y) = (g', s')$, then $I(c) \subseteq I(c')$; and if (x,y) and (g',s') are different edges, then $I(x,y)$ and $I(g',s')$ are disjoint and hence also $I(g,s)$ and $I(g',s')$ since $I(g,s) \subseteq I(x,y)$.

Let the nodes of the decomposition's tree S be exactly the contraction tuples. There is an edge $(c,c') \in E(S)$ if $I(c') \subsetneq I(c)$ and there is no c'' with $I(c') \subsetneq I(c'') \subsetneq I(c)$. By the above properties of contraction tuples, S is, indeed, a tree. We attach the bag $B(c) = \{n,s,p,g\}$ to each tuple $c = (n,s,p,g)$.

We claim that (S,B) is a width-3 tree decomposition of T'. First, its width is clearly at most 3. Second, it has the covering property: Except for two special edges, for every edge $(u,v) \in E(T)$ there is a first i such that $(u,v) \in E(T_i)$, but $(u,v) \notin E(T_{i+1})$, because edges are copied from each T_i to the next until they become part of a contraction tuple c. But, then, by definition, $u,v \in B(c)$. (The only two exceptional edges are those still present in the last T_t. To cover these, we can add one additional bag at the very end to the root. We ignore these edges in the following discussion.) Third, we have to prove the connectedness condition: For this, let $v \in V(T)$ be a fixed node and let $c = (n,s,p,g)$ be a node of S with $v \in B(c)$. Then v is one of n, s, p, or g by construction. Consider the parent c' of c in S. Suppose c was contracted in tree T_i and $c' = (n',s',p',g')$ was contracted in some later T_j. Then in T_j there is still the edge (g,s) and we have $(g,s) = (g',p')$ or $(g,s) = (p',n')$ or $(g,s) = (p',s')$. This means that if v was either n or p, we have $v \notin B(c')$ and if v was either g or s, we have $v \in B(c')$. Repeating this argument, let $c_0 = (n_0,s_0,p_0,g_0)$ be the first ancestor of c such that $v \in \{n_0,p_0\}$. Then $v \in B(c'')$ holds for all ancestors of c up to c_0, but not for any ancestors of c_0. Now, starting from any two nodes c and d with $v \in B(c)$ and $v \in B(d)$, suppose we had $c_0 \neq d_0$. Then $v \in I(g(c_0), s(c_0))$ would hold and also $v \in I(g(d_0), s(d_0))$, but these sets are disjoint. This shows that v lies in every bag on the path from both c and d to a common ancestor. Hence, the set of all nodes of S whose bags contain v is connected.

We already saw that the ancestor relation of S is computable in TC^0. The maximum degree of any node of S is three: A contraction tuple $c = (n,s,p,g)$ can only be a direct child of another contraction tuple c' if the edge (g,s) resulting from contracting c is one of the three edges contracted by c'. □

LEMMA 5.8. *Let G be a graph, let (T,B) be a width-w tree decomposition of G, and let (T',B') be a width-w' tree decomposition of T. Then (T',B'') with $B''(n) = \bigcup_{x \in B'(n)} B(x)$ is a width-$(ww'+w+w')$ tree decomposition of G.*

PROOF. The width bound of (T',B'') holds by its definition. To prove the covering property, note that every bag $B(x)$ that is present in (T,B) is a subset of some $B''(n)$ for some $n \in V(T')$. To prove the connectedness property, consider any node $v \in V(G)$. By the connectedness condition for (T,B), the subgraph $T_v = T[\{x \mid v \in B(x)\}]$ is connected. By the connectedness condition for (T',B'), for each node $x \in V(T_v)$ the subgraph $T_x = T'[\{n \mid x \in B'(n)\}]$ is also connected, and by the covering property of (T',B') for every $\{x,y\} \in E(T_v)$ the trees T_x and T_y share at least one node. Hence, since T_v is connected, the union of all T_x for $x \in V(T_v)$ is connected and this union is exactly $T[\{n \mid v \in \bigcup_{x \in B'(n)} B(x)\}]$. □

Finally, we get the proof of Lemma 5.3 as a corollary.

PROOF OF LEMMA 5.3. Apply Lemma 5.8 to Lemma 5.7. □

Decision and Counting Using Logarithmic-Depth Circuits. We are now ready to prove the Theorems 1.4 and 1.5 from the introduction. The proofs follow the exact same lines as for the theorems related to constant-depth circuits, except that we no longer need to compute a tree decomposition, but balance an existing one. The crucial point is to show the following modified version of Lemma 4.7:

LEMMA 5.9. *Let $\varphi(X_1, \dots, X_k)$ be an* MSO*-formula over some vocabulary τ and $w \in \mathbb{N}$. There is an $s \in \mathbb{N}_0$, an* MSO*-formula $\psi(X_1, \dots, X_k)$ over $\tau_{s\text{-tree}}$, and a function in* DLOGTIME*-uniform* FTC^0 *that, on input of any τ-structure $\mathcal{A}$ with universe A and a width-w tree decomposition (T, B) for $\mathcal{A}$ where T is given in term representation, produces an s-tree structure $\mathcal{T}$ and a term representation of $\mathcal{T}$, such that*

(1) the depth of $\mathcal{T}$ equals the depth of T plus 1, and

(2) we have $\mathrm{histogram}(\mathcal{A}, \varphi)[s] = \mathrm{histogram}(\mathcal{T}, \psi)[s]$ *for all indices $s \in \{0, \dots, |A|\}^k$ and all other entries in the array* $\mathrm{histogram}(\mathcal{T}, \psi)$ *are 0.*

PROOF. The proof structure is identical to that of Lemma 4.7. The only difference is that the input tree is given as a term representation and we must output the resulting tree also as a term representation. For this, we must adjust the representation so that the same number of $w+1$ leaves (the element nodes) are added to all nodes of the tree. This can be achieved in TC^0 using ancestor representations. □

Just as in the proof of Theorem 4.12, in order to prove Theorem 1.5, we actually prove a stronger theorem were the bases are part of the input:

THEOREM 5.10. *For every* MSO*-formula $\varphi(X_1, \dots, X_k)$ over some vocabulary τ and every $w \in \mathbb{N}$, there is a* DLOGTIME*-uniform* #NC^1*-circuit family that, on input of a τ-structure $\mathcal{A}$ along with a width-w tree decomposition in term representation for $\mathcal{A}$ and bases $b \in \mathbb{N}^k$, outputs* $\mathrm{num}_b(\mathrm{histogram}(\mathcal{A}, \varphi))$.

PROOF. After balancing the given tree decomposition using Lemma 5.3, we use Lemma 5.9 to turn $\mathcal{A}$ into an s-tree structure $\mathcal{T}$ and φ into an equivalent MSO-formula ψ. The node degree of $\mathcal{T}$ is bounded in terms of the width of (T, B). We use Theorem 2.19 and consider an equivalent tree automaton A for ψ and extend its multiplicity bound to be higher than the degree of $\mathcal{T}$. Using Lemmas 4.10 and 4.11, we can reduce to the problem of evaluating bounded fan-in arithmetic circuits without subtraction gates, a problem that can be solved in #NC^1. The resulting circuit family produces a number representation of the histogram. □

PROOF OF THEOREM 1.4. For the proof we just need the fact that testing whether a function from #NC^1 outputs a value greater than 0 is an NC^1-computable property. □

An alternative proof of Theorem 1.4 that is based on the approach of Buss et al. (1992) works as follows: First make the decomposition binary. Then transform the input structure into a degree-bounded s-tree structure and consider an equivalent MSO-formula over s-trees. Finally, translate the formula into a classical tree automaton for degree-bounded labeled trees. Since simulating the acceptance behaviour of such tree automata is in NC^1 (Lohrey, 2001), the theorem follows.

5.3. Applications to Evaluating Automata

In this section, we show some examples of how to use the theorems for logarithmic-depth circuit classes to put decision and counting problems into NC^1 and $\#\mathrm{NC}^1$, respectively. Problems will be shown to lie in these classes by reductions to problems that are covered by Theorems 1.4 and 1.5; that means, MSO-definable decision and counting problems on tree-width-bounded structures. In order to apply the theorems, the reductions also need to compute term representations of width-bounded tree decompositions.

Evaluating Boolean and Arithmetic Sentences. One of the well studied problems in the area of logarithmic-depth circuits is the evaluation of Boolean and arithmetic sentences. Evaluating Boolean sentences that are given as a term like $((0 \wedge 1) \vee (1 \vee 1))$ can be done using pebbling-based evaluation strategies that recursively split the input into sentences of almost the same size.

FACT 5.11 (Buss (1987); Buss et al. (1992)). *Evaluating Boolean sentences is complete for* DLOGTIME-*uniform* NC^1 *under* DLOGTIME-*uniform* AC^0*-many-one-reductions.*

Theorem 1.4 provides a different route to prove the containedness in NC^1 by using an MSO-formula that existentially guesses truth values for all operations in the sentence and locally checks whether the value of each operation is consistent with its child values. Theorem 1.5, in turn, can be used to count the number of *proof trees* of Boolean sentences. A proof tree T of a Boolean sentence S is a subsentence that (a) contains the output gate of S, (b) for each $\vee$-operation in T contains exactly one of its children from S, (c) for each $\wedge$-operation in T contains all its children from S, and (d) evaluates to *true.* Since proof trees are MSO-definable, we can use Theorem 1.5 to count them. The evaluation of arithmetic sentences whose inputs are 0 and 1 is closely related to the problem of counting proof trees: If we replace, in an arithmetic sentence like $((0 \cdot 1)+(1+1))$, each $+$ by an $\vee$ and each $\cdot$ by an $\wedge$, then the number of proof trees of the resulting Boolean formula equals the value of the arithmetic sentence (Caussinus et al., 1998). Thus, by replacing operands in this way, Theorem 1.5 provides an alternative MSO- and tree-decomposition-based way to reprove the upper bound from the following fact.

FACT 5.12 (Krebs et al. (2010)). *Evaluating arithmetic sentences with inputs 0 and 1 is complete for* DLOGTIME-*uniform* $\#\mathrm{NC}^1$ *under* DLOGTIME-*uniform* AC^0*-many-one-reductions.*

Evaluating Visible Pushdown Automata. Buss (1987) extended his NC^1-approach for the evaluation of Boolean sentences to also cover the membership problem for parenthesis languages. This approach, in turn, was later generalized to cover context-free languages that are accepted by visible pushdown automata.

DEFINITION 5.13 (Visible Pushdown Automaton). A *visible pushdown automaton* (VPA) is a pushdown automaton $A = (\Sigma, \Gamma, \bot, Q, q_0, Q_a, \Delta)$ with *input alphabet* Σ, *stack alphabet* Γ that contains a distinguished *empty stack symbol* $\bot \in \Gamma$, *state set* Q with a distinguished *initial state* $q_0 \in Q$, a set $Q_a \subseteq Q$ of *accepting states*, and *transition relation* $\Delta \subseteq \Sigma \times \Gamma \times Q \times \Gamma^* \times Q$ that describes how to observe the current input symbol, the topmost stack symbol and the current state, and replace the topmost stack symbol by a string and the current state by a new state. In case of VPAs, the transition relation satisfies the following properties: Σ can be partitioned into three sets Σ_{PUSH}, Σ_{POP}, and $\Sigma_{\text{NO-CHANGE}}$, such that (a) when reading a symbol from Σ_{POP}, the automaton pushes exactly one symbol on the stack, (b) when reading a symbol from Σ_{POP}, it pops the topmost symbol from the stack or leaves an empty stack unchanged, and (c) when reading a symbol from $\Sigma_{\text{NO-CHANGE}}$, it does not alter the stack.

Given some input string, the height of the stack at all positions of the string (a) is the same for any nondeterministic computation and (b) can be computed by observing the type of the input symbols without simulating the automaton explicitly.

FACT 5.14 (Dymond (1988)). *Let A be a* VPA. *The acceptance behaviour of A can be simulated in* NC^1.

Beside deciding whether a string is accepted by a fixed VPA, recently the problem of counting the number of accepting computation paths of nondeterministic VPAs was studied in the context of logarithmic-depth circuits.

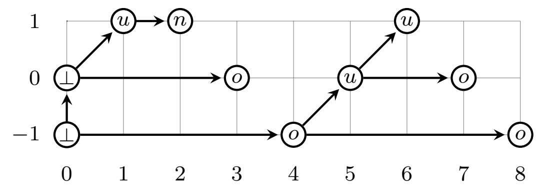
FACT 5.15 (Krebs et al. (2010)). *Let A be a* VPA. *Counting the number of accepting computations of A can be done in* $\#\mathrm{NC}^1$.

We will show how Theorems 1.4 and 1.5 can be used to reprove the results of Krebs et al. and Dymond. For this, consider any fixed VPA $A = (\Sigma, \Gamma, \bot, Q, q_0, Q_a, \Delta)$. We use a FTC^0-computation to translate input strings for A into a structure $\mathcal{S}$ of tree width 1 that represents the input string and a skeleton of the stack at the input positions. In order to apply Theorems 1.4 and 1.5, the reduction also computes a width-1 tree decomposition in ancestor representation for the structure. The vocabulary of the structure and the MSO-formula that describes accepting computations will only depend on the fixed automaton A.

Let $x = x_1 \dots x_n \in \Sigma^*$ be an input string. We start to define the *height function* $h(x,i) := |\{j \le i \mid x_j \in \Sigma_{\text{PUSH}}\}| - |\{j \le i \mid x_j \in \Sigma_{\text{POP}}\}|$. Note that the height function does not always equal the height of A's stack since A may pop on empty stacks during its computation.

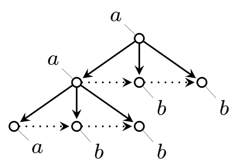
The structure $\mathcal{S}$ we construct is a tree whose nodes are labeled by monadic predicates; it has the vocabulary $\tau = \{E^2\} \cup \{P^1_\sigma \mid \sigma \in \Sigma\} \cup \{P^1_\bot\}$. Let min-height $:= \min\{0, h(x,1), \dots, h(x,n)\}$. The universe of S consists of the tuples $(i, h(x,i))$ for $i \in \{1, \dots, n\}$, and $(0, h)$ for $h \in \{\text{min-height}, \dots, 0\}$. The monadic relations are build as follows: For each position $i \in \{1, \dots, n\}$, we put the node $(i, h(x,i))$ into $P^{\mathcal{S}}_{x_i}$. For each $h \in \{\text{min-height}, \dots, 0\}$, we put the node $(0,h)$ into $P^{\mathcal{S}}_\bot$. The edge relation $E^{\mathcal{S}}$ is build as follows: For each position $i \in \{1, \dots, n\}$ an edge is added in dependence of which part of the alphabet x_i belongs to. If $x_i \in \Sigma_{\text{PUSH}}$, we insert an edge from $(i-1, h(x,i)-1)$ to $(i, h(x,i))$. If $x_i \in \Sigma_{\text{POP}}$, let $i' < i$ be the largest index such that $(i', h(x,i))$ is a node of $\mathcal{S}$. We insert an edge from $(i', h(x,i))$ to $(i, h(x,i))$. If $x_i \in \Sigma_{\text{NO-CHANGE}}$, we insert an edge from $(i-1, h(x,i))$ to $(i, h(x,i))$. In each case the node from which the edge starts from exists. For each $h \in \{\text{min-height}, \dots, -1\}$, we insert an edge from $(0,h)$ to $(0,h+1)$. Using FTC^0-computable arithmetic operations, the height function h and the structure $\mathcal{S}$ that is defined in terms of the string x and the height function h can be computed in FTC^0.

As an example for the construction consider a VPA with alphabet $\Sigma = \{u, o, n\}$ and partition $\Sigma_{\text{PUSH}} = \{u\}$, $\Sigma_{\text{POP}} = \{o\}$, and $\Sigma_{\text{NO-CHANGE}} = \{n\}$. For the input string $x = unoouuoo$, the reduction computes the structure that is shown below, where the indices of the nodes are given by their coordinates on the grid. The information whether a node is in some unary predicate (each element is in exactly one of the $|\Sigma| + 1$ unary predicates) is depicted by labeling the node with the corresponding symbol.



Since the edges of the structure only go from left to right or upwards, the structure does not contain a directed cycle. Moreover, since every node has exactly one ingoing edge, the directed graph underlying the structure is a directed tree with root $(0, \text{min-height})$. It is straightforward to define a tree decomposition of width 1 whose bags are the edges from the structure. An ancestor representation for this decomposition can be computed in FTC^0 by using the fact there is a path from a node (i, h) to a node (i', h') in $\mathcal{S}$ exactly if (a) $i = i' = 0$ and $h < h'$, or (b) $i < i'$, $h \leq h'$, and for all i'' with $i < i'' < i'$ we have $h(x, i'') \geq h$.

To define accepting computation paths we use an MSO-formula φ that consists of two collections of free set variables. The first collection is made up by monadic *state labeling* predicates $Q_1, \ldots, Q_{|Q|}$ used to define a partition of all nodes that are labeled with symbols from Σ and the node $(0, 0)$. A partition corresponds to guessing the states of the automaton at the input positions. The second collection is made up by monadic *stack content* predicates $P_1, \ldots, P_{|\Gamma \setminus \{\perp\}|}$ used to guess a single stack symbol for every PUSH node (that means, nodes that are labeled with unary predicates for symbols from Σ_{PUSH}). The MSO-formula evaluates to true if the sets assigned to the free variables satisfy these conditions and describe a valid and accepting computation. For that, $(0, 0)$ must be labeled by the initial state and the last node must be labeled by an accepting state. Moreover, a state that is assigned to a PUSH node is verified with respect to its labeling symbol from the input alphabet and the state at its predecessor; states at NO-CHANGE node are evaluated in the same way. For a state at a POP node n we observe the stack symbol that labels its sibling node s (the node that is reached by going a single step backwards to its direct predecessor and then following the edge to the other child of the predecessor) and the state that labels the node r that one reaches from s by only going along edges to NO-CHANGE and POP nodes. For the last verification step, the MSO-formula existentially guesses a path through the structure. Applying Theorem 1.5 proves that counting the number of accepting paths of VPAs lies in $\#\text{NC}^1$. Extending φ to a formula that existentially guesses sets for its free variables and applying Theorem 1.4 proves that membership testing for VPAs is in NC^1.



Evaluating Automata for Unranked Ordered Trees. At the beginning of Chapter 2, we discussed tree automata for different kinds of trees. An important automaton notion we discussed, but could not use in the context of constant-depth circuits, works on labeled trees whose nodes have an unbounded number of children that are ordered like shown on the right. For every node, such an *automaton for unranked ordered trees* determines its state based on the label of the node and the outcome of the computation of a finite automaton that works on the sequence of states assigned to the children of the node (Brüggemann-Klein et al., 2001). Gottlob et al. (2005) showed that simulating the acceptance behaviour of a fixed automaton is L-complete if the tree is given as a pointer structure like shown on the right, and NC^1-complete if it is given as a term $[\ a\ [\ a\ [\ a\ ]\ [\ b\ ]\ [\ b\ ]\ ]\ [\ b\ ]\ [\ b\ ]\ ]$. Note that the term representation of ordered trees is unique. It represents the hierarchical information explicitly using bracket symbols and the ordering on siblings implicitly using the order of symbols in the string.

FACT 5.16 (Gottlob et al. (2005)). *Let A be an automaton for unranked ordered trees. The acceptance behaviour of A on trees that are given as terms can be simulated in* NC^1.

This NC^1 upper bound can be reproved using tree decompositions and Theorem 1.4: On input of a term representation of an unranked ordered tree, build a width-2 tree decomposition whose underlying tree is build from the input tree as follows: (1) Use all (solid) edges that lead from a parent node to its left-most child, and (2) use all (dotted) edges between siblings. For every node, put the node itself, its parent node (if it exists), and its right sibling (if it exists) into its bag. A term representation of the tree underlying the decomposition can be constructed in FTC^0 by moving the closing bracket of each node directly in front of the closing bracket of its parent node. Finally, we use an MSO-formula that guesses and checks computations of the automaton; both the horizontal transition relation computed by the finite automaton on the ordered children and the vertical transition relation. Like in the case of counting the number of accepting computations of visible pushdown automata, this also gives the following:

THEOREM 5.17. *Let A be an automaton for unranked ordered trees. Counting the number of accepting computation paths of A on trees that are given as terms can be done in* $\#\mathrm{NC}^1$.

CHAPTER 6

Bounded Tree Width and Logarithmic Space

In the previous chapter we showed that number representations of solution histograms can be computed in DLOGTIME-uniform $\#\mathrm{NC}^1$ for input structures that are accompanied by tree decompositions of bounded width whose underlying trees are given as terms. Our goal in the present chapter is to develop a similar result for logspace and compute histograms for input structures of bounded tree width without given tree decompositions (hence, proving Theorem 1.2 from the introduction). We will invest our main effort in restoring the comfortable situation from the previous chapter. That means, we will show how to compute width-bounded tree decompositions in logspace. After computing decompositions and making them available in term representation, we can plug in the results from the previous chapter to compute number representations of solution histograms using DLOGTIME-uniform $\#\mathrm{NC}^1$-circuit families. Since logarithmic-space-bounded Turing machines can simulate such circuits in the sense that they compute the binary string encoding of the number computed by the arithmetic circuit (see Section 4.1 for details on how the considered complexity classes are related), Theorem 1.2 follows directly once we are able to compute tree decompositions of any bounded width in logspace. Thus, most of the content of this chapter is devoted to this task.

The research on the difficulty of computing width-bounded tree decompositions originally focused on providing fast sequential and parallel algorithms: Bodlaender (1996) presented a linear-time sequential algorithm for constructing tree decompositions of any constant width w. Similarly, the time used by parallel PRAM algorithms was reduced to $O(\log n)$ (Bodlaender and Hagerup, 1998). Concerning the computational complexity of computing width-w tree decompositions, Bodlaender (1989) showed how to do this using NC-circuits, which was later improved by Gottlob et al. (2002) to functional LOGCFL. For the special case of computing width-2 tree decompositions, a logspace algorithm can be deduced from the work of Jakoby et al. (2006) using the result of Reingold (2008).

The linear time bound of Bodlaender (1996) on constructing tree decompositions of any constant width w is typically proved in two steps: First, a linear-time algorithm is presented that, on input of a graph of tree width at most w, computes a tree decomposition of *approximate* width at most $O(w)$. Second, another linear-time algorithm is used to turn the decomposition into one that has *exact* width w. The proofs from the present chapter follow the same plan, but compute all steps using logarithmic space instead of linear time: We first show how to compute tree decompositions of approximate width in logspace. Constructing such decompositions and a term representation of their underlying trees is enough to apply the theorems for logarithmic-depth circuits from the previous chapter and prove Theorem 1.2, constructing string-encoded solution histograms in logspace. This implies Theorem 1.1 as a special case. Using Theorem 1.1 in conjunction with approximate tree decompositions will prove Theorem 1.3 on constructing tree decompositions of exact width.

The construction of tree decompositions of both approximate and exact width rests on the development of the notion of descriptor decompositions, which are vertex-labeled acyclic directed graphs that contain tree decompositions as subgraphs. Using this notion, we face two technical problems: (1) how to compute tree decompositions from descriptor decompositions in logspace, and (2) how to construct descriptor decompositions in logspace. We will only describe how to construct tree decompositions for undirected connected graphs instead of arbitrary relational structures in the present chapter. This is no restriction since structures and their Gaifman graphs have the same tree decompositions (Fact 4.2) and different components of a graph can be decomposed independently.

The chapter is organized as follows: In Section 6.1, we discuss the basic techniques available to design logarithmic-space-bounded DTMs. In Section 6.2 the notion of descriptor decompositions is introduced and it is shown how to construct tree decompositions from given descriptor decompositions. Section 6.3 contains the proofs of how to construct descriptor decompositions of approximate and exact width in logspace. Section 6.4 shows how to apply the Theorems 1.1 and 1.2 to solve problems related to finding paths and matchings in graphs.

6.1. Review of Logarithmic-Space-Bounded Computations

Formally, the class FL, called *functional logspace*, contains all functions $f\colon \{0,1\}^* \to \{0,1\}^*$ that are computable by a deterministic Turing machine (DTM) M whose working space is bounded by a logarithm of the input length. That means, there exists a constant $c \in \mathbb{N}$, such that M outputs $f(s)$ on input $s \in \{0,1\}^*$ using at most $c \cdot \log |s|$ cells of its work tapes. The class L, called *logspace*, contains all languages $L \subseteq \{0,1\}^*$ whose characteristic functions $\chi_L\colon \{0,1\}^* \to \{0,1\}$, with $\chi_L(s) = 1$ if $s \in L$ and $\chi_L(s) = 0$ if $s \notin L$, are in FL (see the textbook of Papadimitriou (1994) for more details on Turing machines and resource bounds).

In Section 4.1, we discussed that all functions from DLOGTIME-uniform #NC^1 are also in FL if the task is to compute binary string representations of output numbers. To prove conditional lower bounds against logarithmic-depth circuit families, L-hardness proofs are commonly used. They are based on many-one reductions computable in DLOGTIME-uniform FNC^1 (Cook and McKenzie, 1987) or FAC^0 (Immerman, 1999); we use the later reducibility notion in the present chapter for proving L-hardness.

Restricting the work space of Turing machines to be logarithmic results in a clean and elegant notion for studying the space complexity of problems that are already known to be polynomial-time solvable: logspace computations always run in polynomial time, while more space needs more than polynomial time a priori, and less space does not allow to store pointers to input entries, like pointers that address vertices in graphs.

While logspace computations seem to be very restrictive at first sight, one can still use basic design patterns to develop algorithms that can be implemented in logarithmic space: It is possible to compose any constant number of logspace-computable functions since FL is closed under composition (Stockmeyer and Meyer, 1973). Moreover, one can always call logspace subroutines since FL is closed under logspace Turing reductions that query characteristic functions of languages from L (Ladner and Lynch, 1976). These techniques already suffice to prove L upper bounds for the reachability problem in undirected forests or directed graphs whose underlying undirected graphs are forests (Cook and McKenzie, 1987). Much more involved problems can be solved in logspace using the algorithm of Reingold (2008) for solving the reachability problem in undirected graphs of any kind. Nowadays this can be seen as a basic algorithmic technique for designing logspace DTMs: Query *Reingold's Algorithm* to find paths in undirected graphs.

6.2. From Descriptor to Tree Decompositions

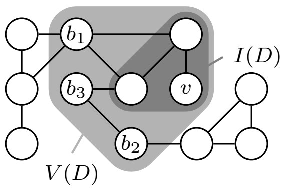
In the algorithms for computing tree decompositions, the decomposition tree will be a subgraph of a larger graph in which it is hidden. These graphs are called descriptor decompositions. In the following, we will first define the notion of a descriptor decomposition. Then we show how to compute tree decompositions from descriptor decompositions in logspace. Descriptor decompositions are build from the following descriptors; they are similar to the ones used in the Chapters 3 and 4, but use constant size sets to store bag vertices instead of sequences of bag vertices.

Definition 6.1 (Descriptors for Descriptor Decompositions). Let $G = (V, E)$ be an undirected connected graph. A *descriptor D in G* is either

(1) a bag $B \subseteq V$ and called *simple*, or
(2) a pair (B, v) consisting of a bag $B \subseteq V$ and a *component selector $v \in V \setminus B$*.

We write $B(D)$ for the bag of D.

We say that D *describes* the following graph $G(D)$: If D is simple, $G(D) := G[B]$. Otherwise let $G(D) := G\big[V(C) \cup B\big]$ where C is the component of $G[V \setminus B]$ that contains v. We write $V(D)$ for the vertex set of the graph $G(D)$. The *interior* $I(D)$ of $G(D)$ is $V(D) \setminus B(D)$. Let D_G denote the descriptor $(\emptyset, v_0)$ where v_0 is a chosen vertex of G. Note that $G(D_G) = G$ since we assumed G to be connected. An example of a graph G, a descriptor $D = (\{b_1, b_2, b_3\}, v)$, and the sets $V(D)$ and $I(D)$ is shown right. The graph $G(D)$ is the induced subgraph $G[V(D)]$.



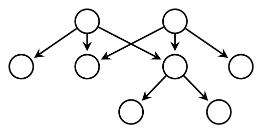
Definition 6.2 (Descriptor Decomposition). Let G be an undirected connected graph. A *descriptor decomposition M* of G is a directed graph M whose vertices are descriptors in G, such that exactly one of them is D_G, and for every node with descriptor D of M and children with descriptors $D_1, \dots, D_m$ the following holds:

(1) For each child descriptor D_i, we have $V(D_i) \subseteq V(D)$ and $I(D_i) \subseteq I(D)$ and at least one inclusion is proper.
(2) For each child descriptor D_i, the set $V(D_i)$ contains at least one vertex from $I(D)$.

(3) For each child descriptor D_i, the set $I(D_i)$ is disjoint from all $V(D_j)$ for $j \neq i$.
(4) Each edge in $G(D)$ that is not between two vertices in $B(D)$ must be present in some $G(D_i)$.

For a descriptor D with children $D_1, \ldots, D_m$, we let D^{interact} be the simple descriptor whose bag contains all vertices present in at least two of the sets in $B(D)$, $B(D_1)$, …, $B(D_m)$. The *width* of a descriptor decomposition is the maximum size over all bags $B(D)$ and $B(D^{\text{interact}})$ for descriptors D in M minus 1.

With Lemma 6.3 we prove that the graphs underlying descriptor decompositions are mangroves. A *mangrove* is a DAG as shown on the right in which there is at most one path between any two vertices. In a mangrove, the subgraph T_r induced on all vertices reachable from a given vertex r is a tree rooted at r. Thus, like a forest, mangroves are unions of trees, but the union is not necessarily disjoint and the trees can merge in complex ways—hence the name mangrove.

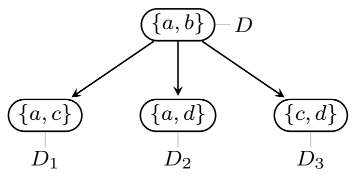


Lemma 6.3 (Descriptor Decompositions are Mangroves). *Let M be a descriptor decomposition of some graph G. The graph underlying M is a mangrove.*

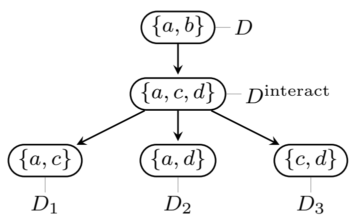
Proof. By property (1) of descriptor decompositions, M is a DAG. To prove that there is at most one path between any two vertices of M, suppose there is a node with descriptor D_s and two different child descriptors D_1 and D_2 such that a node with descriptor D_t is reachable from both. By property (2) of descriptor decompositions, $G(D_t)$ contains at least one $v \in I(D_1)$. By property (3), $G(D_2)$ does *not* contain v, but $G(D_t)$ must be a subgraph of $G(D_2)$ by property (1), which is a contradiction. □

We next prove that given a descriptor decomposition M of a graph G, the graph $M[W]$ where W is the set of all nodes reachable from D_G in M nearly forms a tree decomposition of G: We only need to add an internal vertex between each node and its children whose bag contains all interactions between the children of the node. Formally, we define a graph $T(M)$ as follows: For each node D reachable from D_G in M, it contains two vertices D and D^{interact}. If $D_1, \ldots, D_m$ are the children of D in M, then there are edges from D to D^{interact} and from D^{interact} to each D_i. Label D^{normal} with the bag $B(D)$ and D^{interact} with $B(D^{\text{interact}})$. On the right an example of a descriptor D, its children D_1, D_2, and D_3 in M, and their bags is shown; as well as the resulting nodes and bags in $T(M)$.

Descriptors and bags in M



Nodes and bags in $T(M)$

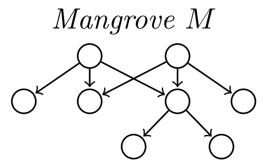


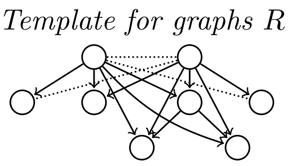
Lemma 6.4 (From Descriptor to Tree Decompositions). *If M is a width-w descriptor decomposition of G, then $T(M)$ is a width-w tree decomposition of G.*

PROOF. We claim that $T(M)$ is a tree decomposition of G. By Lemma 6.3, we know that M is a mangrove, $M[W]$ where W is the set of all vertices reachable from D_G in M is a tree and, thus, also $T(M)$. The width of the tree decomposition follows directly from the definition of the width of descriptor decompositions. We need to check the two properties of a tree decomposition from Definition 4.1.

Proof of the connectedness condition: Consider the set of all nodes of $T(M)$ whose bags contain some vertex v. It suffices to prove that there is a unique node of $T(M)$ whose bag contains v, but whose parent node's bag does not contain v. To see this, consider the set P of all nodes D of $M[W]$ for which $v \in I(D)$. This set includes at least the root D_G. By properties (1) and (3) of the descriptor decomposition M, the set P forms a path in M, ending at a uniquely specified node D. Since v is not an isolated vertex (G is connected), for at least one child D_i of D the graph $G(D_i)$ must contain v. Since v is no longer an interior vertex of D_i, it must be in the bag $B(D_i)$. Now, if there is exactly one child D_i of D whose bag contains v, then $B(D_i)$ will be the only bag that contains v but whose parent's bag does not. Otherwise, if there are several children whose bags contain v, then $B(D^{\text{interact}})$ will be the only bag containing v whose parent's bag does not.

Proof of the cover condition: Consider any edge e of G. Since e is contained in $G(D_G) = G$, there must be some node D of $M[W]$ such that e is contained in $G(D)$, but not in $G(D_i)$ for any of its children D_i (at the latest, this is the case for some leaf of $M[W]$). Then by property (4) of descriptor decompositions, the edge e must be between two vertices in $B(D)$. □





Computing $T(M)$ from M can be done by solving the reachability problem in mangrove graphs. The best upper space bound on the reachability problem for mangroves is $O(\log^2 n/\log\log n)$ from Allender and Lange (1998), which is far from logarithmic. Thus, our algorithm needs access to some kind of additional information. This information will be in the form of what we call transitive closures of related vertices. Let us say that two vertices a and b of a mangrove are *related* if they are both present in some T_r, which is the tree of all nodes reachable from r in M. We say that a graph R is *a transitive closure of the related vertices of M* if the following holds: Whenever a and b are related in M, then there is an edge from a to b in R if, and only, if there is a nonempty path from a to b in M. The example on the right shows a mangrove M at the top. All transitive closures R of M's related vertices can be obtained by arbitrarily adding edges in the lower template graph along the dotted lines, which connect exactly the unrelated vertices of M.

LEMMA 6.5. *There is a logspace* DTM *that, on input of a mangrove M and a transitive closure R of M's related vertices, outputs the transitive closure of M.*

PROOF. Let a mangrove M and a transitive closure R of M's related vertices be given as input as well as two vertices $s, t \in V(M) = V(R)$. We claim that the following algorithm, which clearly needs only logarithmic space, correctly decides whether there is a nonempty path from s to t in M:

```
1 $c \leftarrow s$
2 **while not** $(c,t) \in E(M)$ **do**
3     **if** there is exactly one $v \in V(M)$ with $(c,v) \in E(M) \wedge (v,t) \in E(R)$
4         **then** $c \leftarrow v$
5         **else reject**
6 **accept**
```

The current vertex stored in c is always reachable from s in M and, thus, if the algorithm accepts, there is a nonempty path from s to t in M. For the other direction suppose there is a path from s to t in M. Starting with $c = s$, consider each child v of c in M. All of these children are related to t via the common ancestor c. Thus, $(v,t) \in E(R)$ holds if, and only if, there is a path from v to t in M. This in turn holds for exactly one child v of c since M is a mangrove—namely for the child on the unique path from s to t. This means that the variable c will successively be set to the vertices on the path from s to t and the algorithm will accept. □

In light of the previous lemma, all we need to do to compute $T(M)$ from M is to compute a transitive closure of M's related vertices, which is done in the proof of the following lemma.

Lemma 6.6 (From Descriptor to Tree Decompositions in Logspace). *There is a logspace* DTM *that, on input of any graph G together with a descriptor decomposition M of G, outputs $T(M)$.*

Proof. In Lemma 6.3 we proved that M is a mangrove. We claim that the following graph R is a transitive closure of M's related vertices: Its vertex set is $V(M)$ and there is an edge from D to D' in R if, and only if, D and D' satisfy property 1 of Definition 6.2, that is, $V(D') \subseteq V(D)$ and $I(D') \subseteq I(D)$ and at least one of these two inclusions is proper. Then R is clearly a superset of the transitive closure of M. Second, if there are two disjoint paths leading from a vertex D_s to two vertices D_1 and D_2, then, as argued in Lemma 6.3, $G(D_1)$ and $G(D_2)$ each contains at least one vertex not contained in the other graph. Thus, there is no edge between them in R.

Observe that the graph R is logspace-computable: The algorithm of Reingold (2008) allows us to check in logarithmic space on input of G, D, and a vertex v whether $v \in G(D)$. This allows us to apply Lemma 6.5 to M and R in order to compute the set of vertices reachable from D_G in M in logarithmic space, yielding $T(M)$. □

6.3. Computing Tree Decompositions in Logarithmic Space

In the present section we show how to construct tree decompositions of approximate and exact width in logspace. We start to prove the construction of tree decompositions of approximate width and show how they can be used to prove Theorem 1.2. In light of the previous section, all we need to do to construct tree decompositions of a certain width is to construct descriptor decompositions of this width.

Computing Tree Decompositions of Approximate Width. Algorithms for constructing tree decompositions often employ a specific notion of separators, which are used to split a graph into smaller subgraphs for which tree decompositions can be computed recursively. When one wants to transfer this idea to logarithmic space, one faces the problem that both the recursion stack and the intermediate subgraphs are too large to store. We overcome these problems in two ways: First, instead of avoiding deep recursions, we construct descriptor decompositions from which tree decompositions can be singled out in logspace. Second, we pick a notion of separators that allows us to represent subgraphs in logarithmic space.

LEMMA 6.7. *For every $w \in \mathbb{N}$, there is a logspace* DTM *that, on input of any graph G, either*

(1) outputs a tree decomposition (T, B) for G whose width is at most $3w+3$, or
(2) outputs "no" and $\mathrm{tw}(G) > w$ holds in this case.

By Lemmas 6.4 and 6.6, in order to compute a tree decomposition of a graph, it suffices to compute a descriptor decomposition. We next show that such a descriptor decomposition can be obtained in logarithmic space. As remarked earlier, algorithms for computing tree decompositions internally use different kinds of separators. The ones we use are also known as balanced separators.

DEFINITION 6.8 (Separators). Let G be an undirected graph and let $U \subseteq V(G)$. A *separator $S \subseteq V(G)$ separates U in G* if each component of $G[V(G)\setminus S]$ contains at most $|U|/2$ vertices of U. An *s-separator* is a separator of size at most s.

The following folklore separator property (proved, for example, in the book of Flum and Grohe (2006)) of tree-width-bounded graphs shows that the existence of a separator of size $w+1$ is necessary for graphs of tree width at most w.

FACT 6.9 (Separator Property). *Let G be an undirected graph with $\mathrm{tw}(G) \leq w$ and let $U \subseteq V(G)$. Then there exists a $(w+1)$-separator $S \subseteq V(G)$ for U in G.*

DEFINITION 6.10 (Child Descriptors). Let G be an undirected connected graph with $\mathrm{tw}(G) \leq w$. We define the *child descriptors* of a non-simple descriptor D with $B(D) \leq 2w+2$ in G as follows: Choose a set $W \subseteq V(G)$ with $B(D) \subseteq W$ that has size $2w+3$ if $|V(G)| \geq 2w+3$, and $W = V(G)$, otherwise. Then choose a $(w+1)$-separator S of W in the graph $G(D)$. Let C_1 to C_m be the components of $G[I(D)\setminus S] = G\big[V(D)\setminus(S \cup B(D))\big]$. For each $i \in \{1, \dots, m\}$ choose a vertex $v_i \in C_i$ and let B_i be the set of all vertices in $W \cup S$ that are adjacent in $G(D)$ to a vertex from C_i. Then the descriptors (B_i, v_i) are the child descriptors of D and, unless $S \subseteq W$, additionally the simple descriptor $D_0 = W \cup S$.

LEMMA 6.11 (Size Lemma). *Let D' be a child descriptor of a non-simple descriptor with $|B(D)| \leq 2w+2$. Then*

(1) If D' is simple, then $|B(D')| \leq 3w+4$.
(2) If D' is non-simple, then $|B(D)| \leq 2w+2$

PROOF. The claim for simple D' follows from the fact that its bag is of the form $W \cup S$ with $|W| \leq 2w+3$ and $|S| \leq w+1$. For the case that D'

is non-simple, each B_i can contain at most $|W|/2$ vertices from W and, hence, must have size at most $|W|/2 + |S| \leq w + 1 + w + 1$. $\square$

We are now ready to define the desired descriptor decomposition and to show that it is logspace-computable.

Definition 6.12. Let G be an undirected, connected graph with $\mathrm{tw}(G) \leq w$. Let $M(G)$ be the graph whose vertex set contains all descriptors D in G with $|B(D)| \leq 2w + 2$ for non-simple D and $|B(D)| \leq 3w + 4$ for simple D and where there are edges from each non-simple descriptor exactly to its child descriptors.

Lemma 6.13. *The graph $M(G)$ is a descriptor decomposition of G of width $3w + 3$.*

Proof. By the size lemma, M is well-defined, that is, the child descriptors do, indeed, have the maximum sizes $2w + 2$ or $3w + 4$. Next, M contains D_G. Concerning the four properties of a descriptor decomposition, we argue as follows. Consider the child descriptors D_1, …, D_m, and possibly D_0 of a descriptor D. First, by construction each $G(D_i)$ is clearly a subset of $G(D)$. The set $I(D_0)$ is empty and the other interiors $I(D_i)$ are exactly the components C_i and, thus, subsets of $I(D)$. The construction also ensures that $V(D_i) \subsetneq V(D)$ for $i \in \{1, \dots, m\}$ and, if D_0 is present, $\emptyset = I(D_0) \subsetneq I(D)$. Second, each $G(D_i)$ contains the vertex v_i, which is from the interior of D, and $V(D_0)$ also contains an interior vertex. Third, no C_i is connected to a vertex in another component. Hence, the interior vertices of the D_i are not part of any other $V(D_j)$. Fourth, every edge in $G(D)$ that is not between two vertices from $B(D)$ is either inside a component C_i and thus included in $G(D_i)$; or it is between a vertex in a component C_i and a vertex in $B(D) \cup S$ and thus, again, included in $G(D_i)$; or it is between a vertex in $S \setminus B(D)$ and a vertex in $B(D)$ and thus included in $G(D_0)$.

For the width bound, we need to show $|B(D)| \leq 3w+4$ and $|B(D^{\text{interact}})| \leq 3w + 4$ for each descriptor D in M. This property holds for each $B(D)$ by definition. Moreover, the bag of each interaction descriptor is a subset of some $W \cup S$ with $|W| \leq 2w + 3$ and $|S| \leq w + 1$. $\square$

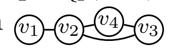
Proof of Lemma 6.7. The logspace DTM either outputs $M(G)$ or "no", and in this case $\mathrm{tw}(G) > w$ holds. Since the size of the vertex set of $M(G)$ is polynomially bounded, all we need to show is that a logspace DTM can compute the set of child descriptors of a descriptor D. For this, it finds the set W and the separator S by choosing some W and testing for each possible set S of size $w + 1$ whether it separates the correct set in $G(D)$. For this, the machine uses the algorithm of Reingold (2008) to determine the components into which S separates $G(D)$. The machine then picks one such S and then determines, again using Reingold's algorithm, the sets B_i and outputs the desired child descriptors. If, at some point, the machine does not find a separator S, it outputs "no" since the separator property is violated. $\square$

Proof of Theorem 1.2. On input of a structure $\mathcal{A}$ of tree width at most w, we first construct a tree decomposition (T, B) of width at most $3w + 3$ for its Gaifman graph $G(\mathcal{A})$ using Lemma 6.7. By Fact 4.2, (T, B) is also a tree decomposition for $\mathcal{A}$. Moreover, a term representation of T can be computed in logspace by traversing it. Since $\#\mathrm{NC}^1 \subseteq \mathrm{FL}$ in the sense that the bitstring representation of numbers that are computed by functions from $\#\mathrm{NC}^1$ can be computed in FL, we can plug in Theorem 1.5 to compute $\mathrm{str}(\mathrm{histogram}(\mathcal{A}, \varphi))$ in logspace. $\square$

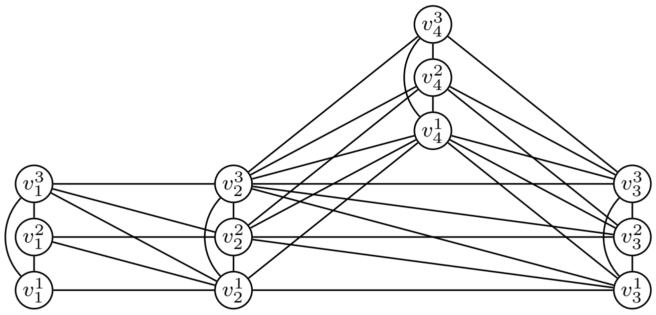
Computing Tree Decompositions of Exact Width. In the present section we prove Theorem 1.3 from the introduction. For its proof we first show that TREE-WIDTH-w, the language of all graphs whose tree width is at most w, is L-complete for each $w \in \mathbb{N}$.

THEOREM 6.14. *For every $w \in \mathbb{N}$, the language* TREE-WIDTH-w *is complete for* L *under* DLOGTIME*-uniform* AC^0*-many-one-reductions.*

PROOF. First, we need to show that TREE-WIDTH-$w \in$ L holds for all w. The work of Robertson and Seymour (2004) implies that for every w the graphs in TREE-WIDTH-w can be characterized by a finite set of forbidden minors. Furthermore, it is also well known (see, for example, (Arnborg et al., 1991)), that the question of whether a graph contains any fixed graph as a minor can be defined using an MSO-formula. This allows us to decide TREE-WIDTH-w as follows: On input of a graph G, use Lemma 6.7 to obtain an approximate tree decomposition T of G. If such a tree decomposition does not exist, the machine rejects; in this case we know from Lemma 6.7 that the tree width of G exceeds w. If the tree decomposition exists, we know that G's tree width is at most $3w + 3$ and we can apply Theorem 1.1 with an MSO-formula that defines the set of graph of tree width w.

Second, we need to show that for every $w \geq 1$ the problem TREE-WIDTH-w is hard for L. For a fixed $w \geq 1$, we reduce the L-complete problem ACYCLICITY for undirected graphs to TREE-WIDTH-w via the following FO-computation: On input of an undirected graph G with $V(G) = \{v_1, \dots, v_n\}$, build a new undirected graph G' as follows: For each vertex $v_i \in V(G)$ the set $V(G')$ contains w vertices $v_i^1, \dots, v_i^w$. In the edge set $E(G')$ they are connected so that they form a w-clique. Next, for each edge $(v_i, v_j) \in E(G)$ with $i < j$, the following edges are present in $E(G')$: For all $p \in \{1, \dots, w\}$ and $q \in \{p, \dots, w\}$ there is an edge between v_i^q and v_j^p. As an example, the graph 

is mapped to the following graph for $w = 3$:



We claim that G is acyclic if, and only if, G' has tree width w. To prove this, first assume that G is acyclic. Then each component of G' has tree width w. To see this, note that a tree decomposition of a component of G' can be obtained from G as follows: Attach the bag $\{v^1, \dots, v^w\}$ to each vertex v of G. Then, for each edge (v_i, v_j) of G with $i < j$, replace the edge by a path of length w and attach the following bags to the new vertices: $\{v_i^1, \dots, v_i^w, v_j^1\}$,

$\{v_i^2, \ldots, v_i^w, v_j^1, v_j^2\}$, $\ldots$, $\{v_i^w, v_j^1, \ldots, v_j^w\}$. The resulting graph is clearly still acyclic, its bags cover all edges of G', and the subgraphs of all vertices whose bags contain a given vertex are connected.

Second assume that G contains a cycle. We show that G' contains a $(w+2)$-clique as a minor and, since the tree width of a graph does not increase by taking minors (Diestel, 2005) and for every tree decomposition each clique is completely contained in at least one bag, this implies $\mathrm{tw}(G') > w$. Let $(v_1, \ldots, v_r, v_1)$ be a cycle of G. In the subgraph $G'[\{v_1^1, \ldots, v_1^w\} \cup \cdots \cup \{v_r^1, \ldots, v_r^w\}]$, for each $j \in \{2, \ldots, r\}$ merge the clique $\{v_j^1, \ldots, v_j^w\}$ to a single node v'_j. In the resulting minor both nodes v'_2 and v'_r have edges to all vertices of the clique $\{v_1^1, \ldots, v_1^w\}$. By merging the path $(v'_2, \ldots, v'_r)$ into the edge (v'_2, v'_r), we obtain the $(w+2)$-clique $\{v_1^1, \ldots, v_1^w, v'_2, v'_r\}$. □

Proof of Theorem 1.3. First test whether the tree width of $G = (V, E)$ is at most w using Theorem 6.14. If the tree width of G exceeds w, we output "no". Otherwise, we proceed to construct a tree decomposition of width w for G in logarithmic space. Recall that, by the Lemmas 6.4 and 6.6, on input of a graph G together with a width-w descriptor decomposition M of G, we can compute a width-w tree decomposition of G in logarithmic space. Thus, in the following it suffices to explain how a descriptor decomposition M of G of width w can be obtained in logarithmic space. Clearly, M has to be different from the one constructed for Lemma 6.7 since we can no longer allow bags larger than $w + 1$. As we define the new descriptor decomposition M below, we also explain how M can be constructed via some logspace DTM.

We start with some preprocessing and trivial cases. If G is not connected, we decompose the components individually. In case $|V| \leq w + 1$, we just output a single bag containing all of V and are done. So, in the following, we may assume that G is connected, $\mathrm{tw}(G) \leq w$, and $|V| > w + 1$.

The following notations will be useful: Let us write K_B for the clique with vertex set B. Given two graphs $G = (V, E)$ and $G' = (V', E')$, let $G \cup G' = (V \cup V', E \cup E')$.

The vertex set $V(M)$ of descriptors contains D_G and all descriptors D with $|B(D)| = w + 1$ such that there is a tree decomposition T of $G(D)$ in which $B(D)$ is attached to some bag of T. Given a descriptor D, we can test whether it is an element of $V(M)$ in logarithmic space as follows: We apply Lemma 6.14 to the graph $G(D) \cup K_{B(D)}$ and include D if this graph has tree width at most w. Observe that, indeed, if there is a tree decomposition T of $G(D)$ in which $B(D)$ is attached to some bag of T, then this tree decomposition is also a tree decomposition of the same width of $G(D) \cup K_{B(D)}$. The other way round, in every tree decomposition of $G(D) \cup K_{B(D)}$ there must be a node whose bag contains the clique $K_{B(D)}$ (this is a fundamental property of tree decompositions proved, for example, by Flum and Grohe (2006)).

To define the edge set $E(M)$, we explain, in the same spirit as in Definition 6.10, which descriptors are the *child descriptors* of a given descriptor $D \in V(M)$. If D is simple, it has no child descriptors. Otherwise, we search for a bag $B' \subseteq V(D)$ with the following properties.

(1) $|B'| = w + 1$ and $B' \cap I(D) \neq \emptyset$.
(2) There is no edge between a vertex in $B(D) \setminus B'$ and a vertex in $I(D)$.

(3) Let $C_1, \ldots, C_m$ be the components of the graph $G[I(D) \setminus B']$. Then for each $i \in \{1, \ldots, m\}$ the graph $G[V(C_i) \cup B'] \cup K_{B'}$ must have tree width at most w.

Clearly, if B' with the above properties exists, we can find it in logarithmic space by iterating over all possible B' and each time invoking Lemma 6.14 on $G[V(C_i) \cup B'] \cup K_{B'}$. We choose one such B' and for every component C_i of $G[I(D) \setminus B']$ we choose a vertex $v_i \in C_i$ and let all (B', v_i) be child descriptors of D; additionally, the simple descriptor B' is a child descriptor of D.

We first show that a set B' with the above properties always exists: Consider a tree decomposition T of $G(D)$ of width exactly w in which there is a node whose bag is B. Without loss of generality we can assume that T has the following properties: For every pair $(n, n') \in E(T)$ we have $B(n) \nsubseteq B(n')$ and $B(n') \nsubseteq B(n)$, and every bag has size $w + 1$. Together with the fact that $I(D)$ is connected, this implies that B has exactly one neighboring bag B'; otherwise, there are two vertices v and v' from different neighboring bags that are connected in $G(D)$, but not in $I(D)$. Using the decomposition T one can see that B' satisfies all the above properties.

It remains to argue that the resulting graph M is a descriptor decomposition of width w. To see that M is a descriptor decomposition, first note that D_G is an element of $V(M)$. Concerning the four properties of a descriptor decomposition, properties 1 to 3 follow for exactly the same reasons as in Lemma 6.13. For property 4, we can account for all edges as follows: Edges inside the C_i and between C_i and B' are covered by the graphs $G(D_i)$. Edges inside B' are covered by the simple descriptor B'. Edges inside B need not be covered. Edges between $B \setminus B'$ and $I(D)$ do not exist. To prove that the width of M is $w+1$, first note that all bags $B(D)$ attached have size at most $w + 1$ by construction. Second, note that the bag of an interaction descriptor D^{interact} is a subset of B'. Hence, the size of interaction bags is at most $w + 1$. $\square$

In conjunction with Lemma 5.3 from the previous chapter, Theorem 1.3 implies that balanced tree decompositions can be computed in logarithmic space:

LEMMA 6.15. *For every $w \in \mathbb{N}$, there is a logspace* DTM *that, on input of a graph G of with tree width at most w, outputs a width-$(4w+3)$ tree decomposition (T, B) of G where T is a binary and balanced tree.*

6.4. Applications to Finding Paths and Matchings

We distinguish two ways of applying the Theorems 1.1 and 1.2. The first way is to consider problems that are defined by MSO-formulæ on input structures of bounded tree width. The second way is to use the theorems as subroutines in algorithms that solve problems whose input structures have unbounded tree width.

Finding Paths and Matchings in Graphs of Bounded Tree Width. Two of the widely studied problems in the area of logarithmic space computations are the problems of testing whether there is a path from some start vertex some target vertex in a graph, the problem REACHABILITY, and the problem of detecting whether a graph has a perfect matching, the problem PERFECT-MATCHING.

While reachability for undirected graphs is known to be solvable in L (Reingold, 2008) and NL-complete for directed graphs (Jones, 1975), it is widely open to understand the complexity of reachability in restricted classes of directed graphs. Das et al. (2010) showed that, for any $k \in \mathbb{N}$, the directed reachability problem can be solved in L for orientations of k-trees, a result which is also known for tree-width-2 directed graphs (Jakoby et al., 2006). Applying Theorem 1.2 to an MSO-definition of paths on incidence structures of graphs, as developed in Section 3.4, gives the following result:

THEOREM 6.16 (Path Problems on Tree-Width-Bounded Graphs). *For every tree width bound $w \in \mathbb{N}$ and each of the following problems, there exists a logspace* DTM *that solves the problem for input graphs $\mathcal{G}$ with $\mathrm{tw}(\mathcal{G}) \leq w$, vertices $s, t \in V(\mathcal{G})$, and (for the case where it applies) a length bound $\ell \in \mathbb{N}$:*

- *Decide whether there is a directed path from s to t in $\mathcal{G}$.*
- *Count the number of directed paths from s to t in $\mathcal{G}$.*
- *Decide whether there is a directed path of length ℓ from s to t.*

Note that the above theorem can also be extended to other problems that are based on computing solution histograms, like the problem of counting the number of paths of a given length ℓ from s to t.

Testing whether a graph has a perfect matching can be done in polynomial time, but it is not known whether it is solvable in parallel using polylogarithmic-depth Boolean circuits. For some restricted classes of graphs, though, the complexity of PERFECT-MATCHING is known. For example, for k-trees it is known to be L-complete (Das et al., 2010). As for the case of solving the reachability problem, the theorems for logspace can be used to extend this result to any tree-width-bounded graph class by using the MSO-definition of matchings on incidence structures of graphs from Example 3.24.

THEOREM 6.17 (Matching Problems on Tree-Width-Bounded Graphs). *For every tree width bound $w \in \mathbb{N}$ and each of the following problems, there exists a logspace* DTM *that solves the problem for input graphs $\mathcal{G}$ with $\mathrm{tw}(\mathcal{G}) \leq w$, and (for the case where it applies) a solution size $m \in \mathbb{N}$:*

- *Decide whether $\mathcal{G}$ has a perfect matching.*
- *Count the number of perfect matchings of $\mathcal{G}$.*
- *Decide whether there is a matching in $\mathcal{G}$ that contains exactly m edges.*

Finding Cycles of Even Length in Undirected Graphs. The problem of whether an undirected graph $\mathcal{G}$ contains a *cycle of odd length* is well known to lie in logspace. The L upper bound can be proved by transforming an n-vertex input graph into a layered graph that results from copying the vertices of $\mathcal{G}$ into n layers, and connecting consecutive layers be edges like in the original graph. Then Reingold's logspace algorithm is used to test whether there exists a path from any vertex s in the first layer to a copy of it that lies in a layer of odd distance. There exists such a path exactly if there is an odd-length cycle that contains s in $\mathcal{G}$. When trying to use this approach for finding *cycles of even length* we run across the problem that an even-length path in the layered graph does not necessarily give us an even-length cycle in the original graph. Surprisingly, the logspace construction of tree decompositions allows us to solve this problem in logspace; even the following generalized version for any $m \in \mathbb{N}$:

$$\textsc{mod-}m\textsc{-cycle} := \{\, \mathrm{str}(\mathcal{G}) \mid \mathcal{G} \text{ is an undirected graph without loops, and contains a cycle whose length is a multiple of } m\}.$$

Theorem 6.18. *For any $m \in \mathbb{N}$,* MOD-m-CYCLE *is complete for* L *under* DLOGTIME-*uniform* AC^0*-many-one-reductions.*

Proof. The work of Thomassen (1988) implies that there exists a constant $w := w(m)$, such that every undirected graph without loops whose tree width exceeds w contains a cycle whose length is a multiple of m. Thus, to prove the theorem, we use a logspace DTM that first applies Theorem 1.3 with tree width bound w. If the tree width of the input graph exceeds w, we output "yes". If the tree width is at most w, we apply Theorem 1.2 with tree width bound w and an MSO-formula that defines edge sets of cycles on the incidence representation of graphs. If the corresponding histogram contains an entry that is greater than 0 at any index that is a multiple of m, we output "yes" and, no, otherwise.

Hardness for L follows from the fact that we can reduce the question of whether a graph contains a cycle (has tree width at least 2) to the question of whether a reduced graph in which we replace each edge by a length-m path has a cycle whose length is a multiple of m. This reduction is, clearly, FO-computable. □

CHAPTER 7

Conclusion

The thesis at hand resolved the computational complexity of MSO-definable problems on tree-decomposable structures. It was shown that these problems are complete for complexity classes defined via logarithmic-space-bounded Turing machines and circuit families of logarithmic and constant depth. Transferring the framework of MSO-based problem definitions to these classes resolved the complexity of a number of problems and yielded elegant proofs of known results using MSO-formulæ and tree decompositions. The developed techniques where either transferred from the area of time-efficient algorithms to the needs of space-bounded and circuit-based computations or developed newly.

7.1. Summary

The first set of results from the thesis spanned input structures of bounded tree width. It was shown that string-encoded solution histograms can be computed in logspace and that there are L-complete path and matching problems covered by this result. The main technique that was developed to prove this is the construction of tree decompositions of bounded width in logspace. For structures that are accompanied by tree decompositions of bounded width in term representations, the thesis presented refined results: number-encoded histograms can be computed in $\#\mathrm{NC}^1$, and MSO-defined decision problems lie in NC^1. These results covered $\#\mathrm{NC}^1$- and NC^1-complete problems, respectively. The technique of balancing tree decompositions in FTC^0 allowed to separate the input balancing from the actual problem-solving step, in contrast to the more involved common approach of combining these steps.

For input structures of bounded tree depth, the logspace result was transferred to constant-depth circuits: It was shown that MSO-definable decision problems can be solved in AC^0, number representations of solution histograms can be computed in GapAC^0, and string representations of solution histograms can be computed in TC^0. For the results related to GapAC^0 and TC^0 the main technical development was the definition of multiset tree automata and an algebraic representation of their computations. The results where applied to show the TC^0-completeness of unary versions of number problems like SUBSETSUM, KNAPSACK, and solving integer linear equation systems with a constant number of equations. The AC^0 result followed as a consequence of the pure model-theoretic result that first-order and GSO-formulæ express the same properties on any tree-depth-bounded class of structures. For proving this result, a new constructive Feferman–Vaught-type theorem for unbounded partitions was developed.

7.2. Outlook

There are a number of open questions that arise from the results of the present thesis:

Construct and enumerate solutions. All theorems and intermediate results in the thesis talk about either deciding whether solutions exist or counting the number thereof. It would be interesting to extend the results to *construct solutions*, instead of just deciding their existence, and *enumerate solutions*, instead of just counting them. Preliminary findings indicate that constructing *some* solution is quite easy, but enumerating all of them systematically appears to be difficult. Before diving into the development of such results, it would be helpful to have a clear and compelling complexity-theoretic application in mind that uses the construction and enumeration of solutions.

Generalizations of monadic second-order logic. It would be interesting to know for which kind of generalizations of MSO-logic the main results of the thesis still hold: Courcelle and Engelfried (2012) consider *counting* MSO-*logic*, which extends the expressive power of MSO to also test whether the cardinality of relations is a multiple of constants. Thus, properties like "is there a dominating set of even length" are directly definable in the logic without the need to refer to a solution histogram. While there is no difference in solving MSO- and counting MSO-properties in the case of bounded tree width—bounded tree width structures can always be enriched by an ordering on the elements of the universe, and on such structures the expressive power of MSO and counting MSO coincides—, there is a difference in the case of tree-depth-bounded structures. Adding an order to a structure of bounded tree depth increases the longest path length in its Gaifman graph and, thus, its tree depth. I believe that, while MSO-definable decision problems are solvable in AC^0, counting MSO-definable decision problems is solvable (only) in ACC (that means in $\mathrm{AC}^0[m]$ for some m depending on the formula at hand). Moreover, as the result on MSO-formulæ and AC^0 is the consequence of a pure model-theoretic result, this might also hold for counting MSO-formulæ and ACC.

A challenging task seems to be counting the number of solutions that are defined by counting MSO-formulæ. Is this still possible in TC^0 for tree-depth-bounded structures?

Number versus string representations of histograms. There is a subtle, but important, difference between the results that are developed for logspace and constant-depth circuits on the one hand and logarithmic-depth circuits on the other hand. In the first case, we have theorems that compute string representations of histograms, while in the second case the result for $\#\mathrm{NC}^1$ computes number representations only. This means, for example, that we cannot solve MSO-definable optimization problems in $\#\mathrm{NC}^1$ based on the computation of histograms since it is not known whether $\#\mathrm{NC}^1$-circuits can look up individual bits of string representations of their computed numbers. More could be said about this if the long-standing open question of whether FNC^1 equals $\#\mathrm{NC}^1$ would be resolved.

From bounded tree width to bounded clique width. Beside the notion of bounded tree width, which is based on width-bounded tree decompositions, another widely studied width notion is *bounded clique width*, which is based on clique expressions that use a bounded number of colors (Courcelle and Engelfried, 2012). Since clique width is more general than tree width in the sense that any graph class of bounded tree width has bounded clique width, it would be interesting to know whether clique expressions can be computed in logspace. Once they are available as terms, the techniques related to logarithmic-depth circuits can also be used in this setting.

Parameter-dependent complexity-theoretic results. The theory of parameterized complexity studies how parameters that are defined on input instances influence the complexity of problems. For example, one can view the tree width of a graph $\mathcal{G}$ as a parameter $p := \mathrm{tw}(\mathcal{G})$ and ask what kind of algorithms exist to compute its tree width. Is there an algorithm with running time $|\mathcal{G}|^{f(p)}$ for some function f (then the problem TREE-WIDTH lies in the parameterized class XP), or is there an algorithms that runs in time $f(p)|\mathcal{G}|^{O(1)}$ (then TREE-WIDTH is in FPT)? While both algorithms solve the problem for any constant p in polynomial time, an algorithm of the later kind, which exists (Bodlaender, 1996), is more desirable since the degree of the polynomial that depends on the size of the input is *fixed.* The notions of varying and fixed polynomial degrees can also be transferred to the logspace case (Flum and Grohe, 2003): The class XL covers problems that can be solved using space $f(p) \log |\mathcal{G}|$, and the class para-L covers problems that only need space $f(p) + O(\log |\mathcal{G}|)$. While for XL the amount of logarithmic space needed varies for different p, the amount of logarithmic space needed for para-L is fixed. The theorems for logspace from this thesis put problems into the class XL. For example, Theorem 6.14 can be rephrased to prove TREE-WIDTH $\in$ XL with respect to the parameter $\mathrm{tw}(\mathcal{G})$, but does also TREE-WIDTH $\in$ para-L hold?

More applications. The aim of my thesis was to present the main results on solving MSO-definable problems. Moreover, applications of these results where given to illustrate their unifying character. Building on the conference papers that found their way into this thesis, applications to problems related to graph isomorphism (Wagner, 2011) and logics for non-monotonic reasoning (Meier et al., 2012) have already been observed. Seeing the large impact of Courcelle's Theorem for designing algorithms, it seems likely that there are many more complexity-theoretic applications of the results of this thesis.

List of Terms

This list describes standard terms that are used in the main text without definition, and terms that are used in the introduction and in the introductory parts of other chapters before their formal definition.

Terms that solely base on non-latin characters appear first in the list, followed by terms that are sorted in accordance with the latin characters they contain; for example $\#\mathrm{NC}^1$ is sorted using the key NC.

δ	Transition function of an automaton. For the case of multiset tree automata see Section 2.1.
τ	A finite relational vocabulary, see Section 2.2 for details.
$\models$	The model relation, see Section 2.2 for details.
φ, ψ, ρ	Denote logical formulæ, see Section 2.2 for details.
α	A variable assignment, see Section 2.2 for details and related terms.
$\mathcal{A}$	A finite relational structure, see Section 2.2 for details.
AC^0	The class of languages decidable by constant-depth Boolean circuits with unbounded fan-in gates.
AC^1	The class of languages decidable by logarithmic-depth Boolean circuits with unbounded fan-in gates.
$\mathcal{A}_{\mathrm{inci}}$	The incidence representation of a structure $\mathcal{A}$, see Section 3.4 for details.
DLOGTIME	The class of languages decidable by random-access logarithmic-time deterministic Turing machines. The notion of DLOGTIME-uniform circuit families is based on this class, see Section 4.1 for details.
DTM	Deterministic Turing machine.
FO	The class of languages decidable by first-order formulæ with build-in ordering and arithmetic predicates. The notion of FO-computations is based on this class, see Section 4.1 for details.
$\mathcal{G}$, G	Graphs are structures $\mathcal{G} = (V, E^{\mathcal{G}})$ over the vocabulary $\tau_{\mathrm{graph}} = \{E^2\}$. Hence, by definition they are directed; undirected graphs are structures $\mathcal{G}$ with a symmetric edge relation. The shorter notation $G = (V, E)$ is also used. Graph-theoretic terms are given in Definition 2.2, see Definition 2.13 for graphs as logical structures.
$G(\cdot)$	The Gaifman graph of a structure, see Definition 3.11.
GapAC^0	The class of functions $f\colon \{0,1\}^* \to \mathbb{Z}$ computable by constant-depth arithmetic circuits with unbounded fan-in gates from $\{+, -, \cdot\}$.
$\mathrm{histogram}(\mathcal{A}, \varphi)$	The solution histogram of a structure $\mathcal{A}$ and a formula φ; an array that stores the number of solutions to the formula with respect to their cardinalities.
k-tree	An undirected graph of tree width k that is maximal in the sense that adding any edge increases its tree width.
L	The class of languages decidable by logarithmic-space-bounded deterministic Turing machines.

LOGCFL	The class of languages decidable by logarithmic-depth Boolean circuits with $\vee$-gates of unbounded fan-in and $\wedge$-gates of bounded fan-in.
$\mathrm{lpl}(\cdot)$	The length of a longest path in a graph. Graphs with bounded longest path length have bounded tree depth, and vice versa (see Section 3.2).
MSO	Monadic second-order.
multiset	Multisets generalize the notion of sets; they can contain the same element more than once. Terms regarding multisets are defined on page 15.
$\mathbb{N}$	Denotes $\{1, 2, 3, \dots\}$, the set of positive integers.
$\mathbb{N}_0$	Denotes $\{0, 1, 2, 3, \dots\}$, the set of nonnegative integers.
NC	The class of languages decidable by polylogarithmic-depth Boolean circuits.
NC^1	The class of languages decidable by logarithmic-depth Boolean circuits with bounded fan-in gates.
$\#\mathrm{NC}^1$	The class of functions $f\colon \{0,1\}^* \to \mathbb{N}_0$ computable by logarithmic-depth arithmetic circuits with bounded fan-in gates from $\{+, \cdot\}$.
NL	The class of languages decidable by logarithmic-space-bounded nondeterministic Turing machines.
NP	The class of languages decidable by nondeterministic Turing machines that are polynomial-time-bounded.
$\mathrm{num}(h)$	The number encoding of a histogram h, see pages 56f. for details.
P	The class of languages decidable by deterministic Turing machines that are polynomial-time-bounded.
PERFECT-MATCHING	The language of string-encoded graphs $\mathrm{str}(\mathcal{G})$ that have a perfect matching.
REACHABILITY	The language of string-encoded graphs $\mathrm{str}(\mathcal{G})$ and vertices $s, t \in V(\mathcal{G})$ with a directed path from s to t.
$\mathrm{str}(\cdot)$	The string encoding of a structure $\mathcal{A}$ or a histogram h. See page 46 in the first and pages 56f. in the second case.
SUBSETSUM	The problem UNARY-SUBSETSUM, which is formally defined on page 59, but with binary-encoded input numbers.
T, (T, l), $\mathcal{T}$	Graph-theoretic terms related to trees T are given in Definition 2.3, for labeled trees (T, l) see Definition 2.4. For trees as logical structures $\mathcal{T}$ and their extension to s-tree structures see Definition 2.12.
(T, B)	A tree decomposition with tree T and bag labeling function B, see Definition 4.1.
TC^0	The class of languages that are decidable by constant-depth Boolean circuits with unbounded fan-in threshold gates.
$\mathrm{td}(\cdot)$	The tree depth of a structure, see Definition 3.12.
$T(\mathcal{T}, S_1, \dots, S_k)$	A labeled tree that arises from an s-tree structure and k solution sets as defined on page 23.
$\mathrm{tw}(\cdot)$	The tree width of a structure, see Definition 4.3.

Bibliography

K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10(2):287–302, 1989. doi:10.1016/0196-6774(89)90017-5.

M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. *Journal of Computer and System Sciences*, 60(2):395–421, 2000. doi:10.1006/jcss.1999.1675.

E. Allender. Arithmetic circuits and counting complexity classes. In J. Krajíček, editor, *Complexity of Computations and Proofs*, volume 13 of *Quaderni di Matematica*, pages 33–72. Seconda Universita di Napoli, 2004. http://ftp.cs.rutgers.edu/pub/allender/quaderni.pdf.

E. Allender and K.-J. Lange. $\mathrm{RUSPACE}(\log n) \subseteq \mathrm{DSPACE}(\log^2 n/\log\log n)$. *Theory of Computing Systems*, 31(5):539–550, 1998. doi:10.1007/s002240000102.

A. Ambainis, D. A. Mix Barrington, and H. LêThanh. On counting AC^0 circuits with negative constants. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS 1998)*, volume 1450 of *Lecture Notes in Computer Science*, pages 409–417. Springer, 1998. doi:10.1007/BFb0055790.

S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.

A. Blumensath, T. Colcombet, and C. Löding. Logical theories and compatible operations. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 73–106. Amsterdam University Press, 2007. http://automata.rwth-aachen.de/download/papers/loeding/blcolo07.pdf.

H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In *Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1988)*, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1989. doi:10.1007/3-540-50728-0_32.

H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1. Technical Report HKUST-TCSC-2001-05, The Hongkong University of Science and Technology, 2001. http://hdl.handle.net/1783.1/738.

J. R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1–6):66–92, 1960. doi:10.1002/malq.19600060105.

S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992. doi:10.1137/0221046.

S. R. Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*, pages 123–131. ACM, 1987. doi:10.1145/28395.28409.

S. R. Buss. Algorithms for boolean formula evaluation and for tree contraction. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 95–115. Oxford University Press, 1993. http://math.ucsd.edu/~sbuss/ResearchWeb/Boolean3/.

H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57(2):200–212, 1998. doi:10.1006/jcss.1998.1588.

A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *RAIRO - Theoretical Informatics and Applications*, 35:259–275, 2001. doi:10.1051/ita:2001119.

S. Cho and D. T. Huynh. On a complexity hierarchy between L and NL. *Information Processing Letters*, 29(4):177–182, 1988. doi:10.1016/0020-0190(88)90057-9.

S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1–3):2–22, 1985. doi:10.1016/S0019-9958(85)80041-3.

S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(5):385–394, 1987. doi:10.1016/0196-6774(87)90018-6.

B. Courcelle. On recognizable sets and tree automata. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, volume 1, pages 93–126. Academic Press, Inc., 1989.

B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press, 1990.

B. Courcelle. The monadic second-order logic of graphs xiv: uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299(1–3):1–36, 2003. doi:10.1016/S0304-3975(02)00578-9.

B. Courcelle and J. Engelfried. *Graph structure and monadic second-order logic, a language theoretic approach*. Cambridge University Press, 2012. http://hal.archives-ouvertes.fr/hal-00646514/fr/.

B. Das, S. Datta, and P. Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, volume 5 of *Leibniz International Proceedings in Informatics*, pages 215–226. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010. doi:10.4230/LIPIcs.STACS.2010.2456.

R. Diestel. *Graph Theory*. Springer, 2005. http://diestel-graph-theory.com/index.html.

J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970. doi:10.1016/S0022-0000(70)80041-1.

P. Dymond. Input-driven languages are in $\log n$ depth. *Information Processing Letters*, 26(5):247–250, 1988. doi:10.1016/0020-0190(88)90148-2.

M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of bodlaender and courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.

M. Elberfeld, M. Grohe, and T. Tantau. Where first-order and monadic second-order logic coincide. In *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2012)*. IEEE Computer Society, 2012a. http://www.tcs.uni-luebeck.de/downloads/papers/2012/msofo.pdf. to appear.

M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics*, pages 66–77. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012b. doi:10.4230/LIPIcs.STACS.2012.66.

C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51, 1961. doi:10.2307/1993511.

S. Feferman and R. Vaught. The first order properties of algebraic systems. *Fundamenta Mathematicæ*, 47:57–103, 1959. http://matwbn.icm.edu.pl/ksiazki/fm/fm47/fm4715.pdf.

J. Flum and M. Grohe. Describing parameterized complexity classes. *Information and Computation*, 187:291–319, 2003. doi:10.1016/S0890-5401(03)00161-5.

J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin Heidelberg, 2006. doi:10.1007/3-540-29953-X.

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

G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL certificates. *Theoretical Computer Science*, 270(1–2):761–777, 2002. doi:10.1016/S0304-3975(01)00108-6.

G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of XPath query evaluation and XML typing. *Journal of the ACM*, 52(2):284–335, 2005. doi:10.1145/1059513.1059520.

E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *ACM Trans. Comput. Logic*, 3:418–463, 2002. doi:10.1145/507382.507388.

M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2011. http://www2.informatik.hu-berlin.de/~grohe/pub/grokre11.pdf.

W. Hesse, E. Allender, and D. A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

O. H. Ibarra, T. Jiang, B. Ravikumar, and J. H. Chang. On some languages in NC^1. In *Proceedings of the Aegean Workshop on Computing: 3rd International Workshop on Parallel Computation and VLSI Theory*, volume 319 of *Lecture Notes in Computer Science*, pages 64–73. Springer, 1988.

doi:10.1007/BFb0040374.

N. Immerman. *Descriptive complexity.* Springer, New York, 1999.

A. Jakoby and T. Tantau. Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *Proceedings of the 27th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2007)*, volume 4855 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2007. doi:10.1007/978-3-540-77050-3_18.

A. Jakoby, M. Liśkiewicz, and R. Reischuk. Space efficient algorithms for series-parallel graphs. *Journal of Algorithms*, 60(2):85–114, 2006. doi:10.1016/j.jalgor.2004.06.010.

B. Jenner. Knapsack problems for NL. *Information Processing Letters*, 54(3):169–174, 1995. doi:10.1016/0020-0190(95)00017-7.

N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11(1):68–85, 1975. doi:10.1016/S0022-0000(75)80050-X.

D. M. Kane. Unary subset-sum is in logspace. *CoRR*, abs/1012.1336, 2010. http://arxiv.org/abs/1012.1336.

A. Krebs, N. Limaye, and M. Mahajan. Counting paths in VPA is complete for #NC^1. In *Proceedings of the 16th Annual International Conference on Computing and Combinatorics (COCOON 2010)*, volume 6196 of *Lecture Notes in Computer Science*, pages 44–53. Springer, 2010. doi:10.1007/978-3-642-14031-0_7.

R. E. Ladner and N. A. Lynch. Relativization of questions about log space computability. *Theory of Computing Systems*, 10:19–32, 1976. doi:10.1007/BF01683260.

L. Libkin. *Elements Of Finite Model Theory.* Springer, 2004.

L. Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006. doi:10.2168/LMCS-2(3:2)2006.

M. Lohrey. On the parallel complexity of tree automata. In *Proceedings of 12th International Conference on Rewriting Techniques and Applications (RTA 2001)*, volume 2051 of *Lecture Notes in Computer Science*, pages 201–215. Springer, 2001. doi:10.1007/3-540-45127-7_16.

J. Makowsky. Algorithmic uses of the Feferman–Vaught theorem. *Annals of Pure and Applied Logic*, 126(1–3):159–213, 2004. doi:10.1016/j.apal.2003.11.002.

A. Meier, J. Schmidt, M. Thomas, and H. Vollmer. On the parameterized complexity of default logic and autoepistemic logic. In *6th International Conference on Language and Automata Theory and Applications (LATA 2012)*, volume 7183 of *Lecture Notes in Computer Science*, pages 389–400. Springer, 2012. doi:10.1007/978-3-642-28332-1_33.

G. L. Miller and J. H. Reif. Parallel tree contraction and its application. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS 1985)*, pages 478–489. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.43.

D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.

B. Monien. On a subclass of pseudopolynomial problems. In *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science (MFCS*

1980), volume 88 of *Lecture Notes in Computer Science*, pages 414–425, 1980. doi:10.1007/BFb0022521.

J. Nešetřil and P. Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008. doi:10.1016/j.ejc.2006.07.013.

C. H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4): 1–24, 2008. doi:10.1145/1391289.1391291.

N. Robertson and P. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

C. Stockhusen. Anwendungen monadischer Logik zweiter Stufe auf Probleme beschränkter Baumweite und deren Platzkomplexität. Diploma Thesis, 2011. http://www.tcs.uni-luebeck.de/downloads/papers/2011/Christoph_Stockhusen_Diplomarbeit.pdf. In German.

L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 5th annual ACM symposium on Theory of computing (STOC 1973)*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

H. Straubing. *Finite automata, formal logic, and circuit complexity.* Birkhäuser, 1994.

J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968. doi:10.1007/BF01691346.

C. Thomassen. On the presence of disjoint subgraphs of a specified type. *Journal of Graph Theory*, 12(1):101–111, 1988. doi:10.1002/jgt.3190120111.

B. A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329, 1961. In Russian.

H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer, Berlin Heidelberg, 1999.

F. Wagner. Graphs of bounded treewidth can be canonized in AC^1. In *6th International Computer Science Symposium in Russia (CSR 2011)*, volume 6651 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2011. doi:10.1007/978-3-642-20712-9_16.

E. Wanke. Bounded tree-width and LOGCFL. *Journal of Algorithms*, 16(3): 470–491, 1994. doi:10.1006/jagm.1994.1022.