Electronic Colloquium on Computational Complexity, Report No. 128 (2011)

Algorithmic Meta Theorems for Circuit Classes of Constant and Logarithmic Depth

Michael Elberfeld Andreas Jakoby Till Tantau

Institut für Theoretische Informatik
Universität zu Lübeck
D-23538 Lübeck, Germany
{elberfeld,jakoby,tantau}@tcs.uni-luebeck.de

September 21, 2011

Abstract

An algorithmic meta theorem for a logic and a class C of structures states that all problems expressible in this logic can be solved efficiently for inputs from C. The prime example is Courcelle's Theorem, which states that monadic second-order (MSO) definable problems are linear-time solvable on graphs of bounded tree width. We contribute new algorithmic meta theorems, which state that MSO-definable problems are (a) solvable by uniform constant-depth circuit families (AC^0 for decision problems and TC^0 for counting problems) when restricted to input structures of bounded tree depth and (b) solvable by uniform logarithmic-depth circuit families (NC^1 for decision problems and $\#NC^1$ for counting problems) when a tree decomposition of bounded width in term representation is part of the input. Applications of our theorems include a TC^0-completeness proof for the unary version of integer linear programming with a fixed number of equations and extensions of a recent result that counting the number of accepting paths of a visible pushdown automaton lies in $\#NC^1$. Our main technical contributions are a new tree automata model for unordered, unranked, labeled trees; a method for representing the tree automata's computations algebraically using convolution circuits; and a lemma on computing balanced width-3 tree decompositions of trees in TC^0, which encapsulates most of the technical difficulties surrounding earlier results connecting tree automata and NC^1.

Keywords: algorithmic meta theorem, monadic second-order logic, circuit complexity, tree width, tree depth

1 Introduction

Courcelle's Theorem [16] states that every monadic second-order (MSO) definable problem can be decided in linear time on graphs of bounded tree width. Since many important graph properties are easily expressible in this logic, Courcelle's Theorem yields a unified framework for showing that numerous problems on graphs of bounded tree width are solvable in linear time. Recently we showed that both Courcelle's Theorem as well as its later extensions [7] also hold when "linear time" is replaced by "logarithmic space" [20], making the power of MSO-definability available for the study of logarithmic space.

The present paper furthers this line of research and transfers the idea of unified MSO-based problem definitions to circuit classes inside logarithmic space. During the course of this paper we identify MSO-based algorithmic meta theorems that place problems in the circuit classes AC^0, $GapAC^0$, TC^0, NC^1, and $\#NC^1$. The classes AC^0, $GapAC^0$, and TC^0 are defined via Boolean (AC^0), arithmetic ($GapAC^0$), and threshold (TC^0) circuit families of constant depth and unbounded fan-in (detailed definitions will be given in Section 2). The classes NC^1 and $\#NC^1$ are defined via Boolean and arithmetic circuits, respectively, of logarithmic depth and bounded fan-in.

ISSN 1433-8092

The inputs for Courcelle's Theorem are graphs of bounded tree width and many MSO-definable problems on such graphs are complete for logarithmic space, including even the question of whether the graph has a certain tree width [20], but also the reachability problem for trees. Thus, in order to prove algorithmic meta theorems that place the complexity of problems inside subclasses of L, we either need to restrict the logic or we need to restrict the kinds of inputs allowed. In the present paper, we consider the latter case: For the constant-depth circuit classes, we only allow input graphs that have *bounded tree depth* (a restriction of bounded tree width). For the logarithmic-depth circuit classes we allow arbitrary graphs of bounded tree width as input, but require that the graphs are *accompanied by tree decompositions in term representation*.

1.1 Our Contributions

Bounded Tree Depth Structures and Constant-Depth Circuits Our first contribution is a set of meta theorems placing problems in constant-depth circuit classes. The inputs for these theorems are structures that have bounded tree *depth*, a measure on graphs that was introduced by Nešetřil and Ossona de Mendez [34] to quantify the similarity of graphs to star graphs (in opposition to tree width, which quantifies the similarity of graphs to trees). Characterizations of when a class C of graphs has bounded tree depth include: (a) All graphs in C have a tree decomposition of both bounded width and depth; or alternatively (b) all graphs in C have bounded longest path length. The tree depth of a logical structure is the tree depth of its Gaifman graph (detailed definitions will be given in Section 2).

Theorem 1.1 (Decision Using Boolean Constant-Depth Circuits)**.** *For every* MSO*-formula ϕ over some signature τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* AC^0*-circuit family that, on input of an arbitrary τ-structure $\mathcal{S}$, outputs* 1 *if, and only if, the tree depth of $\mathcal{S}$ is at most d and $\mathcal{S} \models \phi$ holds.*

As an example application, consider the problem of deciding whether a graph has a perfect matching. The complexity of this MSO-definable problem has been studied in detail and its complexity varies in dependence of the class of graphs under consideration. By the above theorem, deciding whether a graph of bounded tree depth has a perfect matching lies in AC^0. In contrast, it is known that the same problem for graphs of bounded tree *width* is L-complete [17, 20].

Instead of just deciding whether a formula is satisfied by a logical structure, when the formula has a free second-order variable, we can try to count the number of assignments of sets to the free variable that make the formula true. Moreover, if we count the number of solutions with respect to the sizes of these sets, this leads to *cardinality versions* of Courcelle's Theorem. These cardinality versions allow a much broader range of applications than the decision version and we will show how both known results from the literature and also new results can be proved in an elegant manner using these versions.

To formulate the cardinality versions, we need a bit of terminology: Let $\phi(X_1,\dots,X_\ell,Y_1,\dots,Y_k)$ be an MSO-formula with two sets of free set variables, namely the X_i and the Y_j, and let $\mathcal{S}$ be a logical structure with universe S. The *solution histogram* of ϕ and $\mathcal{S}$, denoted by $\mathrm{hist}(\mathcal{S},\phi)$, is an ℓ-dimensional integer array that tells us "how many solutions of a certain size exist". In detail, let $s = (s_1,\dots,s_\ell) \in \{0,\dots,|S|\}^\ell$ be an index vector that prescribes sizes for the sets that are substituted for the X_i. Then $\mathrm{hist}(\mathcal{S},\phi)[s]$ equals the number of subsets $S_1,\dots,S_\ell,S'_1,\dots,S'_k \subseteq S$ with $|S_1| = s_1$, …, $|S_\ell| = s_\ell$ and $\mathcal{S} \models \phi(S_1,\dots,S_\ell,S'_1,\dots,S'_k)$. 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. As a first example, let $\phi_{\mathrm{dom}}(X_1) = \forall x\big(X_1(x) \vee \exists y(X_1(y) \wedge E(y,x))\big)$, which expresses that X_1 is a dominating set in a graph with edge relation E. Then $\mathrm{hist}(\mathcal{G},\phi_{\mathrm{dom}})[s_1]$ is the number of dominating sets of size s_1 in the graph $\mathcal{G}$. As a second example, let $\phi_{\mathrm{matching}}(Y_1)$ be the formula expressing that Y_1 is an edge set that is a perfect matching in G. Then, since $\ell = 0$, the histogram $\mathrm{hist}(\mathcal{G},\phi_{\mathrm{matching}})$ is just a scalar value that tells us how many perfect matchings G has.

In order to represent a 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 $k|S|$ will suffice). Then $\mathrm{num}(h)$ is the single number representing the whole of the memory contents (a formal definition of $\mathrm{num}(h)$ will be given later). In particular, the bits of any single entry of h can easily be obtained from the bits of $\mathrm{num}(h)$.

Theorem 1.2 (Histogram Computation Using Arithmetic Constant-Depth Circuits)**.** *For every* MSO*-formula $\phi(X_1,\dots,X_\ell,Y_1,\dots,Y_k)$ over some signature τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* GapAC^0*-circuit family that, on input of a τ-structure $\mathcal{S}$ of tree depth at most d, outputs* $\mathrm{num}(\mathrm{hist}(\mathcal{S},\phi))$.

Note that by Theorem 1.1, we can check in AC^0 whether an input structure $\mathcal{S}$ has tree depth d and, if not, we could output an error value like -1. Applying Theorem 1.2 to the formula $\mathrm{hist}(\mathcal{G},\phi_{\text{matching}})$ shows that counting the number of perfect matchings in graphs of bounded tree depths lies in GapAC^0. Since GapAC^0 is contained in FTC^0, the functional version of the class TC^0 of constant-depth circuits with threshold gates, computing a particular bit of the number $\mathrm{num}(\mathrm{hist}(\mathcal{S},\phi))$ can be done using a TC^0-circuit:

Corollary 1.3. *For every* MSO*-formula $\phi(X_1,\dots,X_\ell,Y_1,\dots,Y_k)$ over some signature τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* TC^0*-circuit family that, on input of a τ-structure $\mathcal{S}$ of tree depth at most d, an ℓ-dimensional index s, and a bit position i, outputs the ith bit of* $\mathrm{hist}(\mathcal{S},\phi)[s]$.

We cannot hope to place the computation of solution histograms in any complexity class smaller than FTC^0 since the TC^0-complete problem MAJORITY is easily expressible using an MSO-formula and the histogram: Turning a string s into a logical structure $\mathcal{S} = (\{1,\dots,|s|\}, P_1^{\mathcal{S}})$ in the usual manner by setting $i \in P_1^{\mathcal{S}} \Leftrightarrow s[i] = 1$, for the MSO-formula $\phi(X_1) = \forall x(X_1(x) \to P_1(x))$ more than half of the input bits are 1 if, and only if, $\mathrm{hist}(\mathcal{S},\phi)\big[\lfloor |s|/2 \rfloor + 1\big] > 0$. In Section 3.5 we show how extensions of this idea can be used to prove the TC^0-completeness of the unary version of SUBSETSUM and also of other number problems, the most powerful being unary integer linear programming with a constant number of equations.

Bounded Width Tree Decompositions in Term Representation and Logarithmic-Depth Circuits Our second contribution are algorithmic meta theorems for NC^1 and its arithmetic companion class $\#\mathrm{NC}^1$. For these theorems the input structure is equipped with a tree decomposition of bounded width (no longer of bounded depth, though) given in term representation. The term representation of a tree like  is the string $[\,[\,]\,[\,[\,]\,[\,]\,]\,]$, which exhibits the tree's ancestor relation. It is this ancestor relation that is needed as part of the input; the bags and their contents can be encoded in any reasonable way such as a set of pairs of tree node numbers and associated bags.

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

Unless $\mathrm{NC}^1 = \mathrm{L}$, neither can the tree decomposition be omitted from the input nor can it be represented as a pointer structure, because even given a tree decomposition of the input graph as a pointer structure as part of the input, the reachability problem is still an MSO-definable L-complete problem.

As an example of an application of the theorem, consider the problem of deciding the language accepted by a tree automaton. It is well known that every such language lies in NC^1 [30, 23]. The above theorem allows us to reprove this fact succinctly: an MSO-formula can easily check (using existential second-order quantifiers) whether there is an assignment of states to the nodes of the tree that is locally consistent and that makes the automaton accept.

Theorem 1.5 (Histogram Computation Using Arithmetic Logarithmic-Depth Circuits)**.** *For every* MSO*-formula $\phi(X_1,\dots,X_\ell,Y_1,\dots,Y_k)$ over some signature τ and every $w \in \mathbb{N}$, there is a* DLOGTIME*-uniform*

$\#\mathrm{NC}^1$*-circuit family that, on input of a* τ*-structure* $\mathcal{S}$ *along with a width-*w *tree decomposition in term representation for* $\mathcal{S}$*, outputs* $\mathrm{num}(\mathrm{hist}(\mathcal{S}, \phi))$.

As an application of this theorem, we will show in Section 4.5 how it can be used to compute the number of accepting paths of nondeterministic visible pushdown automata.

Technical Contributions The proofs of the algorithmic meta theorems for constant-depth circuits rest on two new technical tools. First, we introduce a new model of automata, which we call *multiset automata*, that exactly captures the MSO-definable problems on unordered unranked labeled trees. We show that multiset automata enjoy standard closure properties like closure under union or complement and that for every nondeterministic multiset automaton there exists an equivalent deterministic one. Introducing a new automaton model turned out to be necessary since standard automata-theoretic approaches to proving meta theorems cannot be applied in the context of constant-depth circuits: all known approach include preprocessing steps that enlarge the depth of input trees by at least a logarithmic factor, making them unfeasible for simulation by constant-depth circuits. Second, we develop algebraic representations of the computations of multiset automata using arithmetic circuits that keep track of the number of ways in which states can be reached.

In the context of research on logarithmic-depth circuits, trees in term representation are a natural form of input. In many papers (including the present), a central problem is that a logarithmic-depth circuit cannot work on the term representation directly when it has large depth. Instead, some sort of balancing must be done where the represented tree is recursively divided into smaller parts. 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. We present a new way of dealing with trees of arbitrary depth: We develop an algorithm that takes a tree T as input and outputs a width-3 tree decomposition of T that is perfectly balanced and, hence, has logarithmic depth. The bags of this decomposition form a hierarchical separation of T into subtrees along which a recursive algorithm can work. A key property of our construction is that it can be performed by uniform constant-depth threshold circuits.

1.2 Related Work

Algorithmic meta theorems for first-order and monadic second-order logic have been intensively studied from the perspective of achieving a low runtime (a current overview can be found in the survey article by Grohe and Kreutzer [24]). While these results place many problems in LINTIME or at least in P (or FPT when viewed as parameterized problems), there is less work on meta theorems that lead to exact classifications in complexity theoretic terms. Two exceptions are a paper of Wanke [46], which shows that all problems that are captured by Courcelle's Theorem are in LOGCFL, and our paper [20], which places these problems further down into L.

Tree automata-based techniques are routinely used to prove time- and space-efficient variants of Courcelle's Theorem [7, 20]. The problem of deciding whether a fixed tree automaton accepts a given tree in term representation lies in NC^1 both in the ranked [30] and the unranked case [23]. As discussed in Section 4.5, Theorem 1.5 can be used to generalize these results and prove that the number of accepting computations of nondeterministic tree automata can be counted in $\#\mathrm{NC}^1$.

Buss [12] used pebbling-based strategies to evaluate Boolean sentences in uniform NC^1. His method was later adopted to evaluate arithmetic sentences [11] and, more recently, to prove that the number of accepting computations of nondeterministic visible pushdown automata can be counted in $\#\mathrm{NC}^1$ [28]. Our proof plan for the Theorems 1.4 and 1.5 differs from these techniques: We do not evaluate and balance the input at the same time using logarithmic-depth circuits. We consider these tasks separately by first computing a balanced (and, thus, logarithmic-depth bounded) version of the input structure in FTC^0 and,

then, proceed to work on the balanced structure using logarithmic-depth circuits without applying any balancing strategies.

1.3 Organization of This Paper

The paper is organized as follows: In Section 2, we will discuss the logical, graph theoretic and complexity theoretic background of our work. In Section 3 we prove algorithmic meta theorems for constant-depth circuit classes. Section 4 contains proofs of the algorithmic meta theorems for classes defined via logarithmic-depth circuits; both Sections 3 and 4 contain discussions of how to apply the theorems to yield new and unify old results on the complexity of particular problems.

2 Background

2.1 Logical Structures and Monadic Second-Order Logic

A *signature* τ (or vocabulary) is a set of *relation symbols* R together with a mapping assigning an *arity* $r \geq 1$ to each relation symbol. In slight abuse of notation, we write $R \in \tau$ to indicate that R lies in τ and $R^r \in \tau$ to additionally indicate that R has arity r. A (finite) *τ-structure* $\mathcal{S} = (S, R_1^{\mathcal{S}}, \ldots, R_m^{\mathcal{S}})$ consists of a nonempty, finite set S, the *universe* of $\mathcal{S}$, and for each relation symbol $R_i^{r_i} \in \tau$ a *relation* $R_i^{\mathcal{S}} \subseteq S^{r_i}$. In the present paper we consider only finite structures.

Monadic second-order logic (MSO-logic) is the fragment of second-order logic where all variables are either first-order variables $x_1, x_2, \ldots$ (also called *element variables*) or unary second-order variables $X_1, X_2, \ldots$ (also called *set variables*). The MSO-*formulas* over a vocabulary τ are inductively defined as follows: The *atomic formulas* are of the forms $x = y$, $X(z)$, $R(x_1, \ldots, x_r)$, where $x, y, z, x_1, \ldots, x_r$ are element variables, X is a set variable, and $R^r \in \tau$. Formulas are build from atomic formulas by *connectives* $\neg$ and $\wedge$, *existential element quantifiers* $\exists x$, and *existential set quantifiers* $\exists X$. *Bound* and *free* variables are defined as usual. We write $\phi(X_1, \ldots, X_\ell)$ for a formula ϕ with free set variables $X_1, \ldots, X_\ell$. For a vocabulary τ, a τ-structure $\mathcal{S}$ with universe S, a τ-formula $\phi(X_1, \ldots, X_\ell)$, and sets $S_1, \ldots, S_\ell \subseteq S$, we write $\mathcal{S} \models \phi(S_1, \ldots, S_\ell)$ to indicate that ϕ holds in the structure $\mathcal{S}$ if each X_i is interpreted as S_i.

If we want structures to be processed by circuits, we need to encode them as strings. We do so in the usual way, see also [26] for a detailed discussion: An r-ary relation is represented by a bitstring whose ith bit is 1 if, and only if, the ith r-tuple (in lexicographic order) of elements of the universe is an element of the relation. We denote the encoding of a logical structure $\mathcal{S}$ by $\text{code}(\mathcal{S})$.

2.2 Graphs and Trees

A *(directed) graph* is a pair G, consisting of a set $V(G)$ of *vertices* and a set $E(G) \subseteq V(G) \times V(G)$ of *edges*. A *subgraph* of a graph G is a graph G' with $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The *subgraph of G induced on a set $U \subseteq V(G)$* is the graph $G[U]$ with $V(G[U]) = U$ and $E(G[U]) = E(G) \cap (U \times U)$. The *children* of a vertex v of a directed graph G are all vertices u with $(v,u) \in E(G)$. A *(simple) path* from a vertex $s \in V(G)$ to a vertex $t \in V(G)$ is a sequence of distinct vertices $v_1, \ldots, v_m$ with $s = v_1$, $t = v_m$, and $(v_i, v_{i+1}) \in E(G)$ for all $i \in \{1, \ldots, m-1\}$; the path's length is $m-1$. The maximum length of a simple path in a graph G is called its *longest path length* and denoted by $\text{lpl}(G)$. We treat *undirected graphs* as special cases of directed graphs, namely as directed graphs with a symmetric edge relation. An undirected graph is *connected* if there exists a path between any two of its vertices. The *components* $C_1, \ldots, C_m$ of an undirected graph G are its maximal connected subgraphs; the empty graph has zero components. Graphs can be seen as logical structures, namely $\{E^2\}$-structures $\mathcal{G} = (V, E^{\mathcal{G}})$.

A *(directed) tree* is a directed graph T together with 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. Nodes without children are called *leaves*. The *degree* of a tree (sometimes also called the *rank* of the

tree) is the maximum number of children any node has. A *(directed) forest* is a disjoint union of rooted directed trees. The *depth* of a forest F is $\mathrm{lpl}(F)$. A *labeled forest over an alphabet* Σ is a pair (F,l) that consists of a forest F and a mapping $l\colon V(F)\to\Sigma$.

2.3 Tree Width and Tree Depth

The tree width measure generalizes the notion of trees. Robertson and Seymour [38] defined it through tree decompositions of graphs: A *tree decomposition* D of a connected undirected graph G is a tree T_D together with a labeling function $B_D\colon V(T_D)\to\mathcal{P}(V(G))$, where $\mathcal{P}(X)$ is the power set of X, that satisfies two properties: The *connectedness condition* states that for all $v\in V(G)$ the induced subtree $T_D\big[\{n\in V(T_D)\mid v\in B_D(n)\}\big]$ is nonempty and connected. The *edge cover condition* states that for every edge $e=(v,w)\in E$, there is an node $n\in V(T_D)$ with $v,w\in B_D(n)$. The sets $B_D(n)$ are called *bags*. The *width* of a tree decomposition D is $\max_{n\in V(T_D)}|B_D(n)|-1$, its *depth* is the depth of T_D. The *tree width* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width over all its tree decompositions. The language $\textsc{tree-width}=\{(G,w)\mid \mathrm{tw}(G)\leq w\}$, which is the same problem as deciding whether embeddings into k-trees exist, is NP-complete in general [6], while for any constant w, the problem $\textsc{tree-width-}w=\{G\mid \mathrm{tw}(G)\leq w\}$ is complete for L [20].

While tree width generalizes the concept of trees, tree depth is a more restrictive graph measure which generalizes the concept of star graphs. It was introduced by Nešetřil and Ossoma de Mendez [34] through closures of trees: The *closure* $\mathrm{clos}(T)$ of a directed tree T is the graph with vertex set $V(\mathrm{clos}(T))=V(T)$ and edge set $E(\mathrm{clos}(T))=\{(v,w)\in V(T)\times V(T)\mid \text{there is a } v\text{-to-}w \text{ or a } w\text{-to-}v \text{ path in } T\}$. The *tree depth* of a connected graph G, denoted by $\mathrm{td}(G)$, is 1 plus the minimum depth of a rooted tree T with $V(G)=V(T)$ and $E(G)\subseteq E(\mathrm{clos}(T))$. The tree depth of a graph with components $C_1,\ldots,C_m$ is $\max_{i\in\{1,\ldots,m\}}\mathrm{td}(C_i)$. Deciding $\textsc{tree-depth}=\{(G,d)\mid \mathrm{td}(G)\leq d\}$, which is the same problem of deciding whether a graph has an elimination tree of a given depth [34], is NP-complete [8]. At the end of Section 3.3 we show $\textsc{tree-depth-}d=\{G\mid \mathrm{td}(G)\leq d\}\in \mathrm{AC}^0$ for any constant $d\in\mathbb{N}$. In Section 3 we will make use of the following bound on the longest path length of a graph in terms of its tree depth that was observed by Nešetřil and Ossoma de Mendez [35]. We provide a proof of it for completeness.

Lemma 2.1. *For undirected graphs G we have* $\mathrm{lpl}(G)\leq 2^{\mathrm{td}(G)}-2$.

Proof. Without loss of generality let G be connected. We prove the lemma by induction over $\mathrm{td}(G)$. If $\mathrm{td}(G)=1$, then $|V(G)|=1$ and, therefore, $\mathrm{lpl}(G)=0=2^1-2$. For the case that G's tree depth is larger than 1, we consider a tree T of depth $\mathrm{td}(G)-1$ on G's vertices with $E(G)\subseteq\mathrm{clos}(T)$. Every simple path in G that does not cross the root of T has length at most $2^{\mathrm{td}(G)-1}-2$ by the induction hypothesis. Every path in G that crosses the root r of T has the form $(v_1,\ldots,v_n,r,w_1,\ldots,w_m)$, where the subpaths $(v_1,\ldots,v_n)$ and $(w_1,\ldots,w_m)$ lie completely inside subgraphs $G[V(T_1)]$ and $G[V(T_2)]$ for two subtrees T_1 and T_2 of T. Thus, the overall length of such a path is bounded by $2+2(2^{\mathrm{td}(G)-1}-2)=2^{\mathrm{td}(G)}-2$. □

The concepts of tree width and tree depth generalize from graphs to logical structures as follows: The *Gaifman graph* of a structure $\mathcal{S}$ with universe S, denoted by $G_{\mathcal{S}}$, is an undirected graph that has vertex set S and there is an edge $(a,a')\in S\times S$ if, and only if, a relation $R^{\mathcal{S}}$ from $\mathcal{S}$ contains a tuple $(a_1,\ldots,a_r)\in R^{\mathcal{S}}$ with $a,a'\in\{a_1,\ldots,a_r\}$. The *tree width* (*tree depth*) of a logical structure $\mathcal{S}$ equals the tree width (tree depth) of $G_{\mathcal{S}}$. A class of structures has *bounded tree width* (*bounded tree depth*) if the tree width (tree depth) of all its structures is bounded by a constant. Tree decompositions for logical structures can also be defined directly [22]: Here the bags are subsets of the universe of the structure and for every tuple inside any relation, there is a bag containing all the elements of the tuple. A logical structure and its Gaifman graph have exactly the same tree decompositions since tuples of r elements give rise to cliques of size r in the Gaifman graph and for every clique in the graph there is always a bag that contains this clique.

2.4 Constant-Depth and Logarithmic-Depth Circuit Classes

Circuit Classes In the present paper we are concerned with uniform circuit complexity classes that are inside deterministic logarithmic space. 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 $\{x_1,\ldots,x_n,\neg x_1,\ldots,\neg x_n\}$ 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 $\{x_1,\ldots,x_n,\neg x_1,\ldots,\neg x_n\}$ and unbounded fan-in inner gates from $\{\wedge,\vee,0,1\}$; and TC^0, whose definition is the same as that of AC^0 except that we also allow unbounded fan-in gates that decide $\textsc{majority} = \{y_1 \ldots y_n \in \{0,1\}^* \mid \sum_{i\in\{1,\ldots,n\}} y_i > n/2\}$. 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 $L \subseteq \{0,1\}^*$.

The classes AC^0 and NC^1 can be *arithmetized* by replacing, in the definition of each class, the admitted input gates by $\{x_1,\ldots,x_n,1-x_1,\ldots,1-x_n\}$ and the Boolean inner gates by arithmetic gates $\{\times,+,0,1\}$. This leads to circuit families that compute functions $f\colon \{0,1\}^* \to \mathbb{N}$; 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 [45] calls the above 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.

Uniformity We use the above complexity classes in their DLOGTIME-uniform variants, as defined by Barrington, Immerman and Straubing [33]. For constant-depth circuit families with unbounded fan-in gates, 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 [33]. 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 also include tuples describing paths between gates in the circuits, can be decided by random-access logarithmic-time alternating Turing machines [40, 12, 33].

The equality $\mathrm{PAC}^0 = \mathrm{TC}^0$ (and $\mathrm{GapAC}^0 = \mathrm{FTC}^0$) was first shown in the P-uniform setting [2, 5], and later refined to also hold in the DLOGTIME-uniformity setting [25]. The classes $\#\mathrm{NC}^1$, GapNC^1, and PNC^1 where introduced by Caussinus et al. [14] who showed $\mathrm{PNC}^1 \subseteq \mathrm{L}$ and $\mathrm{FNC}^1 \subseteq \#\mathrm{NC}^1$ (note that, in order to state inclusion relations between classes of functions that compute numbers and classes of functions that compute strings, we consider binary string representation of numbers). 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 [15] (see also [3]). The language classes that are defined via Boolean and counting arithmetic circuits are related as follows:

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

Reducibility Notions To compare the complexity of problems, we will use different uniform reducibility notions: We will use DLOGTIME-uniform AC^0 many-one and Turing reductions that are equivalent to first-order many-one and Turing reductions as studied in the book of Immerman [26]. Moreover, we will use DLOGTIME-uniform TC^0 many-one reductions. All uniform circuit classes defined above are closed under both kinds of AC^0 reductions; uniform TC^0, FTC^0, and all their superclasses defined above are closed under TC^0 many-one reductions.

For a general introduction to the field of circuit complexity, we refer to the book of Vollmer [45] and a survey article of Allender [3].

3 Algorithmic Meta Theorems For Constant-Depth Circuit Classes

In the present section we prove the algorithmic meta theorems that relate monadic second-order properties of graphs of bounded tree depth to constant-depth circuit classes (Theorems 1.1 and 1.2 from the introduction). The route toward proving them is the following:

1. In Section 3.1 we show how a tree decomposition of a logical structure of bounded tree depth can be computed using first-order reductions.
2. In Section 3.2 we show how, once the tree decomposition is available, we can adjust the original MSO-formula to an equivalent formula for the computed tree. This first step allows us to replace the task of computing solution histograms for structures of any signature by the more manageable problem of computing solution histograms for trees.
3. In Section 3.3 we introduce the notion of multiset automata for unordered unranked labeled trees, prove standard closure properties for these automata, and show that they capture exactly the MSO-definable properties of unordered unranked labeled trees. This turns the problem of deciding formulas into the problem of evaluating multiset automata.
4. In Section 3.4 we explain how to reduce computing the number of ways in which multiset tree automata accept an input tree to evaluating arithmetic circuits of constant depth.
   In the course of this last step, 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 our formulas ϕ are all of the form $\phi(X_1,\dots,X_k)$, that is, we may assume that no variables Y_i are present. This is why the lemmas and theorems of the present section are all formulated without references to any Y_j.
5. At the end, in Section 3.5, we apply the algorithmic meta theorems to concrete problems. We show, in particular, that the unary version of integer linear programming with a constant number of equations is complete for TC^0.

3.1 Computing Tree Decompositions of Bounded Width and Depth

The first step toward our goal of proving Theorems 1.1 and 1.2 is to compute tree decompositions of bounded width and depth for input structures of bounded tree depth. The following lemma states that this is possible to achieve using first-order reductions.

Lemma 3.1. *Let τ be a signature and $d \in \mathbb{N}$. There is a first-order computable function that, on input of the encoding* $\mathrm{code}(\mathcal{S})$ *of a τ-structure $\mathcal{S}$, outputs either*

1. *a tree decomposition D of $\mathcal{S}$ of width at most 2^d-3 and depth at most 2^d-1 or*
2. *"no" and* $\mathrm{td}(\mathcal{S}) > d$ *holds in this case.*

Proof. On input of a structure $\mathcal{S}$, we first build its first-order definable Gaifman graph $G_{\mathcal{S}}$. Then we use another appropriate first-order formula to test whether the graph contains a path of length 2^d-1. If there is such a path, we know $\mathrm{td}(\mathcal{S}) > d$ by Lemma 2.1 and output "no". Otherwise, the computation

proceeds to construct a closure tree of depth at most $2^d - 2$ for each component of $G_{\mathcal{S}}$. This means that for each component C, a tree T is constructed with $V(C) = V(T)$, $E(C) \subseteq \mathrm{clos}(T)$, and $\mathrm{lpl}(T) \leq \mathrm{lpl}(C) \leq 2^d - 2$. Finally, each closure tree T is turned into a tree decomposition (T, B) for C by assigning to each node $v \in V(T)$ the bag $B(v) = \{w \mid w \text{ lies on the path from } v \text{ to the root of } T\}$, and, if $G_{\mathcal{S}}$ has more than a single component, the tree decompositions for the components are merged by connecting their roots to a new root with an empty bag. The width bound $2^d - 3$ and the depth bound $2^d - 1$ of the tree decomposition follow directly from the depth bound $2^d - 2$ of the closure trees. Since T's depth is bounded, constructing the tree decomposition from the closure trees is first-order definable. We are left to describe the computation of closure trees.

We use a parallel first-order computable procedure that constructs depth-first graph search trees for connected undirected graphs of bounded longest path length. A depth-first graph search tree T for an undirected connected graph G is a closure tree of bounded depth for G since, firstly, $\mathrm{lpl}(T) \leq \mathrm{lpl}(G)$ and, secondly, each edge $(v, w) \in E(G)$ is either a tree edge (that means $(v, w) \in E(G)$) or a back-edge (that means w lies on the path from v to the root of T) of the graph search. Thus, $V(T) = V(G)$ and $E(G) \subseteq E(T)$, as was also observed by Grohe and Kreutzer [24].

We construct a depth-first graph search tree T by using $2^d - 2$ first-order computations of the same kind that are arranged in a series. The input to each *computation layer i* is a partial depth-first search tree T_i that is extended in parallel to a tree T_{i+1}. The input to the first layer is any vertex $v \in V(G)$, the output of the last layer is a depth-first search tree for the whole graph G. Each layer considers the components $C_1, \ldots, C_m$ of $G[V \setminus T_i]$ in parallel. In order to mimic the sequential computation of depth-first search trees, it finds, for each C_j, the vertex $v_j \in V(T_i)$ that is connected to some vertex from C_j in G with longest distance from the root of T_i in T_i. Then it singles out a longest simple path P_j in $G[C_j \cup \{v_j\}]$ that starts at v_j. The layer outputs the tree T_{i+1} that is the union of T_i and all paths P_j. Finally, $2^d - 2$ layers are sufficient to construct T since the lengths of the longest simple paths in the considered components decrease after each layer. □

3.2 Turning Tree-Depth-Bounded Structures into Depth-Bounded Tree Structures

In the previous section we saw that tree decompositions (T, B) of bounded width and depth can be constructed using first-order formulas for logical structures $\mathcal{S}$ of bounded tree depth. Our aim is to "work" with these tree decompositions rather than the original structures. However, the formula ϕ for which we wish to decide $\phi \models \mathcal{S}$ or to compute a histogram refers to $\mathcal{S}$, not to (T, B). Thus, our objective is to transform the formula ϕ into a new formula ψ that refers to this tree rather than the structure $\mathcal{S}$, such that $\phi \models \mathcal{S} \Leftrightarrow \psi \models (T, B)$. We must also ensure that the histograms are not modified by this transformation. Lemma 3.2 states that such a transformation is, indeed, first-order computable. For the formulation of the lemma, we use the following terminology: An *s-tree structure* is a structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ over the signature $\tau_{s\text{-tree}} = \{E^2, P_1^1, \ldots, P_s^1\}$ where $(V, E^{\mathcal{T}})$ is a directed tree.

Lemma 3.2. *Let $\phi(X_1, \ldots, X_\ell)$ be an* MSO*-formula over some signature τ and $w \in \mathbb{N}$. There is an $s \in \mathbb{N}$, and a* MSO*-formula $\psi(X_1, \ldots, X_\ell)$ over $\tau_{s\text{-tree}}$, and a first-order computable function that, on input of any τ-structure $\mathcal{S}$ with universe S and a width-w tree decomposition $D = (T_D, B_D)$ for $\mathcal{S}$, produces an s-tree structure $\mathcal{T}$, such that*

1. *the depth of $\mathcal{T}$ equals the depth of T_D plus 1, and*
2. *for all indices $i \in \{0, \ldots, |S|\}^\ell$ we have $\mathrm{hist}(\mathcal{S}, \phi)[i] = \mathrm{hist}(\mathcal{T}, \psi)[i]$ and all other entries in the array $\mathrm{hist}(\mathcal{T}, \psi)$ are 0.*

Proof. The proof is similar to the logspace-reduction we used before [20, Lemma IV.1], but avoids a recursive coloring of the elements of $\mathcal{T}$ to make the transformation first-order computable. We describe the whole reduction and highlight the steps that differ from the previous approach.

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*. In detail, the set V_B is exactly $V(T_D)$. The set V_E is the disjoint union of the sets $\{e^n_1, \ldots, e^n_{r_n}\}$ for $n \in V(T_D)$ with attached bag $B_D(n) = \{e_1, \ldots, e_{r_n}\}$, where some ordering is chosen for each bag. For an element node $x = e^n_i$, 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 $e_i \in S$. The edges of $\mathcal{T}$ are as follows: All edges of T_D are also present in $\mathcal{T}$. Additionally, for each $x \in V_E$ there is an edge from $n(x)$ to x.

The s unary predicates that are present in $\mathcal{T}$ fall into four groups:

1. *Node type predicates:* We define the predicate $P^{\mathcal{T}}_B = V_B$ and the predicate $P^{\mathcal{T}}_E = V_E$.
2. *Element ordering predicates:* We use $w+1$ predicates $P^{\mathcal{T}}_1, \ldots, P^{\mathcal{T}}_{w+1}$ to record a total ordering for the element nodes of each bag: For each bag node n with attached element nodes $\{x_1, \ldots, x_r\}$ we set $x_1 \in P^{\mathcal{T}}_{i(x_1)}, \ldots, x_r \in P^{\mathcal{T}}_{i(x_r)}$.
3. *Structure predicates:* These predicates are used to represent the relations from the structure $\mathcal{S}$ as follows: To represent a relation $R^{\mathcal{S}}$ of arity r of $\mathcal{S}$, we introduce new predicates $P^{\mathcal{T}}_{i_1,\ldots,i_r}$ for every $i_1, \ldots, i_r \in \{1, \ldots, w+1\}$. They locally encode the tuples of $R^{\mathcal{S}}$ at the bags with $(i_1, \ldots, i_r)$ being the local indices of the element of a tuple of $R^{\mathcal{T}}$. In detail, for every tuple $(x_1, \ldots, x_r) \in V^r_E$ with $n(x_1) = \cdots = n(x_r)$ and $(e(x_1), \ldots, e(x_r)) \in R^{\mathcal{S}}$, let $x_j \in P^{\mathcal{T}}_{i(x_1),\ldots,i(x_r)}$ for $j \in \{1, \ldots, r\}$. Since a tree decomposition puts the elements of a tuple completely into at least one bag, for all tuples $(e_1, \ldots, e_r) \in R^{\mathcal{S}}$ there are element nodes $x_1, \ldots, x_r$ with $n(x_1) = \cdots = n(x_r)$ and $x_1 \in P^{\mathcal{T}}_{i(x_1),\ldots,i(x_r)} \wedge x_1 \in P^{\mathcal{T}}_{i(x_1)}, \ldots, x_r \in P^{\mathcal{T}}_{i(x_1),\ldots,i(x_r)} \wedge x_r \in P^{\mathcal{T}}_{i(x_r)}$.
4. *Equivalence predicates:* These predicates are used to relate element nodes that stand for the same elements in the structure; the construction of these predicates differs from our previous method [20, Lemma IV.1]. For the logspace reduction we used a coloring that groups element nodes that stand for the same element from the structure into the same color class, such that the components of the subforests of $\mathcal{T}$ that are induced by the different color classes are in one-to-one corresponds to the elements of $\mathcal{S}$. Since assigning colors to the element nodes is not a local operation (elements of $\mathcal{S}$ may be distributed around the whole decomposition), we use a different approach to relate equivalent element nodes. We introduce $w+1$ predicates $N^{\mathcal{T}}_1, \ldots, N^{\mathcal{T}}_{w+1}$ and put an element node x into $N^{\mathcal{T}}_j$ 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 $N^{\mathcal{T}}_i$, we insert it into the predicate $P^{\mathcal{T}}_R$, the set of *representative element nodes*. By this construction, there is a one-to-one correspondence between $P^{\mathcal{T}}_R$ and the universe of $\mathcal{S}$.

All steps of the reduction are first-order computable. The formula ψ is build on top of the following two subformulas: (a) The MSO-$\tau_{s\text{-tree}}$-formula $\psi_{\text{equ}}(x,y)$ that 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 $\mathcal{S}$. (b) The MSO-$\tau_{s\text{-tree}}$-formula $\psi_R(x_1, \ldots, x_r)$, where the x_i are *first-order* variables, that is true for representative element nodes x_i if, and only if, $(e(x_1), \ldots, e(x_r)) \in R^{\mathcal{S}}$: The formula tests whether there are element nodes $y_1, \ldots, y_r$ such that $\psi_{\text{equ}}(x_1, y_1) \wedge \cdots \wedge \psi_{\text{equ}}(x_r, y_r)$, the y_i are children of the same bag node, and there is an index tuple $(i_1, \ldots, i_r)$ with $P_{i_1,\ldots,i_r}(y_1) \wedge y_1 \in P_{i_1} \wedge \cdots \wedge P_{i_1,\ldots,i_r}(y_r) \wedge y_r \in P_{i_r}$. To build ψ, we first extend the formula ϕ such that only elements and subsets of $P^{\mathcal{T}}_R$ are permissible for the free and bounded variables. Second, substitute $x = y$ by $\psi_{\text{equ}}(x,y)$. Third, substitute every $R(x_1, \ldots, x_r)$ with the formula $\psi_R(x_1, \ldots, x_r)$. □

In Section 4 we will use a reduction that is similar to the one from the previous lemma, but applies to structures that are equipped with a tree decomposition in term representations. Lemma 3.1 and Lemma 3.2 together provide a transformation from evaluating MSO-formulas on logical structures of bounded tree depth to s-tree structures of bounded depth.

3.3 Turning Formulas on Tree Structures into Tree Automata

The trees underlying the s-tree structures that are produced by Lemma 3.2 do not impose an order on sibling nodes and nodes may have an unbounded number of children. Such trees, with the s unary predicates represented by binary strings, are known as *unordered, unranked* labeled trees in the literature [29]. "Unordered" means that there is no total order on sibling nodes and "unranked" stands for unbounded degree. In this section we introduce a new notion of automata that is appropriate for unordered labeled trees and prove that it exactly captures the MSO-definable properties of unordered labeled trees, resulting in a theorem which can be seen as an extension of the Büchi–Elgot–Trakhtenbrot Theorem [10, 21, 43]. Moreover, the translation between MSO-formula and automata will preserve the sizes and number of solutions, thereby establishing a reduction from computing solution histograms for MSO-formulas on s-tree structures to evaluating tree automata.

Tree-automata-based proofs of time and space efficient variants of Courcelle's Theorem transform input structures into trees where the underlying tree has bounded degree. Then, in these proofs MSO-formulas on bounded degree trees are transformed into the classical tree automata for ranked labeled trees that were developed in the 1970's by Doner [18] and Thatcher and Wright [42]. Adopting this strategy and transforming s-tree structures with unbounded degree into tree structures 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 with constant-depth circuits. Due to this reason, we need an automaton model that does not force us to change the topology of the tree. For a similar reason, we cannot use some order on the children and translate to tree automata for ordered unranked trees that are studied in the context of XML processing [9, 23]; here the horizontal data dependencies on sibling subtrees are too high. In fact, such automata are able to decide any regular property on the ordered children of a node and, thus, cannot be simulated by constant-depth circuits.

The only automaton model from the literature that does not introduce dependencies between nodes that cannot be handled by constant-depth circuits is due to Libkin [29] who defined *counting unranked tree automata*, which are equivalent to MSO on unordered trees. The transition function 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. However, it is unclear (at least to us) how these automata could be used to compute solution histograms since we need to relate the states assigned to the subtrees of a node with the state that is assigned to the whole tree in a transparent way, without "hiding it inside a Boolean function."

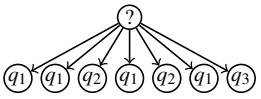
In this section, we develop multiset automata as a notion that exactly captures the MSO-definable properties of unordered labeled trees (unranked or ranked) and that allows us to control the assignment of states to the children of a node such that we can later establish a cardinality-preserving transformation into arithmetic circuits in Section 3.4.

For readers familiar with hedge automata [9] and the first-order and monoid-based characterizations of regular languages [41], we remark that the multiset automata we introduce are equivalent to bottom-up hedge automata whose horizontal languages can be recognized by finite monoids that are commutative and aperiodic.

Definition of Multiset Automata The basic idea behind our definition of a multiset 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 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 in the example to 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 would then be determined by the entry of the transition relation for this particular multiset. In the following, we formalize these ideas.



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 $\#_M \colon U \to \mathbb{N}$ 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 $\mathcal{P}_1(U)$ 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 $e \notin V$.

Definition 3.3 (Multiset Automata)**.** A *nondeterministic (bottom-up) multiset 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 *multiplicity bound* m. 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 3.4 (Computation of a Multiset Automaton)**.** Let (T, l) be a labeled tree, where $l \colon V(T) \to \Sigma$ is the labelling function, and let $A = (\Sigma, Q, Q_a, \Delta)$ be a multiset automaton. A *computation* of A on (T, l) is a partial assignment $q \colon V(T) \to Q$ such that for every node $n \in V(T)$ for which $q(n)$ is defined, we have that (a) the value $q(c)$ is defined for each child c of n in T and (b) for the mulitset $M = \{q(c) \mid c \text{ is a child of } n\}$ we have $(l(n), M|_m, q(n)) \in \Delta$. A computation is *accepting*, if $q(r) \in Q_a$ holds for the root node r of T. The *tree language* $L(A)$ contains all labeled trees accepted by A.

Closure Properties of Multiset Automata We now show that multiset automata enjoy the same closure properties as usual tree automata: The class of tree languages accepted by multiset automata is closed under intersection, union, complement, and for every nondeterministic multiset automaton there is a deterministic automaton accepting the same tree language. These closure properties will be crucial for the transformation of MSO-formulas into multiset automata.

Lemma 3.5. *For all multiset automata A and B there is a multiset automaton C with $L(C) = L(A) \cap L(B)$.*

Proof. For this proof, we introduce the following operations on multisets: For a universe U, the *projections* $\pi_1, \pi_2 \colon \mathcal{P}_\omega(U^2) \to \mathcal{P}_\omega(U)$ of multisets of pairs to their first and second components are defined as $\#_{\pi_1(M)}(e) = \sum_{f \in U} \#_M((e, f))$ and $\#_{\pi_2(M)}(f) = \sum_{e \in U} \#_M((e, f))$. Observe that if $M|_m = N|_m$ holds for two multisets M and N, then we also have $\pi_1(M)|_m = \pi_1(N)|_m$ and $\pi_2(M)|_m = \pi_2(N)|_m$.

Let $A = (\Sigma, Q, Q_a, \Delta_A)$ and $B = (\Sigma, P, P_a, \Delta_B)$ be multiset 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

$$\Delta_C = \big\{ \big(\sigma, M|_k, (q,p)\big) \mid \sigma \in \Sigma,\, q \in Q,\, p \in P,\, M \in \mathcal{P}_\omega(Q \times P),\\ (\sigma, \pi_1(M)|_m, q) \in \Delta_A,\, (\sigma, \pi_2(M)|_n, p) \in \Delta_B \big\}.$$

We show by induction on the tree depth 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 $(\sigma, \{q_1, \ldots, q_t\}|_m, q) \in \Delta_A$ and let p_1 to p_t be states reached by B at these children such that $(\sigma, \{p_1, \ldots, p_t\}|_n, p) \in \Delta_B$. By the induction hypothesis, C can reach (q_1, p_1) to (q_t, p_t) at c_1 to c_t, respectively. By definition of Δ_C, we immediately get that C can now reach (q,p) at the root.

For the if-direction, let C reach (q_1, p_1) to (q_t, p_t) at the children and let $\big(\sigma, \{(q_1,p_1), \ldots, (q_t,p_t)\}|_k, (q,p)\big) \in \Delta_C$. 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, there must be a set M with $\{(q_1,p_1), \ldots, (q_t,p_t)\}|_k = M|_k$ and $(\sigma, \pi_1(M)|_m, q) \in \Delta_A$ and $(\sigma, \pi_2(M)|_n, p) \in \Delta_B$. Now, since $\{(q_1,p_1), \ldots, (q_t,p_t)\}|_k = M|_k$, we get $\pi_1(\{(q_1,p_1), \ldots, (q_t,p_t)\})|_k = \pi_1(M)|_k$, which implies $\{q_1, \ldots, q_t\}|_k = \pi_1(M)|_k$, which in turn implies $\{q_1, \ldots, q_t\}|_m = \pi_1(M)|_m$ because of $m \leq k$. This last conclusion shows that $(\sigma, \{q_1, \ldots, q_t\}|_m, q) \in \Delta_A$ and, thus, q is reachable by A at the root as claimed. The same argument, with q's replaced by p's and π_1 replaced by π_2, shows that p is reachable by B at the root. □

Lemma 3.6. *For every nondeterministic multiset automaton A there is a deterministic multiset automaton B with $L(A) = L(B)$.*

Proof. For this proof, we introduce the following operation on multisets: For a universe U, let $P_U = \mathcal{P}_1(U)$ be the usual powerset of U. We define the *choice relation* $\iota \subseteq \mathcal{P}_\omega(U) \times \mathcal{P}_\omega(P_U)$ as follows: $(\{v_1, \ldots, v_t\}, \{V_1, \ldots, V_t\}) \in \iota$ whenever $v_i \in V_i$ holds for all $i \in \{1, \ldots, t\}$. A key observation for the following construction will be that for every m and every $W \in \mathcal{P}_\omega(P_U)$, the following set (not multiset) equality holds:

$$\big\{ V|_m \;\big|\; (V,W) \in \iota \big\} = \big\{ V|_m \;\big|\; (V, W|_{m\cdot|U|}) \in \iota \big\}. \tag{1}$$

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 (a) $(V_i, W_i) \in \iota$, (b) $V_i|_m = V|_m$, and (c) $W_i|_{m\cdot|U|} = W|_{m\cdot|U|}$. 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 $v \in V_i$ that has multiplicity exceeding m and that is an element of a set $X \in W_i$ of multiplicity exceeding $m \cdot |U|$. 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 set $V' = V_i$ constructed in this way we have $V'|_m = V|_m$ and $W_i = W|_{m\cdot|U|}$ 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 this time starting with $V_1 = V$ and $W_1 = W|_{m\cdot|U|}$. This time, seek an element v in V_i that has multiplicity at least m in V_i and a set X in W_i whose multiplicity is at least $m \cdot |U|$, but still less than $\#_W(X)$. We set $V_{i+1} = V \cup \{v\}$ and

$W_{i+1} = W_i \cup \{X\}$. Clearly, the three properties are still met by this construction. We claim that for the last $V' = V_i$ 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 automaton with multiplicity bound m. We call the elements of $P_Q = \mathcal{P}_1(Q)$ *power states.* Let $B = (\Sigma, P_Q, \{Q' \in P_Q \mid Q' \cap Q_a \neq \emptyset\}, \delta)$ with multiplicity bound $k = m \cdot |Q|$ and transition function $\delta \colon \Sigma \times \mathcal{P}_k(P) \to P$ defined by $\delta(\sigma, W) = \{q \in Q \mid \exists V \in \mathcal{P}_\omega(Q) \text{ with } (V,W) \in \iota, (\sigma, V|_m, q) \in \Delta\}$.

We prove by induction on the tree depth 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 $W = \{S_1, \ldots, S_t\}$. Now, by definition of the computation of multiset automata, the set S of states reached by A at the root is $\{q \in Q \mid \exists V \in \mathcal{P}_\omega(U) \text{ with } (V,W) \in \iota, (\sigma, V|_m, q) \in \Delta\}$. By equation (1), the set $\{V|_m \mid (V,W) \in \iota\}$ is the same as $\{V|_m \mid (V, W|_k) \in \iota\}$. Thus, we can also write S as $\{q \in Q \mid \exists V \in \mathcal{P}_\omega(U) \text{ with } (V, W|_k) \in \iota, (\sigma, V|_m, q) \in \Delta\}$, which is exactly $\delta(\sigma, W|_k)$. Hence, the set of states that are assigned to the root of the tree by nondeterministic computations of A equals the power state that is assigned to it by B. □

Lemma 3.7. *For every multiset automaton A there is a multiset automaton B accepting the complement of $L(A)$.*

Proof. Make A deterministic, if necessary, and exchange accepting and rejecting states. □

A Büchi-Elgot-Trakhtenbrot-like Theorem for Multiset Automata and Unordered Labeled Trees Given an s-tree structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ and sets $S_1, \ldots, S_\ell \subseteq V$, let us write $T(\mathcal{T}, S_1, \ldots, S_\ell)$ for the labeled tree whose node set is V, whose edge set is $E^{\mathcal{T}}$, and whose labeling function maps each node $v \in V$ to the bitstring $l_1 \ldots l_s x_1 \ldots x_\ell \in \{0,1\}^{s+\ell}$ 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 $\ell = 0$. Using multiset automata, we prove the following cardinality-preserving theorem for unordered labeled trees. We remark that for the purposes of the present paper, only the first part of the theorem would suffice.

Theorem 3.8. *Let $s, \ell \in \mathbb{N}$.*

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

Proof. Our proof follows Arnborg et al. [7], but modified to unranked trees rather than ranked trees and multiset automata rather than usual tree automata.

For the first claim of the theorem, let a formula $\phi(X_1, \ldots, X_\ell)$ be given. The first step is to transform it into an equivalent formula $\psi(X_1, \ldots, X_\ell)$ without element variables by introducing an additional set of atomic formulas. In detail, we replace every occurrence of a first-order quantifier $\exists x(\rho)$ by $\exists X(\text{singleton}(X) \wedge \rho)$. Here, X is a fresh second-order variable and singleton(X) is a new kind of atomic formula with the obvious semantics: singleton(S) holds for a set S if, and only if, S contains exactly one element. Next, every occurrence of the automatic formula $x = y$ is replaced by subset$(X,Y) \wedge$ subset(Y,X) for the second-order variables X and Y that correspond to x and y, respectively. The interpretation of the new atomic formula subset(S,T) is that it is true if, and only if, $S \subseteq T$. The atomic formula $X(y)$ is

replaced by subset(Y,X), where Y is the second-order variable corresponding to y. Finally, the atomic formula $E(x,y)$ is replaced by the new atomic formula edge(X,Y), where edge(S,T) is true if, and only if, for every $t \in T$ there is a $s \in S$ with $(s,t) \in E^{\mathcal{T}}$. The formula ψ that results from these transformations is clearly equivalent to ϕ.

The next step in the construction of [7] is to transfer the formula ψ into an automaton. This is done by induction on the structure of the formula ψ. First, assume that ψ is an automatic formula. Because of the transformations we applied, ψ must now be of one of the following forms: singleton(X), subset(X,Y), or edge(X,Y). In each case, there exists a multiset automaton accepting exactly those trees satisfying the formula: The automaton for singleton(X) needs three states q_0, q_1, and $q_{\text{too many}}$, with only q_1 being accepting, and has multiplicity 2. 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. However, when q_1 is reached at two children simultaneously or when a child is in state q_1 and the node is also an element of X, the automaton switches to the error state $q_{\text{too many}}$. Next, for the atomic formula subset(X,Y), an automaton must simply check that for all nodes whenever the node is in X, it is also in Y. Finally, the automaton for edge(X,Y) will use a state to signal each node that is an element of Y and will test whether each node is an element of X when one of its children has this state.

For the inductive step, we must now build an automaton for formulas composed using negation, conjunction, and the monadic existential quantifier. These automata can be constructed using the closure properties of multiset automata that we proved earlier in Lemmas 3.5 to 3.7: When $\phi = \alpha \wedge \beta$, we construct the intersection product automata from Lemma 3.5 of the automata for α and β. When $\phi = \neg\alpha$, we take the complement automaton from Lemma 3.7. Finally, when $\phi = \exists X(\alpha)$, we first transform the automaton for α 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 3.6.

For the second claim of the theorem, the sought MSO-formula can be obtained as follows: We use $|Q|$ many set variables $Q_1, \ldots, Q_{|Q|}$ that are quantified existentially and we require that each node n is an element of exactly one of these sets. The formula ϕ tests that the root gets assigned a state from Q_a and that, whenever $Q_q(n)$ holds for a node n, for the multiset $M = \{q' \mid \exists c(Q_{q'}(c) \wedge c \text{ is a child of } n)\}$ we have $(\sigma, M|_m, q) \in \Delta$, where σ is the label of n. To see that we can, indeed, express that the capped multiset of states assigned to the children of n is a given multiset $M|_m$ using an MSO-formula (actually, even a first-order formula suffices), it is best to give an example: Suppose that for $m = 3$ we wish to test whether $M|_3 = \{q_1, q_1, q_1, q_2, q_3, q_3, q_4, q_4, q_4\}$ holds for a node n. This is the case if, and only if, (a) there are nine distinct children of n such exactly three of them have state q_1 assigned to them, exactly one has state q_2 assigned, exactly two state q_3, and exactly three have state q_4 assigned; and (b) all children *other* than these nine children must have either state q_1 or q_4 assigned to them. □

Deciding MSO-Properties in Constant Depth Theorem 3.8 shows that in order to decide whether a given MSO-formula is true for a given tree, we can instead evaluate a multiset automaton. Since we saw earlier in this section that any logical structure of bounded tree depth can be transformed into a labeled tree of constant depth, we have all the ingredients together to prove Theorem 1.1 from the introduction:

Proof of Theorem 1.1 We first consider only the case that the input structures are guaranteed to have tree depth d and do not care what the circuits output when this is not the case. Later on, we will remedy this problem.

Let ϕ by an MSO-formula over a signature τ and let $d \in \mathbb{N}$. By Lemmas 3.1 and 3.2 and Theorem 3.8, we can turn ϕ into a fixed multiset automaton A and we can turn (using a first-order computable function) any logical structure $\mathcal{S}$ of tree depth at most d into an unordered, unranked, labeled tree (T,l) of constant depth such that $(T,l) \in L(A) \Leftrightarrow \mathcal{S} \models \phi$. Thus, all that remains to argue is that the behaviour of any fixed deterministic multiset automaton on given bounded depth labeled trees can be simulated in DLOGTIME-uniform AC^0 (or, equivalently, FO). However, this construction is essentially the same as the

one given in the proof of the second claim of Theorem 3.8: For every node of the tree, we can hard-wire the behaviour of the automaton in dependence of the states reached at the children by testing for each possible capped multiset $M|_m$ whether the states reached at the children do, indeed, form this multiset. Since the tree has constant depth, we can compute the state reached at the root in a constant number of layers.

Now we have to address the problem that input structures $\mathcal{S}$ may have tree depth larger than d. If the tree depth is very high, namely not only larger than d, but even larger than 2^d-2, this can be detected using Lemma 3.2. Thus, in a constant-depth preprocessing step, we can check whether this is the case and, if so, reject. The difficult case arises when the tree depth is between $d+1$ and 2^d-2 and it is known to be NP-complete to decide whether an input graph has tree depth d, when d is part of the input [8]. However, d is fixed and we can apply the following trick: It is known that the class of graphs of tree depth at most d is closed under taking minors and thus MSO-definable via a formula ψ that describes a finite set of forbidden minors [39]. Now, after the preprocessing that ensures that the input graph's tree depth is as most 2^d-2, we apply all of the above constructions, but not to ϕ and depth d, but rather to ψ and depth $d'=2^d-2$. This time, because of the preprocessing, we will get a definite answer to the question of whether the input graph satisfies ψ and, thus, of whether it has tree depth at most d. If the answer is negative, we also reject. □

3.4 From Automaton Evaluation to Arithmetic Circuit Evaluation

In the previous section we saw how Theorem 3.8 enables us to decide MSO-formulas on structures of bounded tree depth using constant-depth circuits. Instead of just *deciding* the formulas, in the present section we turn our attention to the more challenging problem of computing the solution histograms. Our aim will be to replace the evaluation of automata by the evaluation of convolution circuits, see Lemma 3.11, such that the circuits's outputs are the sought solution histograms. Then we reduce the evaluation of convolution circuits to the evaluation of arithmetic circuits. Since arithmetic circuits of constant depth can be evaluated in GapAC^0, we get Theorem 1.2.

From Formula Histograms to Multicoloring Histograms Theorem 3.8 establishes a link between formulas and multiset 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 formulas and labelings of the trees that make the automaton accept. In order to talk more easily about the number of such labelings, we recall the notion of *multicolorings* from [20].

An $[r]^\ell$*-array* is an ℓ-dimensional array of integers where all indices $i=(i_1,\dots,i_\ell)$ are elements of the index set $[r]^\ell=\{0,\dots,r-1\}^\ell$. We call r the *range*. Given a set S, a *multicoloring* of S is a tuple $(S_1,\dots,S_\ell)$ of subsets $S_j\subseteq S$ for $j\in\{1,\dots,\ell\}$. Given a set X of multicolorings of S, let $\mathrm{hist}(X)$ denote the $[|S|+1]^\ell$-array whose entry at index $i=(i_1,\dots,i_\ell)$ is the number of multicolorings $(S_1,\dots,S_\ell)\in X$ with $|S_1|=i_1$, …, $|S_\ell|=i_\ell$. For instance, for $S=\{1,2\}$ and $X=\{(\{1\},\{1\}),(\{2\},\{2\}),(\{2\},\emptyset)\}$, we have $\mathrm{hist}(X)=\left(\begin{smallmatrix}0&0&0\\1&2&0\\0&0&0\end{smallmatrix}\right)$.

The connection between multicolorings and tree automata is as follows: Given a multiset automaton $A=(\{0,1\}^{s+\ell},Q,Q_a,\Delta)$ and an s-tree structure $\mathcal{T}$ with universe V, let us write $S_A(\mathcal{T},P)$ for the set of tuples $(S_1,\dots,S_\ell)$ with $S_i\subseteq V$ for which A reaches a state $q\in P$ at the root of $T(\mathcal{T},S_1,\dots,S_\ell)$. Clearly, $S_A(\mathcal{T},P)$ is a set of multicolorings of V. In particular, for the automaton A constructed in Theorem 3.8 for a formula ϕ we have $\mathrm{hist}(\mathcal{T},\phi)=\mathrm{hist}(S_A(\mathcal{T},Q_a))$. This means that "all" we have to do in the following is to devise a way of computing $\mathrm{hist}(S_A(\mathcal{T},Q_a))$ for a given automaton A and a tree $\mathcal{T}$.

Before we proceed, it will be useful to recall some simple operations from [20] 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 A, we have $\mathrm{hist}(X_1\cup X_2)=\mathrm{hist}(X_1)+\mathrm{hist}(X_2)$ where the addition of arrays

is just the component-wise addition. Second, given two disjoint sets A and B and sets of multicolorings X and Y of A and B, respectively, let us write $X \otimes Y$ for the set of multicolorings $\{(A_1 \cup B_1, \ldots, A_\ell \cup B_\ell) \mid (A_1, \ldots, A_\ell) \in X, (B_1, \ldots, B_\ell) \in Y\}$. This operation combines two multicolorings of different parts of a tree into a single one "in all possible ways." To understand its effect, consider the case where $\ell = 1$. Then $X \otimes Y = \{A \cup B \mid A \in X, B \in Y\}$ and consider, say, $\mathrm{hist}(X \otimes Y)[3]$. This is number of ways we can choose sets $A \in X$ and $B \in Y$ with $|A \cup B| = 3$. It is not hard to see that this is exactly equal to $\mathrm{hist}(X)[0] \cdot \mathrm{hist}(Y)[3] + \mathrm{hist}(X)[1] \cdot \mathrm{hist}(Y)[2] + \mathrm{hist}(X)[2] \cdot \mathrm{hist}(Y)[1] + \mathrm{hist}(X)[3] \cdot \mathrm{hist}(Y)[0]$. This sum is also known as the *convolution* of the two histogram arrays at position 3. In general, given two arrays A and B with the same dimension ℓ and ranges r and s, respectively, their *convolution* is the $[r+s-1]^\ell$-array $C = A * B$ with

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

With these definitions, $\mathrm{hist}(A \otimes B) = \mathrm{hist}(A) * \mathrm{hist}(B)$.

Definition of Convolution Circuits Our ultimate goal is to prove Theorem 1.2, which states that we can compute (number encodings of) solution histograms using constant-depth 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). The gates of these circuits will be addition ("+"-gates) and convolution gates ("$*$"-gates). Addition gates allow us to unite histograms that arise from disjoint sets of solutions: For instance, suppose that for some deterministic multiset automaton M we have found a way to compute the histograms $h_q = \mathrm{hist}(S_M(\mathcal{T}, q))$ for all states $q \in Q_a$. Then the sum over all of these histograms will be the histogram of all multicolorings that make M 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 $S_M(\mathcal{T}, q)$ is exactly $S_M(\mathcal{T}_1, q_1) \otimes S_M(\mathcal{T}_2, q_2)$, 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 M reach q_1 can be combined with every multicoloring of $\mathcal{T}_2$ that makes M reach q_2. But then, as we saw above, the desired solution histogram $\mathrm{hist}(S_M(\mathcal{T}, q))$ is given by the convolution of the subtrees histograms.

The formal definition of a convolution circuit is now straightforward:

Definition 3.9 (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 there children are ordered. Constant gates can be labeled with arbitrary arrays. A convolution circuit without subtraction gates will be called *positive*.

In slight abuse of notation, when describing the structure of circuits, we will sometimes just write down formulas involving addition and convolution operators. For instance, $A * B + C$ denotes the circuit starting with an addition gate at the top, one convolution gate as a child, and the three leaves A, B, and C. When a gate has many predecessors P_1, …, P_n, we use the notation $\sum_i P_i$ for addition gates and $\prod_i P_i$ for convolution gates.

Definition 3.10 (Computation of a Convolution Circuit)**.** The input for a convolution circuit C with n inputs is a sequence $(A_1, \ldots, A_n)$ of arrays. The output $\mathrm{val}(g, C[A_1, \ldots, A_n])$ of a gate g is the component-wise addition, component-wise difference, or convolution of the arrays that are output by the gate's predecessors. For the ith input gate, $\mathrm{val}(g, C[A_1, \ldots, A_n])$ is A_i; while for constant gates its value is the constant attached to it. The array produced at the output gate will be denoted $\mathrm{val}(C[A_1, \ldots, A_n])$ or, if there are no input gates, just $\mathrm{val}(C)$.

Turning Automata Computations Into Convolution Circuits We are now ready to prove a lemma that shows how the histograms of multiset automata on labeled trees can be computed using convolution circuits.

Lemma 3.11. *Let $A = (\{0,1\}^{s+\ell}, Q, Q_a, \delta)$ be a deterministic multiset automaton with multiplicity bound $m \in \mathbb{N}$. Then there is a first-order computable function that maps every s-tree structure $\mathcal{T} = (V, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ to a convolution circuit C such that*

1. $\mathrm{val}(C) = \mathrm{hist}(S_A(\mathcal{T}, Q_a))$,
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 bounded by m, then C is positive.

Before we prove the lemma, let us try to get some intuition first. The easy case arises when $\mathcal{T}$ is a binary tree, which is the situation that we studied in a previous paper [20]. Here, the construction of C roughly works as follows: We first replace each node n of $\mathcal{T}$ by $|Q|$ gates g_q^n, one for each state $q \in Q$. Let $\mathcal{T}_n$ be the tree rooted at n. The objective is to have $\mathrm{val}(g_q^n, C) = \mathrm{hist}(S_A(\mathcal{T}_n, \{q\}))$, that is, the gate g_q^n should tell us how many multicolorings of the tree $\mathcal{T}_n$ cause A to reach state q at node n. To achieve this, g_q^n is an addition gate that combines the results of several new gates, namely one new gate for each pair (q_1, q_2) such that A will switch to state q when it reaches the states q_1 and q_2 at the children c_1 and c_2 of n. Now, all of these new gates will be convolution gates that combine the histograms coming from their respective gates $g_{q_1}^{c_1}$ and $g_{q_2}^{c_2}$. (The actual construction is slightly more involved, since we also have to take the coloring of the node n into account.) When we construct the whole circuit C in this way, for the root node r we have one gate g_q^r for each state that tells us exactly the histogram of the set of multicolorings making the automaton reach state q at the root. Thus, by adding one addition gate at the top that sums up over all the outputs of all g_q^r with $q \in Q_a$, we get the desired circuit. Note that the circuit is positive.

It is straightforward to extend the above construction to any tree of bounded degree, but for trees of unbounded degree, complications arise. To keep things simple, consider an automaton with multiplicity $m = 1$ and $Q = \{q_1, q_2\}$ and assume that the root has, say, ten children c_1 to c_{10} at which the subtrees $\mathcal{T}_1$ to $\mathcal{T}_{10}$ are rooted. As above, assume that for each child c_i we have two gates g_1^i and g_2^i setup in such a way that $\mathrm{val}(g_1^i, C) = \mathrm{hist}(S_A(\mathcal{T}_i, q_1))$ and $\mathrm{val}(g_2^i, C) = \mathrm{hist}(S_A(\mathcal{T}_i, q_2))$. The question is how we can combine these twenty histograms to a histogram for the whole tree.

Let us start with the easy cases: What is the histogram of all multicolorings that make A reach q_1 at *all* c_i? Clearly, this is given by $h_1 = \mathrm{val}(g_1^1, C) * \mathrm{val}(g_1^2, C) * \cdots * \mathrm{val}(g_1^{10}, C)$ since this counts exactly the number of ways in which the multicolorings of the different trees can be combined. Similarly, the histogram of all multicolorings where q_2 is reached at all c_i is $h_2 = \mathrm{val}(g_2^1, C) * \cdots * \mathrm{val}(g_2^{10}, C)$. Finally, the histogram of all multicolorings where q_1 or q_2 is reached at all c_i is $h = (\mathrm{val}(g_1^1, C) + \mathrm{val}(g_2^1, C)) * \cdots * (\mathrm{val}(g_1^{10}, C) + \mathrm{val}(g_2^{10}, C))$.

The harder case is to compute the histograms of all multicolorings such that for the multiset M of states reached by A at c_1 to c_{10} we have $M|_1 = \{q_1, q_2\}$. These multicolorings must cause the automaton to reach q_1 at some children and q_2 at some (other) children. The trick is to see that is the same as the set of multicolorings where the automaton reaches q_1 or q_2 at each child, minus the multicolorings where it reaches *only* q_1 and also minus the multicolorings where it reaches *only* q_2. This means that the sought histogram is given exactly by $h - h_1 - h_2$.

In general, in order to compute the histogram of all multicolorings making the automaton reach a set of states whose capped version equals some given multiset, we first compute the histogram of all multicolorings making the automaton reach any of the capped states in the multiset and then subtract appropriate histograms where only some of the capped states are reached exclusively. To compute these, in turn, we may need to apply a similar construction, leading to a slightly involved recursive definition whose details are given in the following proof of Lemma 3.11.

Proof of Lemma 3.11 Let $A = (\{0,1\}^{s+\ell}, Q_a, \delta)$ 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, let us introduce some new terminology. 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 sets 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 of 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 $C_{\mathrm{trans}}(\mathcal{T}', P)$ have the form
$$\sum_{\substack{x \in \{0,1\}^\ell,\ M \in \mathcal{P}_m(Q) \text{ s.t.} \\ \delta(zx, M) \in P}} C_{\mathrm{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M) * \chi(x),$$
where $\chi(x)$ is the ℓ-dimensional array with a 1-entry at position x and 0-entries at all other positions. The $C_{\mathrm{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M)$ are the multiset circuits defined next. Note that each C_{trans} has depth 2.
2. Let $\mathcal{T}_1, \dots, \mathcal{T}_n$ be the child subtrees of $\mathcal{T}'$ and let $M \in \mathcal{P}_m(Q)$. Then the circuit $C_{\mathrm{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M)$ is of the form $C_{\mathrm{uncap}} - C_{\mathrm{correct}}$, where C_{uncap} is what we call the *uncapped circuit* and C_{correct} is what will be called 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 $f\colon \{1, \dots, n\} \to \{\{q\} \mid q \in \mathrm{rare}(M)\} \cup \{\mathrm{plenty}(M)\}$ such that $M|_{\mathrm{rare}(M)} = \{i \in \{1, \dots, n\} \mid f(i) \neq \mathrm{plenty}(M)\}|_{\mathrm{rare}(M)}$ (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, including zero) and each subcircuit has the form $\prod_{i=1}^n C_{\mathrm{trans}}(\mathcal{T}_i, f(i))$.
The correction circuit also starts with an addition gate. This gate is directly connected to all circuits $C_{\mathrm{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, N)$ where $N \subsetneq M$ and $M|_{\mathrm{rare}(M)} = N|_{\mathrm{rare}(M)}$. (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. Also note that if the number n of children is bounded by the multiplicity m, there is no $N \subsetneq M$ with the property $M|_{\mathrm{rare}(M)} = N|_{\mathrm{rare}(M)}$ and the correction circuit is empty and can be left out.

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

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

Claim 1. *For each $P \subseteq Q$ we have* $\mathrm{val}(C_{\mathrm{trans}}(\mathcal{T}', P)) = \mathrm{hist}(S_A(\mathcal{T}', P))$.

Claim 2. *For each $M \in \mathcal{P}_m(Q)$ we have*
$$\mathrm{val}(C_{\mathrm{multi}}(\mathcal{T}_1, \dots, \mathcal{T}_n, M)) = \sum_{\substack{q_1, \dots, q_n \in Q \text{ s.t.} \\ \{q_1, \dots, q_n\}|_m = M}} \prod_{i=1}^n \mathrm{val}(C_{\mathrm{trans}}(\mathcal{T}_i, q_i)).$$

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} referred to the circuits $C_{\text{multi}}(\mathcal{T}_1,\ldots,\mathcal{T}_n,N)$ 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, the array $\text{val}(C_{\text{multi}}(\mathcal{T}_1,\ldots,\mathcal{T}_n,M))$ 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 $(q_1,\ldots,q_n)$ of states $q_i \in Q$: Let us say that the sequence is *perfect* if $\{q_1,\ldots,q_n\}|_m = M$. Let us call it *good* if $\{q_1,\ldots,q_n\}|_{\text{rare}(M)} = M|_{\text{rare}(M)}$. 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 left hand side of the following equation and we claim that it can be rewritten as the right hand side:

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

To prove this equality, first fix a function f. Let us say that a sequence $(q_1,\ldots,q_n)$ 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 $\prod_{i=1}^{n} \text{val}(C_{\text{trans}}(\mathcal{T}_i,f(i)))$ as follows:

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

Since every sequence $(q_1,\ldots,q_n)$ is good for exactly one f and since $\{q_1,\ldots,q_n\}|_{\text{rare}(M)} = \{i \in \{1,\ldots,n\} \mid f(i) \neq \text{plenty}(M)\}|_{\text{rare}(M)}$ 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:

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

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

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

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 Degree: 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.

As argued earlier, if the number of children of each node is bounded by m, no correction circuits are needed and, hence, the circuit is positive. □

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 using a single number $\mathrm{num}(h) \in \mathbb{N}$, imagine h to be stored in computer memory. Then $\mathrm{num}_b(h) = \sum_{i \in \{0,\dots,|S|\}} h[i] b^i$, where $\log_2 b$ is the "word size" of the memory, is a single number that represents the whole of the memory contents. In particular, for sufficiently large b, the bit representation of each $h[i]$ can be retrieved easily from the bit representation of $\mathrm{num}_b(h)$. For multidimensional histograms, use a vector $b = (b_1, \dots, b_\ell)$ of bases and set $\mathrm{num}_b(h) = \sum_{(i_1,\dots,i_\ell) \in \{0,\dots,|S|\}^\ell} h[i_1, \dots, i_\ell] b_1^{i_1} \cdots b_\ell^{i_\ell}$.

By choosing ever larger bases b_i, namely $b_1 = 2^{|S|}$, $b_2 = 2^{|S|^2}$, $b_3 = 2^{|S|^3}$, all histogram entries can be retrieved from the bits of $\mathrm{num}_b(h)$. However, we can also set a b_i to just 1. If we do so for all b_i, then the number $\mathrm{num}_b(h)$ is just the sum over all entries of the histogram, and setting the last k many b_i to 1 while setting the first ℓ many to ever larger values, we get a number that encodes for each index into the first ℓ dimensions of the histogram the sum over all entries for the last k dimensions. Thus, in order to prove Theorem 1.2, it suffices to prove the following theorem:

Theorem 3.12. *For every* MSO*-formula $\phi(X_1, \dots, X_\ell)$ over some signature τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* GapAC^0*-circuit family that, on input of a τ-structure $\mathcal{S}$ of tree depth at most d and a vector $b \in \mathbb{N}^\ell$ of bases, outputs* $\mathrm{num}_b(\mathrm{hist}(\mathcal{S}, \phi))$.

Proof. Clearly, given two arrays A and B, we have $\mathrm{num}_b(A+B) = \mathrm{num}_b(A) + \mathrm{num}_b(B)$ and also $\mathrm{num}_b(A - B) = \mathrm{num}_b(A) - \mathrm{num}_b(B)$. More importantly, we also have $\mathrm{num}_b(A * B) = \mathrm{num}_b(A) \cdot \mathrm{num}_b(B)$ as the following computation shows:

$$\begin{aligned}
\mathrm{num}_b(A * B) &= \textstyle\sum_{k_1,\dots,k_\ell \in [r]} (A * B)[k_1, \dots, k_\ell] b_1^{k_1} \cdots b_\ell^{k_\ell} \\
&= \textstyle\sum_{k_1,\dots,k_\ell \in [r]} \quad \sum_{i_1 + j_1 = k_1, \dots, i_\ell + j_\ell = k_\ell} A[i_1, \dots, i_\ell] B[j_1, \dots, j_\ell] b_1^{k_1} \cdots b_\ell^{k_\ell} \\
&= \textstyle\sum_{k_1,\dots,k_\ell \in [r]} \quad \sum_{i_1 + j_1 = k_1, \dots, i_\ell + j_\ell = k_\ell} A[i_1, \dots, i_\ell] B[j_1, \dots, j_\ell] b_1^{i_1 + j_1} \cdots b_\ell^{i_\ell + j_\ell} \\
&= \textstyle\sum_{i_1,\dots,i_\ell \in [r]} A[i_1, \dots, i_\ell] b_1^{i_1} \cdots b_\ell^{i_\ell} \sum_{j_1,\dots,j_\ell \in [r]} B[j_1, \dots, j_\ell] b_1^{j_1} \cdots b_\ell^{j_\ell} \\
&= \mathrm{num}_b(A) \cdot \mathrm{num}_b(B).
\end{aligned}$$

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

Claim. *There is a first-order computable function that gets a convolution circuit C with n input gates as input and outputs an arithmetic circuit A with $n+\ell$ input gates such for all $b=(b_1,\dots,b_\ell)$ the following holds:*

$$\mathrm{num}_b\big(\mathrm{val}(C[R_1,\dots,R_n])\big) = \mathrm{val}(A[\mathrm{num}_b(R_1),\dots,\mathrm{num}_b(R_n),b_1,\dots,b_\ell]).$$

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

Proof. The circuit A 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 the arithmetic circuit evaluating the formula

$$\mathrm{num}_b(X) = \textstyle\sum_{k_1,\dots,k_\ell\in[r]} X[k_1,\dots,k_\ell] b_1^{k_1}\cdots b_\ell^{k_\ell}.$$

The above circuit has constant depth and each number $X[k_1,\dots,k_\ell]$ can easily be expressed in constant depth and size logarithmic in its value. □

Returning to the proof of Theorem 3.12, let us recapitulate the sequence of transformations introduced in this section: Starting with an MSO-formula ϕ over a signature τ and $d\in\mathbb{N}$, by Lemmas 3.1 and 3.2 and Theorem 3.8 we can turn ϕ into a fixed multiset automaton A and we can turn (using a first-order reduction) any logical structure $\mathcal{S}$ of tree depth at most d into an s-tree structure $\mathcal{T}$ of constant depth such that $\mathrm{hist}(\mathcal{S},\phi) = \mathrm{hist}(S_A(T,Q_a))$. By Lemma 3.11 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, the range of all constants in the circuit is $[2]^\ell = \{0,1\}^\ell$. By the above claim, we can turn the convolution circuit into an arithmetic circuit that takes bases as (additional and only) inputs. Since all constants in the convolution circuit have constant range, the resulting arithmetic circuit has constant depth. In particular, it can be evaluated in GapAC^0 as claimed. □

3.5 Application: Placing Problems in Constant-Depth Circuit Classes

The algorithmic meta theorems developed in this section allow putting problems into the uniform circuit classes AC^0, GapAC^0 and TC^0 by using direct MSO-based definitions of problems on structures of bounded tree depth or reductions to MSO-definable problems on bounded tree depth structures.

We start by considering problems that are MSO-definable. By restricting them to input structures of bounded tree depth we can apply the meta theorems from this section. We saw already at the end of the proof of Theorem 1.1, that for every d the language TREE-DEPTH-d lies in DLOGTIME-uniform AC^0 and the argument relied heavily on the fact that this language is MSO-definable. Three problems that nicely exemplify the use of Theorems 1.1 and 1.2 and Corollary 1.3 in a unified way are related to perfect matchings in graphs: In the introduction we mentioned that Theorems 1.1 and 1.2 can be used to show that the problems of deciding whether a graph of bounded tree depth has a perfect matching and counting the number of theses matchings lie in DLOGTIME-uniform AC^0 and GapAC^0, respectively, via a MSO formula $\phi_{\text{matching}}(Y)$ that defines perfect matchings on structures encoding the incidence matrix of graphs G. That means $\phi_{\text{matching}}(Y)$ is true for sets $E'\subseteq E(G)$ that are perfect matchings.

Counting the number of perfect matchings in #P-complete in general [44] and in FL for graphs of bounded tree width [20]. Das, Datta, and Nimbhorkar [17] have pointed out that the problem of deciding whether a graph has a perfect matching is hard for L, implying that it is L-complete. By using the formula $\phi_{\text{matching}}(Y)$ and Corollary 1.3 we can show that the optimization problem of deciding whether the number of perfect matchings in a graph of bounded tree depth exceeds a given threshold value is in TC^0. In fact, this problem is complete for TC^0:

Theorem 3.13. *For every $d \in \mathbb{N}$, the language $\{(G,s) \mid G$ has tree depth at most d and at least s perfect matchings$\}$ is* TC^0*-complete under* AC^0*-reductions.*

Proof. We are only left to prove TC^0-hardness under AC^0-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 G that is the union of n components $G_1, \dots, G_n$. Each component G_i has four vertices. We connect them to form the cycle  if $x_i = 1$ and, otherwise, we connect them as follows:  . It is an easy observation that the first graph has two perfect matchings while the second graph has one. Thus, since the graphs G_i are not connected, at least $\lfloor n/2 \rfloor + 1$ entries in $x_1 \dots x_n$ are 1 exactly if the number of perfect matchings of G is at least $2^{\lfloor n/2 \rfloor + 1}$. □

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. An important example are problems whose inputs are sequence of numbers and we ask whether we can add up the numbers in a certain way. In the introduction we saw that MAJORITY can easily be reduced to a problem that fits to Corollary 1.3. Similar arguments work for the unary version of the subset sum problem: UNARY-SUBSETSUM $= \{1^{a_1}\#\dots\#1^{a_n}\#\#1^s \mid \exists I \subseteq \{1,\dots,n\}$ with $\sum_{i \in I} a_i = s\}$. The quest of resolving the complexity of this problems has a long history, see [20] for a discussion. Recently it was shown to lie in L by us [20] and, independently, by Kane [27]. For our proof of UNARY-SUBSETSUM $\in$ L we mapped UNARY-SUBSETSUM instances $1^{a_1}\#\dots\#1^{a_n}\#\#1^b$ to a forest $\mathcal{F}$ consisting of n stars where the ith star has a_i vertices and use an MSO-formula $\phi(X)$ that forces solution sets $S \subseteq V(\mathcal{F})$ to cover each star either completely or not at all. Since the reduction is easily accomplished in FO and the forest has tree depth 2, we can apply Corollary 1.3 to get UNARY-SUBSETSUM $\in \mathrm{TC}^0$ since $1^{a_1}\#\dots\#1^{a_n}\#\#1^b \in$ UNARY-SUBSETSUM exactly if $\mathrm{hist}(\mathcal{F},\phi)[b] > 0$. It is clear that UNARY-SUBSETSUM is also hard for TC^0: 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 $\{0,1\}$-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 values that are assigned to the variables are bounded by some threshold: For example, SUBSETSUM is the problem of deciding, for a given vector $a \in \mathbb{N}^n$ and a target value $b \in \mathbb{N}$ whether there exists a vector $s \in \{0,1\}^n$ with $a^T s = b$. The application of Corollary 1.3 to the unary version of SUBSETSUM can be adjusted to solve systems of a constant number of linear equations with integer coefficients 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-PROGRAMMING, is a linear equation system with integer coefficients $(A,b) \in \mathbb{Z}^{\ell \times m}$ and an upper bound $t \in \mathbb{N}$. The problem asks whether there is an $s \in \{0,\dots,t\}^m$ with $As = b$. 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, in [37]). Actually, its unary version is TC^0-complete:

Theorem 3.14. *For each $\ell \in \mathbb{N}$, ℓ-*INTEGER-LINEAR-PROGRAMMING *where input numbers are given in unary is complete for* TC^0.

Proof. We show how to reduce the unary version of ℓ-INTEGER-LINEAR-PROGRAMMING to an MSO-definable problem on structures of tree depth 4 that can be solved in TC^0 by applying Corollary 1.3. We first discuss the case that the coefficients of the equation system are from $\mathbb{N}$ and later extend this to coefficients from $\mathbb{Z}$.

Given an equation system $(A,b) \in \mathbb{N}^{\ell \times m} \times \mathbb{N}^\ell$ and a bound $t \in \mathbb{N}$, we build a forest $\mathcal{F}$ of m trees $\mathcal{T}_1, \dots, \mathcal{T}_m$. Each tree T_i has a root r_i to which we attach $t+1$ child nodes $v_{i,0}, \dots, v_{i,t}$. 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 $a_{i,j,1}, \dots, a_{i,j,\ell}$. We establish k unary predicates $P_1^{\mathcal{F}}, \dots, P_\ell^{\mathcal{F}}$ 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 $\phi(X_1,\dots,X_\ell)$ whose solutions $S_1, \dots, S_\ell$ must satisfy the following properties: There exists a set of vertices $\{v_{1,j_1},\dots,v_{m,j_m}\}$ (that means, a value for

each variable), such that for each $k \in \{1,\dots,\ell\}$ (each equation with index k), exactly the leaves below the nodes $a_{i,j_1,k}$, …, $a_{m,j_m,k}$ 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 to the variables. Thus, $\text{hist}(\mathcal{F},\phi)[b] > 0$ exactly if $(A,b) \in$ ℓ-INTEGER-LINEAR-PROGRAMMING.

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 $P_+^{\mathcal{F}}$ an $P_-^{\mathcal{F}}$ that are used to label the $a_{i,j,k}$ nodes and 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 $j \in \{1,\dots,t\}$, and $a_{i,j,k} \in P_-^{\mathcal{F}}$ otherwise. We extend the previous formula to the formula $\phi(X_{1,+},\dots,X_{\ell,+},X_{1,-},\dots,X_{\ell,-})$ 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 $i = i_1,\dots,i_\ell,i_{\ell+1},\dots,i_{2\ell}$ with $\text{hist}(\mathcal{F},\phi)[i] > 0$ and $b[1] = i_1 - i_{\ell+1}$, …, $b[\ell] = i_\ell - i_{2\ell}$.

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

Papadimitriou [36] 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 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 an input with this bound and applying the previous theorem, we can solve this problem in TC^0.

4 Algorithmic Meta Theorems For Logarithmic-Depth Circuit Classes

In the present section we prove the Theorems 1.4 and 1.5, which involve circuits of *logarithmic* depth rather than constant depth as in the previous section. The inputs now consist of (an encoding of) a logical structures $\mathcal{S}$ together with a tree decomposition D of $\mathcal{S}$, where T_D is given in term representation. The proofs follow along the same lines as those of Theorems 1.1 and 1.2, which involved the following steps:

1. Compute a tree decomposition of the input structure.
2. Move from formulas on the input structures to formulas on trees.
3. Move from formulas on trees to the evaluation of tree automata.
4. Move from the evaluation of tree automata to convolution circuits.
5. Move from convolution circuits to 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. All of the other steps are also possible when the tree depth is no longer constant, the resulting circuits then simply have arbitrary depth. Since it is known that tree automata can be evaluated in NC^1 on trees given in term representation [30, 23], Theorem 1.4 follows (almost) immediately from our previous arguments, see Section 4.2 for the proof details.

The main obstacle in proving Theorem 1.5 is that one can evaluate arithmetic *formulas* of arbitrary depth in $\#\text{NC}^1$ [11, 14], but evaluating arithmetic *circuits* can be done in $\#\text{NC}^1$ only if the circuit has logarithmic depth (evaluating arithmetic circuits of arbitrary depth is already FP-hard when we cap the numbers to enforce the outputs to have only polynomial length, which they need not have in general). This means that, at some point in the course of the proof of Theorem 1.5, we need to move from trees or circuits of arbitrary depth to logarithmic depth. Previous papers, such as [28], have faced a similar obstacle, namely evaluating tree-like structures of arbitrary depth whose nodes perform a complicated algebraic operation on the values of their children. In these papers, the approach was to somehow extend the ideas used in the proof that evaluating arithmetic formulas can be done in $\#\text{NC}^1$ [11, 14] to more general algebraic structures.

Our approach to tackling this problem is different and may be of independent interest. Rather than trying to adapt algorithms to the convolution computations that would be needed in our setting, we attack the problem at a much earlier stage: We balance the tree decomposition. Since all of our later algorithms do not increase the depth of the considered trees, we get the desired arithmetic circuits of logarithmic depth. In detail, we show how a balanced width-3 tree decomposition of an arbitrary tree can be computed using constant-depth threshold circuits. The construction has two key properties. First, it is based on the classical tree contraction method, which is used a lot in the context of parallel random access machines, but which hitherto was not used in the context of NC^1. Using it will allow us to compute a balanced tree decomposition even in TC^0 and not only in NC^1. Second, the tree decomposition we compute does *not* have the property that the nodes of each bag form a balanced separator of some part of the tree. Normally, recursive NC^1 algorithms find sets of nodes that in each step split up the tree into components that are smaller than the current tree by a certain factor. This is not the case for the sets of nodes in our bags: While we *can* ensure that the whole tree is balanced and, hence, has logarithmic depth, we *cannot* ensure that the elements of any individual bag split the tree in some balanced way. Naturally, a lot of bags will have this balancing property (otherwise no tree decomposition of logarithmic depth would result), but we cannot say anything about where these balancing bags will lie. It seems that this more global approach (just find a tree decomposition of logarithmic depth) instead of the traditional local approach (find a balancing separator for each subtree recursively) allows us to lower the circuit complexity to a constant depth.

In Section 4.1 we first review term representations, then we prove Theorem 1.4 in Section 4.2. In Section 4.3 we prove the technical result on how a width-3 tree decomposition of any tree can be computed in FTC^0; and then use this result in Section 4.4 to prove Theorem 1.5. In Section 4.5 we sketch applications of the established meta theorems.

4.1 Term and Ancestor Representations of Trees

Up to now, the details of how tree decompositions are encoded as strings was not important; indeed, in the context of constant tree *depth* almost any encoding of the input graph and of tree decompositions will do 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 (for directed trees, this is exactly the transitive closure) 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. In the following, let T be a (rooted, directed) tree.

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 $T_1, \ldots, T_k$ (in some arbitrary order) and if t_1 to t_k are term representations of T_1 to T_k, respectively, then $[t_1 \ldots t_k]$ is a term representation of T. For instance, the tree  has the two term presentations [[ ] [[ ] [ ]]] and [[[ ] [ ]] [ ]].

The set of *ancestor representations* of a tree T whose nodes are bitstrings is a string s 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 example tree  , with the nodes being 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 representation, 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. Let $R\colon V(T) \to \{1,\dots,q\}$ map each node of the tree to 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: 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 u and v. These observations imply the following lemma:

Lemma 4.1. *There is a* DLOGTIME*-uniform* TC^0*-family that maps every term representation of a tree T to an ancestor representation of T (for an appropriate naming of the nodes of T).*

It is also possible to convert ancestor representation to term representations:

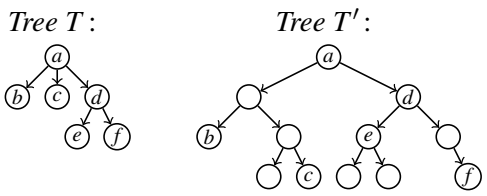
Lemma 4.2. *There is a* DLOGTIME*-uniform* TC^0*-family that maps every ancestor representation of a tree T to a term representation of T.*

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, …, 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 a TC^0-circuit, 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 $L(n)+2d_n+1$. □

Balancing Trees of Bounded Degree and Logarithmic Depth Ancestor representations make a number of modifications easy. In particular, we can use them to balance trees of logarithmic depth. For this, we recall a definition from [20]: 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 right, where 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 4.3. *There is a* DLOGTIME*-uniform* TC^0*-family 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 in T' every inner node has exactly two children.*

Proof. For every node of n 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. Note that these new nodes inherit their ancestors and descendants (other than n and its children) from the original node n. For nodes of degree 1, just add a second child that is white. Again, the ancestor relation is inherited. □

Lemma 4.4. *For every c, there is a* DLOGTIME*-uniform* TC^0*-family 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 balanced (that is, every inner node has degree 2 and all leaves are on the same level).*

Proof. First, using Lemma 4.3, we may assume that the degree of all inner nodes of T is 2. We compute the depth d of the tree. Now, for every leaf l of T at a depth $d' < d$, we add a new balanced binary tree of depth $d-d'$ 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. □

4.2 Decision by Logarithmic-Depth Circuits for Term Representations

We are now ready to prove Theorem 1.4 from the introduction. It states, that every MSO-property of graphs of bounded tree width can be decided in NC^1, provided a tree decomposition is given in term representation. The proof follows the exact same lines as for Theorem 1.1, except that while we no longer need to compute a tree decomposition (it is part of the input, after all), we now have to make sure that our transformations of the input still yield a term representation of the resulting trees. It turns out that the crucial point is to show the following modified version of Lemma 3.2:

Lemma 4.5. *Let $\phi(X_1,\dots,X_\ell)$ be an* MSO*-formula over some signature τ and $w \in \mathbb{N}$. There is an $s \in \mathbb{N}$, and a* MSO*-formula $\psi(X_1,\dots,X_\ell)$ over $\tau_{s\text{-tree}}$, and a* DLOGTIME*-uniform* FTC^0*-circuit family that, on input of any τ-structure $\mathcal{S}$ with universe S and a width-w tree decomposition $D = (T_D, B_D)$ for $\mathcal{S}$ and a term representation of T_D, produces an s-tree structure $\mathcal{T}$, whose underlying tree is T, and a term representation of T, such that for all indices $i \in \{0,\dots,|S|\}^\ell$ we have* $\mathrm{hist}(\mathcal{S},\phi)[i] = \mathrm{hist}(\mathcal{T},\psi)[i]$ *and all other entries in* $\mathrm{hist}(\mathcal{T},\psi)$ *are 0.*

Proof. The proof structure is identical to that of Lemma 3.2. 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 just adjust the representation so that the same number of w leaves are added to all nodes of the tree. This can easily be achieved in TC^0 using ancestor representations. □

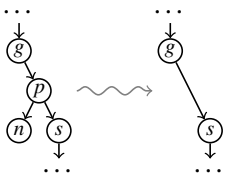
Proof of Theorem 1.4 We transform the input structure into an s-tree structure $\mathcal{T}$ and consider an equivalent MSO-formula ψ over $\tau_{s\text{-tree}}$ for ϕ using Lemma 4.5. Then we translate the formula into a classical tree automaton for binary labeled trees. Since evaluating such tree automata is in NC^1 [30], the claim follows. □

4.3 Computing Width-3 Tree Decompositions of Trees in Constant Depth

In the present section we show that using only TC^0-circuits, for every tree T given in term representation we can compute a width-3 tree decomposition (T_D, B_D) of T (regarded as a graph) such that T_D is a perfectly balanced binary tree (and, hence, has logarithmic depth).

Review of Tree Contraction Algorithms Our following proof uses the classical method of tree contraction [32, 1], which is used extensively in the context of PRAM-algorithms, but no NC^1-algorithms, let alone TC^0-algorithms, seem to be based on this method to the best of our knowledge. Starting with a tree T in which every inner node has exactly two children, 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 4.6. 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*).



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.

Parallel algorithms use tree contraction to perform a computation on the tree as follows: During each contraction step, they keep track of the "effect" the nodes that have been pruned and bypassed would have had. This "keeping track" is done by attaching information to the edges of the tree and during each contraction step the new edge from the sibling to the grandparent must store which effect the deleted node and its parent would have had. Tree contraction algorithms mainly differ in what is stored on these edges. For instance, for evaluating arithmetic trees over $+$ and $\times$, we store a linear function of the form $f(x) = ax + b$ along each edge in the form of the two numbers (a,b). In the context of our proof, however, nothing needs to be stored; it is the contraction tuples themselves that will serve as bags in the to-be-constructed tree decomposition.

Computing Prune and Bypass Positions in Constant Depth. 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, $\{l_2, l_4, l_6, \ldots, l_m\}$ will be removed, in the second round $\{l_3, l_7, l_{11}, l_{15}, \ldots, l_{m-1}\}$, and so on. In general, in the ith round we remove the leaves $\{l_{1+j2^{i-1}} \mid j \in \{1,3,5,\ldots,2^i-1\}\} = \{l_{j2^i-2^{i-1}+1} \mid j \in \{1,2,3,\ldots,2^{i-1}\}\}$.

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 $\mathrm{rank}(l)$ be the maximal number i such that l is still an element of T_i. Clearly, this rank function is computable in FTC^0.

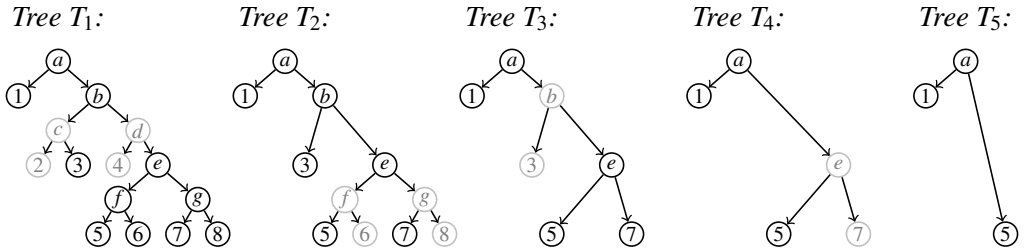


Figure 1: Example of a tree $T = T_1$ and the trees T_2, T_3, T_4 and T_5. In T_1 and T_2 the prune-and-bypass operation is applied to the even-numbered leaves that are left children (2 and 4) or right children (6 and 8) of their parents in T_1, respectively. In T_3 and T_4 the operation is applied to 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. 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 (but, possibly, the grandparent). 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 [13]:

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, which is the class under inspection in his paper. 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.

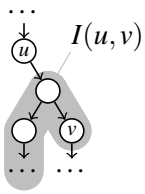
Computing the Tree Decomposition

Theorem 4.7. *There is a* DLOGTIME*-uniform* FTC^0*-circuit family 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 set, 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 4.3 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 4.4 allows us to embed S into a balanced tree and extending the bag function 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 $u \succ v$ let $I(u,v) = \{n \in V(T') \mid u \succ n \wedge v \not\succ n\}$ be the set of nodes "between" u and v. (Note, however, that a node in $I(u,v)$ does not need to lie on the path from u to v.) 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 2 for an example.



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 $r = \text{rank}(n)$ and $r' = \text{rank}(n')$ and without loss of generality assume $r \leq r'$. 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 $I(g,s) \subseteq I(x,y)$.

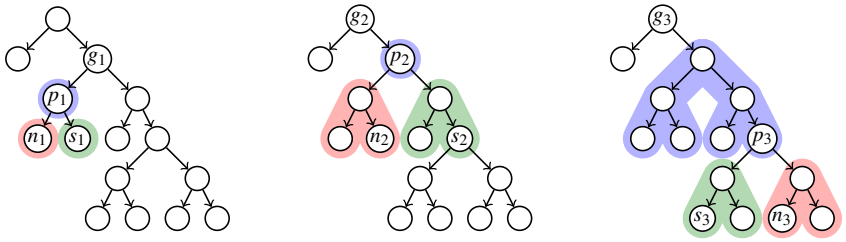


Figure 2: Three contraction tuples $c_1 = (n_1,s_1,p_1,g_1)$ to $c_3 = (n_3,s_3,p_3,g_3)$ of the tree T from Figure 1. Note how in each tree the sets $I(p_i,n_i)$ (red), $I(p_i,s_i)$ (green), and $I(g_i,p_i)$ (blue) are disjoint and that their union $I(c_i)$ is exactly one of the sets for the next tuple.

Let the nodes of 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.

It remains to show that we can compute an ancestor representation of S and that S has bounded degree. We saw already 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'. □

4.4 Computing Histograms by Logarithmic-Depth Circuits for Term Representations

Recall that our goal for the present section is to prove Theorem 1.5, that our line of proof was to do the same sequence of transformations as we did in Section 3 for constant depth circuits, and that the missing building block was a procedure to turn an arbitrary tree decomposition into a tree decomposition of logarithmic depth. Theorem 4.7 from the previous section provides us with the tools to build this missing block. In the following, we first show in Lemma 4.9 how this theorem can be used to balance tree decompositions and then prove Theorem 1.5.

Lemma 4.8. *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. Clearly, the width of (T',B'') is at most $ww'+w+w'$. 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 $T[\{n \mid v \in \bigcup_{x\in B'(n)} B(x)\}]$. □

As a corollary we obtain:

Lemma 4.9. *Let w be a fixed tree width. Then there is a* DLOGTIME*-uniform* TC^0*-circuit family that gets a term-represented width-w tree decomposition D of a graph G as input and outputs a term-represented width-$(4w+3)$ tree decomposition D' of G that is balanced.*

Proof. Apply Lemma 4.8 to Theorem 4.7. □

Just as in the proof of Theorem 3.12, in order to prove Theorem 1.5, we actually prove a stronger theorem were the bases are part of the input:

Theorem 4.10. *For every* MSO*-formula $\phi(X_1,\dots,X_\ell)$ over some signature τ and every $w \in \mathbb{N}$, there is a* DLOGTIME*-uniform* $\#\mathrm{NC}^1$*-circuit family that, on input of a τ-structure $\mathcal{S}$ along with a width-w tree decomposition in term representation for $\mathcal{S}$ and bases $b \in \mathbb{N}^\ell$, outputs* $\mathrm{num}_b(\mathrm{hist}(\mathcal{S},\phi))$.

Proof. After balancing the given tree decomposition using Lemma 4.9, we use a variant of Lemma 3.2 that expects tree decompositions $(\mathrm{term}(T),B)$ in term representations as inputs to turn $\mathcal{S}$ into an s-tree structure $\mathcal{T}$ and ϕ into an equivalent MSO-formula ψ. Note that the node degree of $\mathcal{T}$ is bounded in terms of the width of (T,B). We use Theorem 3.8 and consider an equivalent tree automaton A for ψ and extend its multiplicity bound to be higher than the degree of $\mathcal{T}$. Using the claim proved in Theorem 3.12, we can reduce to the problem of evaluating bounded fan-in arithmetic circuits without subtraction gates, a problem that can be solved in $\#\mathrm{NC}^1$. The resulting circuit family produces a number representation of the histogram. □

4.5 Application: Placing Problems in Logarithmic-Depth Circuit Classes

In this section, we show some examples of how to use the algorithmic meta 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 MSO-definable decision and counting problems on bounded tree width structures. In order to apply Theorems 1.4 and 1.5, the reductions also compute term or ancestor representations of bounded width tree decompositions.

Evaluating Boolean and Arithmetic Sentences The problem of evaluating Boolean sentences that are given as terms is well known to lie in DLOGTIME-uniform NC^1 via pebbling-based evaluation strategies that recursively split input sentences into sentences of almost the same size [12, 11] (a Boolean sentence is a Boolean formula with operations $\wedge$ and $\vee$ where the values of all input bits are determined). Theorem 1.4 provides a different route to prove this result 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 of S 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, each $+$ by an $\vee$ and each $\times$ by an $\wedge$, then the number of proof trees of the resulting Boolean formula equals the value of the arithmetic sentence [14]. Thus, by replacing operands in this way, Theorem 1.5 provides an alternative MSO- and tree-decomposition-based route to reprove the fact that arithmetic sentence evaluation is in $\#\mathrm{NC}^1$ [11, 14].

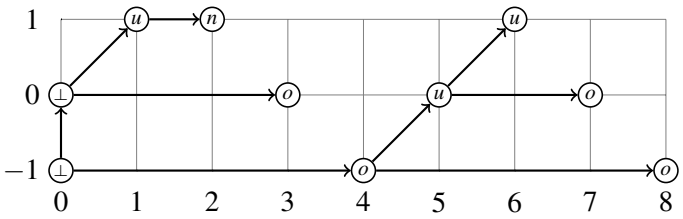
Evaluating Visible Pushdown Automata Buss [12] extended his NC^1-approach for the evaluation of Boolean sentences to also cover the membership problem for parenthesis languages. Later researchers adapted this approach to show that larger classes of context-free languages can be decided in NC^1, with the most general one being the result of Dymond [19] that input-driven languages are in NC^1. Input-driven languages are recognizable by *visible pushdown automata* (VPAs), which are pushdown automata $A = (\Sigma, \Gamma, \bot, Q, q_0, Q_a, \Delta)$ (that means Σ is the *input alphabet*, Γ is the *stack alphabet* with a distinguished *empty stack symbol* $\bot \in \Gamma$, Q is the *state set* with an *initial state* $q_0 \in Q$ and a subset $Q_a \subseteq Q$ of *accepting states*, and $\Delta \subseteq \Sigma \times \Gamma \times Q \times \Gamma^* \times Q$ is the *transition relation* 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) with the following property: 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 (see [4] for automata theoretic aspects of VPAs). Thus, 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. 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 and shown to be complete for $\#\mathrm{NC}^1$ by Krebs, Limaye, and Mahajan [28]. We will show how Theorems 1.4 and 1.5 can be used to reprove that these decision and counting problems are in NC^1 and $\#\mathrm{NC}^1$, respectively.

Let $A = (\Sigma, \Gamma, \bot, Q, q_0, Q_a, \Delta)$ be any VPA. We use a FTC^0-computation to translate input strings for A into a tree width-1 structure 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 signature of the structure as well as an MSO-formula that describes accepting computations depend only 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* $h_A(x,i) = |\{j \leq i \mid x_j \in \Sigma_{\text{PUSH}}\}| - |\{j \leq i \mid x_j \in \Sigma_{\text{POP}}\}|$. 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 $\mathcal{S}$ that we construct is a vertex labeled graph over the signature $\tau_A = \{E^2\} \cup \{P^1_\sigma \mid \sigma \in \Sigma\} \cup \{P^1_\bot\}$. Its node set S consists of the tuples $(i, h_A(x,i))$ for $i \in \{1, \dots, n\}$, and the tuples $(0,h)$ for $h \in \{\min_{i \in \{1,\dots,n\}} h_A(x,i), \dots, 0\}$. Its relations are build as follows: For each position $i \in \{1, \dots, n\}$, we put the node $(i, h_A(x,i))$ into $P^{\mathcal{S}}_{x_i}$. For

each $h \in \{\min_{i\in\{1,\ldots,n\}} h_A(x,i),\ldots,0\}$, we put the node $(0,h)$ into $P_\perp^{\mathcal{S}}$. For each position $i \in \{1,\ldots,n\}$, an edge (an element of $E^{\mathcal{S}}$) 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 $(i-1,h_A(x,i-1)) = (i-1,h_A(x,i)-1)$ to $(i,h_A(x,i))$. If $x_i \in \Sigma_{\text{POP}}$, let $i' < i$ be the largest index such that $(i',h_A(x,i))$ is a node of $\mathcal{S}$ (such a node always exists). We insert an edge from $(i',h_A(x,i')) = (i',h_A(x,i))$ to $(i,h_A(x,i))$. If $x_i \in \Sigma_{\text{NO-CHANGE}}$, we insert an edge from $(i-1,h_A(x,i-1)) = (i-1,h_A(x,i))$ to $(i,h_A(x,i))$. For each $h \in \{\min_{i\in\{1,\ldots,n\}} h_A(x,i),\ldots,-1\}$, we insert an edge from $(0,h)$ to $(0,h+1)$. Using FTC^0 computable arithmetic operations, the height function h_A and the structure $\mathcal{S}$ that is defined in terms of x and h_A 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 on the right, where the names of the nodes are given by their coordinates on the grid and 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.



By definition, edges of the structure lead only from left to right and there are no edges going downwards. Thus, the edges form a directed tree and, therefore, the structure's tree width is always 1. 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 S exactly if (a) $i = i' = 0$ and $h < h'$, or (b) $i < i'$, $h \le h'$, and for all i'' with $i < i'' < i'$ we have $h_A(x,i'') \ge h$.

To define accepting computation paths we use an MSO-formula ϕ that consists of two collections of free set variables. The first collection are unary *state labeling predicates* $Q_0,\ldots,Q_{|Q|-1}$ used to define a partition of all nodes that are labeled with symbols from Σ and the node $(0,0)$. An assignment of such a partition to these variables corresponds to guessing the states of the automaton at the input positions. The second collection of unary *stack content predicates* $P_1,\ldots,P_{|\Gamma\setminus\{\perp\}|}$ is used to guess a single stack symbol for every PUSH node. The MSO-formula evaluates to true if the sets assigned to the free variables satisfy these partition 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 at 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 elements are evaluated in the same way. For a state at a POP node e 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. We note that the structure that is computed by our reduction is different from the *mountain range* structures used by Mehlhorn in the context of context-free language recognition [31]; in fact, mountain ranges have unbounded tree width since they use additional edges to connect pairs of nodes that we connect implicitly by using an MSO formula that guesses paths between them. Applying Theorem 1.5 reproves 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 reproves that membership testing for input-driven languages is in NC^1.

Evaluating Tree Automata Our new method of balancing trees using width-3 tree decomposition, Theorem 4.7, allows us to give a "one paragraph proof" that deciding whether a fixed ranked deterministic tree automaton A accepts a tree T given in term representation as input lies in NC^1:

First, compute a width-3 tree decomposition (T_D, B_D) of T using Theorem 4.7. Modify the bag function B_D as follows: For each node n in a bag B, add all children of n to B. It is easily seen that in result we still have a tree decomposition of T and, since A is ranked, its width w is still some constant depending only on A. For a node n with a bag $B = \{x_1, \ldots, x_{|B|}\}$ attached to it, consider all sequences $(q_1, \ldots, q_{|B|})$ where all q_i are arbitrary states of A. We call such a sequence *valid*, if it has the following three properties: (a) If B contains a node x_i of T and also all of its child nodes $x_{i_1}, \ldots, x_{i_k}$ in T, then the transition function of A must map $(q_{i_1}, \ldots, q_{i_k})$ together with x_i's label to the state q_i. (b) We must be able to pick one valid sequence from each bag of the children of n in T_D such that when a node x_i is present in the bag of a child, the child's sequence assigns the state q_i to this node. (c) If x_i is the root of T, then q_i must be an accepting state. It is not hard to see that, since the number of possible sequences is fixed, an NC^1-circuit can compute the sets of valid sequences for all nodes of T_D in a bottom-up fashion and that A accepts T if, and only if, there is a valid sequence for the bag of T_D's root.

We remark that one could try to use Theorem 1.4 to reprove the same result more easily: An MSO-formula can easily guess a supposedly reached state for each node of an input tree and then check the local and hence global correctness of the guess. However, inside the proof of Theorem 1.4 we use the fact that tree automata can be evaluated in NC^1, so we cannot use our theorems to (re)prove this fact.

5 Conclusion

In the present paper we transferred the idea of unifying computational problems by using MSO-based problem definitions and tree decompositions to circuit complexity classes inside logarithmic space, leading to algorithmic meta theorems for Boolean and arithmetic circuit classes of constant and logarithmic depth. The theorems that are related to constant-depth circuits state that solving MSO-definable decision problems and computing MSO-definable solution histograms can be done in DLOGTIME-uniform AC^0 and GapAC^0, respectively, for input structures of bounded tree depth. The theorems for logarithmic-depth circuits state that these problems lie inside NC^1 and $\#\mathrm{NC}^1$, respectively, for input structures that are equipped with a tree decomposition of bounded width in term representation. While the theorem for GapAC^0 covers, for example, the problem of counting the total number of all solutions, it does not cover MSO-definable optimization problems that ask whether the number of solutions lies above (or below) some threshold that is given as part of the input; this can be remedied by using $\mathrm{TC}^0 = \mathrm{PAC}^0$ circuit families, which are able to simulate GapAC^0 circuits and look up individual bits of the histogram encoding. The algorithmic meta theorems stated and proved in this paper can be used to place problems in the corresponding uniform circuit classes. Regarding constant-depth circuits, we discussed how to put the unary version of SUBSETSUM and, more generally, the problem of solving a linear equation system that contains a constant number of equations whose coefficient are given in unary into TC^0. The most general application for logarithmic-depth circuits showed an alternative proof of a recent result that one can count the number of accepting paths of visible pushdown automata in $\#\mathrm{NC}^1$.

Relation to Prior Works and Techniques The structure of the proofs of the algorithmic meta theorems in this paper follow a classical plan: After constructing a tree decomposition (if not already provided as part of the input), the problem of computing the solution histogram for the input structure is reduced to computing an equivalent solution histogram for tree structures, which, in turn, is reduced to the problem of evaluating automata. We used the same proof plan in a recent paper [20] for which it was enough to compute all steps in deterministic logarithmic space. For the current paper we redesigned all proof building steps to satisfy the algorithmic requirements imposed by constant and logarithmic depth computations. While none of the proof steps survived this transformation from L to circuit classes unchanged, the most difficult parts were (a) finding an appropriate automaton notion for unordered unranked trees, (b) developing a simulation based on arithmetic circuits for it, and (c) balancing tree decompositions using FTC^0 reductions.

Outlook and Open Problems We highlighted some applications of the presented algorithmic meta theorems in this paper. One research direction would be to find more applications of the theorems. For example, can the theorems for logarithmic-depth circuits be used to simulate some generalization of visible pushdown automata where the height of the stack at different positions in time can be computed in advance; say, in NC^1 instead of FTC^0? Another direction would be to find algorithmic meta theorems that unify problems lying in other complexity classes around logarithmic space. Such research would need to address all three dimensions of algorithmic meta theorems: (a) the considered logic, (b) the considered class of input structures, and (c) the considered complexity class. We may go from MSO to more expressive or less expressive logics (like, for example, MSO on graphs where we can only quantify over vertex sets). Or we may consider other classes of structures that are more or less restrictive than bounded tree width (like, for example, bounded clique width). Other interesting complexity classes that are related to both space-bounded computations and circuits are LOGDCFL and LOGCFL.

Acknowledgements

Many of the results presented in this paper were initiated by discussions during the 2011 Dagstuhl seminar Computational Complexity of Discrete Problems. With special thank going to Martin Grohe for pointing us toward the notion of tree depth, we would like to thank the organizers of the seminar and all participants. We would also like to thank Christoph Stockhusen for helping us with the proof of TC^0-completeness of the integer programming problem.

References

[1] 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.

[2] 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.

[3] 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. Available from: http://ftp.cs.rutgers.edu/pub/allender/quaderni.pdf.

[4] R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

[5] 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 *LNCS*, pages 409–417. Springer, 1998. doi:10.1007/BFb0055790.

[6] 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.

[7] 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.

[8] 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.

[9] 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. Available from: http://hdl.handle.net/1783.1/738.

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

[11] 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.

[12] 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.

[13] 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. Available from: http://math.ucsd.edu/~sbuss/ResearchWeb/Boolean3/.

[14] 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.

[15] 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.

[16] 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.

[17] 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 *LIPIcs*, pages 215–226. Schloss Dagstuhl LZI, 2010. doi:10.4230/LIPIcs.STACS.2010.2456.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In M. Grohe and J. Makowsky, editors, *Model Theoretic Methods in Finite Combinatorics*, AMS Contemporary Mathematics Series. American Mathematical Society, 2011. to appear. Available from: http://www2.informatik.hu-berlin.de/~grohe/pub/grokre11.pdf.

[25] 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.

[26] N. Immerman. *Descriptive complexity*. Springer, New York, 1999.

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

[28] 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 *LNCS*, pages 44–53. Springer, 2010. Available from: 10.1007/978-3-642-14031-0_7.

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

[30] 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 *LNCS*, pages 201–215. Springer, 2001. doi:10.1007/3-540-45127-7_16.

[31] K. Mehlhorn. Pebbling moutain ranges and its application of DCFL-recognition. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming (ICALP 1980)*, volume 85 of *LNCS*, pages 422–435. Springer, 1980. doi:10.1007/3-540-10003-2_89.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

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

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

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

[44] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979. doi:10.1016/0304-3975(79)90044-6.

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

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

ECCC ISSN 1433-8092

http://eccc.hpi-web.de