Logspace Versions of the Theorems of Bodlaender and Courcelle

Michael Elberfeld   Andreas Jakoby   Till Tantau

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

August 19, 2010

Abstract

Bodlaender's Theorem states that for every k there is a linear-time algorithm that decides whether an input graph has tree width k and, if so, computes a width-k tree composition. Courcelle's Theorem builds on Bodlaender's Theorem and states that for every monadic second-order formula ϕ and for every k there is a linear-time algorithm that decides whether a given logical structure $\mathcal{A}$ of tree width at most k satisfies ϕ. We prove that both theorems still hold when "linear time" is replaced by "logarithmic space." The transfer of the powerful theoretical framework of monadic second-order logic and bounded tree width to logarithmic space allows us to settle a number of both old and recent open problems in the logspace world.

Keywords: logarithmic space, tree width, partial k-trees, monadic second-order logic, Bodlaender's Theorem, Courcelle's Theorem

1 Introduction

For graphs of bounded tree width, a concept introduced by Robertson and Seymour [37] and known under several different names such as *partial k-trees*, the computational complexity of many difficult problems drops significantly compared to the same problem for general graphs. For instance, for every k the NP-complete problem HAMILTONICITY can be solved in linear sequential and in logarithmic parallel time when restricted to graphs of tree width at most k. The same is true for many other NP-complete problems, see [10] for an overview. To achieve these time bounds, algorithms for problems on graphs of bounded tree width need access to a tree decomposition of the input graph. Bodlaender's Theorem [8] states that for every fixed k, on input of a graph G of tree width at most k such a tree decomposition can be computed in linear time. Note that k must, indeed, be a fixed constant since it is NP-complete [2] to decide on input (G, k) whether G has tree width at most k.

These results have inspired researchers to investigate whether graphs of bounded tree width may also be helpful in the study of logarithmic space. Here, "difficult" problems include normally easy ones like reachability or matching. The hope is that these problems might be decidable by deterministic logspace Turing machines (logspace DTMs) for graphs of bounded tree width. Only partial results were obtained, for instance for graphs of tree width 2 or for k-trees, and two 2010 papers [19, 20] identify an analogue of Bodlaender's Theorem for logarithmic space as the central piece missing in recent advances in the study of logarithmic space. Our first main result is such an analogue:

Theorem 1.1. *For every $k \geq 1$, there is a logspace DTM that on input of any graph G of tree width at most k outputs a width-k tree decomposition of G.*

The design of early efficient algorithms for problems on graphs of bounded tree width was a laborious process involving complex, problem-dependent arguments. A breakthrough came with Courcelle's Theorem [17], which in conjunction with Bodlaender's Theorem yields that all graph properties expressible in monadic second-order logic (MSO-logic) can be solved in linear time on graphs of bounded tree width. Since many graph properties are easily expressible in this logic, we get a simple, unified framework for showing that all of these problems are efficiently solvable.

In the logspace world, the situation resembles the one before Courcelle's work: each paper uses similar, but still problem-dependent arguments to establish membership in L or at least in LOGCFL. Our second main result is that Courcelle's Theorem also holds for logarithmic space, enabling us to apply the same unifying framework as for linear time:

Theorem 1.2. *For every $k \geq 1$ and every* MSO-*formula ϕ, there is a logspace* DTM *that on input of any logical structure $\mathcal{A}$ of tree width at most k decides whether $\mathcal{A} \models \phi$ holds.*

Courcelle's original theorem has been generalized in different ways (known as *functional, optimization, counting,* and other versions). We also prove such a generalized version, which we call the *cardinality version* and which allows a wider range of applications than Theorem 1.2. For its formulation we introduce the notion of *solution histograms:* Let $\phi(X_1, \ldots, X_d)$ be an MSO-formula whose free predicate variables are exactly the X_i and let $\mathcal{A}$ be a logical structure with universe A. Then histogram$(\mathcal{A}, \phi)$ is the d-dimensional integer array whose entry at the d-dimensional index $(i_1, \ldots, i_d) \in \{0, \ldots, |A|\}^d$ tells us how *many* subsets $S_1, \ldots, S_d \subseteq A$ exist with $|S_1| = i_1$, ..., $|S_d| = i_d$ and $\mathcal{A} \models \phi(S_1, \ldots, S_d)$. Observe that solution histograms store a lot of information about ϕ and $\mathcal{A}$, including the number of satisfying assignments for ϕ in the form of the sum of all entries. Also observe that Theorem 1.2 is a special case of the following Theorem 1.3 for $d = 0$ since the 0-dimensional solution histogram is just a single scalar that is 1 if $\mathcal{A} \models \phi$, and 0 otherwise.

Theorem 1.3 (Logspace Cardinality Version of Courcelle's Theorem). *For every $k \geq 1$ and every* MSO-*formula $\phi(X_1, \ldots, X_d)$, there is a logspace* DTM *that on input of any logical structure $\mathcal{A}$ of tree width at most k outputs* histogram$(\mathcal{A}, \phi)$.

The above theorems make no claim concerning the behavior of the machines for input structures that have a tree width larger than k, but the following lemma shows that this could be remedied:

Lemma 1.4. *For every $k \geq 1$ the language* TREE-WIDTH-k, *which contains exactly the graphs of tree width at most k, is* L-*complete under first-order reductions.*

Our main technical contributions are the following: For the proof of the logspace version of Bodlaender's Theorem, the main difficulty lies in coming up with an appropriate notion of graph separators and in showing how the recursive decomposition can be done in logarithmic space. We side-step the recursion by reducing to a special version of the reachability problem for mangrove graphs, which we show to lie in L. For the logspace cardinality version of Courcelle's Theorem, we show how computing the number of satisfying assignments relates to tree automata (a standard tool in proofs of Courcelle's Theorem) and how it can be reduced to evaluating an arithmetic tree whose entries are tensors that are added and convoluted. This problem, in turn, can be reduced to evaluating a normal arithmetic tree over addition and multiplication, a problem known to be logspace solvable.

Applications. Our results can be applied in a number of areas. Since in the present paper we focus on proving the main theorems, we will not explore these applications in more detail. Nevertheless, we below try to sketch their impact on some of these areas.

First, in parameterized complexity theory numerous problems like 3-COLORABLE or HAMILTONICITY can be shown to be fixed-parameter tractable with respect to the parameter "tree width" by expressing

the problems in MSO-logic and applying Courcelle's Theorem. By Theorem 1.2, all of these results can be transferred to the logspace setting. Problems like DOMINATING-SET are also expressible as MSO-formulas, but now $\phi(X)$ holds if X is a dominating set and the question is whether ϕ can be satisfied by an X having a certain size. The logspace *cardinality* version of Courcelle's Theorem shows that this problem can also be decided in logarithmic space. Thus, for every k the languages $\{G \mid \mathrm{tw}(G) \leq k$ and G is 3-colorable$\}$ and $\{(G,s) \mid \mathrm{tw}(G) \leq k$ and G has a dominating set of size at most $s\}$ lie in L and this is the case for many other problems. Even when a problem is not directly expressible in MSO-logic, it may still be possible to use Courcelle's Theorem inside a larger algorithm. A simple example is computing the chromatic number of tree width bounded graphs in logarithmic space: no MSO-formula $\phi(X)$ is known that expresses that a graph has chromatic number $|X|$, but it is easily seen [22] that graphs of tree width k have chromatic number at most $k+1$ and we can successively test whether a graph is 1-, 2-, ..., $(k+1)$-colorable to compute its chromatic number.

Second, a number of graph properties, such as reachability, can be expressed in MSO-logic, but they can already be checked efficiently on *arbitrary* graphs and applying Courcelle's classical theorem yields no new insights. From a logspace perspective, the situation is different: Applying Theorem 1.3 to the formula $\phi(X)$ expressing that X is a simple path from s to t shows that the problem $\{(G,s,t) \mid \mathrm{tw}(G) \leq k$, there is a path from s to t in $G\}$ lies in L. Indeed, on input of (G,s,t,d) we can even compute in logarithmic space the exact number of simple paths from s to t of length exactly d. Another example is the matching problem, where the MSO-characterization allows us to compute the exact number of perfect matchings of graphs of bounded tree width in logarithmic space.

Third, the logspace version of Courcelle's Theorem has applications in the study of pseudopolynomial NP-complete problems: The classical NP-complete problem KNAPSACK is well-known to become tractable when input numbers are coded in unary: UNARY-KNAPSACK $\in$ NL. Inspired by Cook's conjecture [16] that "a problem in NL which is probably not complete is the knapsack problem with unary weights," a line of research began to capture its complexity with specialized complexity classes lying between L and NL [27, 15, 31], see also [33]. Our Theorem 1.3 shows that UNARY-SUBSETSUM $\in$ L: Given unary-coded numbers a_1, ..., a_n and a target sum s, construct the graph consisting of n stars, where the ith star has $a_i - 1$ leafs, and the formula $\phi(X)$ expressing that X must always contain a star completely or not at all. Then there is a satisfying assignment of cardinality s iff there is a subset $I \subseteq \{1, \dots, n\}$ with $\sum_{i \in I} a_i = s$. Similar, but slightly more complex arguments show that UNARY-KNAPSACK $\in$ L and also that UNARY-k-KNAPSACK $\in$ L for every k, where there are k knapsacks.

Related Work. The research on the difficulty of computing width-k tree decompositions was originally focussed on time complexity. The linear time bound of Bodlaender's Theorem [8] is the best possible result on the sequential time complexity and improves on previous results [2, 35]. Similarly, the parallel time complexity was reduced in a line of papers [13, 6, 32, 9] to $O(\log n)$. Concerning the space complexity, an algorithm of Gottlob et al. [23], which extends an algorithm of Wanke [39], shows that computing width-k tree decompositions lies in the class LOGCFL = $\mathrm{SAC}^1 \subseteq \mathrm{AC}^1$. For the special case of computing width-2 tree decompositions a logspace algorithm can be deduced from [28, 29], together with Reingold's algorithm [36].

Algorithmic variants of Courcelle's Theorem also solve MSO-definable counting and optimization problems [3, 11, 18], similar to the cardinality version studied in the present paper. A recent survey [10] spans the time complexity of tree width related problems. The best result concerning the space complexity of analogues of Courcelle's Theorems is due to Wanke [39]. It places MSO-definable decision problems for graphs of bounded tree width in LOGCFL. Additionally, MSO-definable optimization problems like VERTEX-COVER on bounded tree width graphs lie in LOGCFL.

Only few problems were known to lie in L when restricted to graphs of bounded tree width. In [30] we showed that the reachability problem, the shortest path, and also the longest path problems lie in L when restricted to graphs of tree width 2. Das et al. [19] study k-trees, a special case of graphs of

tree width k, and show that for these graphs the reachability and the perfect matching problem and, if the graph is a directed acyclic graph (DAG), also the shortest and longest path problems lie in L. All of these results are special cases of the logspace cardinality version of Courcelle's Theorem.

Concerning our proof techniques, the idea of using separators in the construction of tree decompositions is used in many other papers [32, 35]. Mangroves are also used in the study of the isomorphism problem for k-trees [4]. Our reduction in the logspace cardinality version of Courcelle's Theorem is a strong generalization of the reduction to $\{max, +\}$-trees that we first proposed in [30] and was later also used in [19].

Organization of This Paper. In Section 3 we show that given a graph of tree width at most k, we can compute a tree decomposition of width $4k + 3$, called an *approximate* tree decomposition, in logarithmic space. In Section 4 we prove the logspace cardinality version of Courcelle's Theorem. The algorithms of this section work on approximate tree decompositions. In Section 5 we prove the L-completeness of TREE-WIDTH-k and the logspace version of Bodlaender's Theorem by showing that approximate tree decompositions can be turned into optimal ones in logarithmic space.

2 Preliminaries

In the following we shortly describe notations and concepts that are used in the present paper. For a detailed discussion, we refer to the textbook of Flum and Grohe [22].

Structures and Monadic Second-Order Logic. A *vocabulary* τ 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* $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}})$ consists of a nonempty, finite set A, the *universe* of $\mathcal{A}$, and for each relation symbol $R_i^{r_i} \in \tau$ a *relation* $R_i^{\mathcal{A}} \subseteq A^{r_i}$. In the present paper we only consider finite structures.

Let $\mathcal{A}$ be a τ-structure. Another τ-structure $\mathcal{B}$ is a *substructure* of $\mathcal{A}$ if $B \subseteq A$ and $R^{\mathcal{B}} \subseteq R^{\mathcal{A}}$ holds for all relations $R \in \tau$. For a subset $B \subseteq A$, the substructure of $\mathcal{A}$ that is *induced on B* has universe B and for all $R^r \in \tau$ we have $R^{\mathcal{B}} = R^{\mathcal{A}} \cap B^r$. We denote it by $\mathcal{A}[B]$.

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* ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$), *element quantifiers* ($\exists x, \forall x$), and *set quantifiers* ($\exists X, \forall X$). *Bound* and *free* variables are defined as usual. We write $\phi(X_1, \ldots, X_d)$ for a formula ϕ with free variables $X_1, \ldots, X_d$. For a vocabulary τ, a τ-structure $\mathcal{A}$ with universe A, a τ-formula $\phi(X_1, \ldots, X_d)$, and sets $S_1, \ldots, S_d \subseteq A$, we write $\mathcal{A} \models \phi(S_1, \ldots, S_d)$ to indicate that ϕ holds in the structure $\mathcal{A}$ if each X_i is interpreted as S_i

For the special case of graphs, monadic second-order logic is also called MSO$_1$-logic and one can additionally study the so-called MSO$_2$-logic where one may quantify not only over sets of vertices, but also over sets of edges [26]. However, this logic is just a special case of MSO-logic if we allow arbitrary vocabularies τ, as we do in the present paper. Because of this, we only consider MSO-logic as defined above.

If we want structures to be processed by Turing machines, we need to encode them as strings. This can be done by representing the elements of the universe A by numbers $\{0, \ldots, |A| - 1\}$ and the tuples of every relation $R^{\mathcal{A}}$ by tuples of numbers.

Unless stated otherwise, we assume all numbers in inputs and outputs to be coded in binary.

Graphs and Trees. A *directed graph* is a pair (V, E), consisting of a set V of *vertices* and a set $E \subseteq V \times V$ of *edges*. We write $V(G)$ for G's vertex set and $E(G)$ for the edge set. 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 a graph G are its maximal connected subgraphs; the empty graph has zero components. Graphs are special cases of logical structures, namely $\{E^2\}$-structures $\mathcal{G} = (V, E^{\mathcal{G}})$. The concept *subgraph* and *induced subgraph* are inherited from general structures. The *children* of a vertex v of a directed graph $G = (V, E)$ are all vertices u with $(v, u) \in E$.

A *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. In the present paper all trees are assumed to have a distinguished root and are assumed to be directed. This is no loss of generality since undirected trees (acyclic, connected, undirected graphs) can easily be rooted in logarithmic space. We use the term *nodes* to refer to the vertices of a tree. A *labeled tree with label alphabet* Σ is a tree T together with a mapping $l \colon V(T) \to \Sigma$.

The *leafs* of a tree are nodes without children; all other nodes are *inner nodes*. A tree is *binary* if every inner node has *exactly* two children. In a binary tree, we may wish to distinguish between the *left* and the *right* child of a node. In such a case, we call T a *binary tree with distinguished left and right children*. Formally, T is a labeled tree where for every inner node exactly one of the children has the label "is left child." A tree is *balanced* if all root-to-leaf paths have the same length. Note that balanced binary trees have depth $\lceil \log_2(|V(T)| + 1) \rceil$.

Tree Automata. A (binary, bottom-up) *tree automaton* is a tuple $M = (Q, Q_a, q_0, \Sigma, \delta)$, where Q is the set of *states*, $Q_a \subseteq Q$ is the set of *accepting states*, q_0 is the *initial state*, Σ is the *alphabet*, and $\delta \colon Q \times Q \times \Sigma \to Q$ is the *state transition function*. A tree automaton inductively assigns a state to a labeled binary tree T with distinguished left and right children as follows: The empty tree has state q_0. The state of a nonempty tree with root r and root label $l(r)$, left subtree state q_{left}, and right subtree state q_{right} is $\delta(q_{\text{left}}, q_{\text{right}}, l(r))$. The tree automaton *accepts* all trees to which it assigns a state from Q_a.

Tree Decompositions and the Gaifman Graph. The concept of tree decompositions of graphs was introduced by Robertson and Seymour [37]; we use a generalized definition for logical structures [22]:

Definition 2.1 (Tree Decomposition). A *tree decomposition* of a τ-structure $\mathcal{A}$ is a labeled tree T whose labeling function $B \colon V(T) \to \{X \mid X \subseteq A\}$ has the following properties:

1. For all $a \in A$, the induced subtree $T\big[\{n \in V(T) \mid a \in B(n)\}\big]$ is nonempty and connected.
2. For every $R^r \in \tau$ and every tuple $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$, there is an $n \in V(T)$ with $\{a_1, \ldots, a_r\} \subseteq B(n)$.

The sets $B(n)$ are called *bags*. The *width* of a tree decomposition T is $\max_{n \in V(T)} |B(n)| - 1$. The *tree width* of a structure $\mathcal{A}$, denoted by $\text{tw}(\mathcal{A})$, is the minimum width over all its tree decompositions. A class of τ-structures has *bounded tree width* if the tree width of all its elements is bounded by a constant.

The *Gaifman graph* of a structure $\mathcal{A}$ is an undirected graph that has vertex set A and there is an edge $(a, a') \in A \times A$ iff one of the relations $R^{\mathcal{A}}$ contains a tuple $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$ with $a, a' \in \{a_1, \ldots, a_r\}$. Since a tuple of r elements from the structure gives rise to a clique of size r in the Gaifman graph and in a tree decomposition every clique is completely contained in some bag, the following fact holds:

Fact 2.2 ([22]). *Let $\mathcal{A}$ be a structure. Then every tree decomposition of the Gaifman graph of $\mathcal{A}$ is also a tree decomposition of $\mathcal{A}$ and vice versa.*

3 Computing Approximate Tree Decompositions in Logarithmic Space

Bodlaender's Theorem is typically proved in two steps: First, a linear-time algorithm is presented that on input of a graph of tree width at most k computes a tree decomposition of width at most $O(k)$, called an *approximate* tree decomposition. Second, another linear-time algorithm is used to turn the approximate tree decomposition into an optimal one. We proceed similarly: The present section is devoted to a proof of Lemma 3.1 below, which states that approximate tree decompositions can be computed in logspace. In Section 5 we will show how optimal tree decompositions can be computed.

Lemma 3.1. *For every $k \geq 1$, there is a logspace DTM that on input of any structure $\mathcal{A}$ with $\mathrm{tw}(\mathcal{A}) \leq k$ outputs a tree decomposition for $\mathcal{A}$*

1. *whose width is at most $4k+3$ and*
2. *whose decomposition tree is binary and balanced.*

For many applications, it suffices to have access to tree decompositions satisfying part 1 of the lemma. However, for the proof of the logspace cardinality version of Courcelle's Theorem in Section 4 we need access to tree decompositions of constant degree and logarithmic depth. Part 2 shows that such tree decompositions can be obtained in logarithmic space.

In the following, for the proof of Lemma 3.1 we only consider *undirected, connected graphs* instead of arbitrary logical structures, because a structure and its Gaifman graph have the same tree decompositions [22] and because different components of a graph can be decomposed independently.

Algorithms for constructing tree decompositions often employ a specific notion of *separators*, which are used to split a graph into smaller subgraphs for which tree decompositions can be computed recursively. When one wants to transfer this idea to logarithmic space, one faces the problem that both the recursion stack and the intermediate subgraphs are too large to store. We overcome these problems in two ways: First, instead of avoiding deep recursions, we show how a special version of the reachability problem in mangrove graphs can be used to identify the desired tree decomposition. Second, we pick a notion of separators that allows us to represent subgraphs in logarithmic space.

3.1 Transitive Closures of Mangroves.

In our tree decomposition algorithm, the decomposition tree will be an induced subgraph of a larger graph in which it is "hidden." In order to recover the tree, we need to compute the set of vertices reachable from a given vertex. For the *mangrove graphs* that arise in our proofs the best upper space bound on the reachability problem is $O(\log^2 n / \log \log n)$, see [1], which is far from logarithmic. So, our algorithms will need access to some kind of additional information. This information will be in the form of what we call *transitive closures of related vertices*.

A *mangrove* [1] is a DAG in which there is at most one path between any two vertices. In a mangrove, the subgraph T_r induced on all vertices reachable from a given vertex r is a tree rooted at r. Let us say that two vertices a and b of a mangrove are *related* if they are both present in some T_r, that is, they are both reachable from some vertex r. We say that a graph R is *a transitive closure of the related vertices of M* if the following holds: Whenever a and b are related in M, then there is an edge from a to b in R iff there is a non-empty path from a to b in M. The example on the right shows a mangrove M at the top. All transitive closures R of M's related vertices can be obtained by arbitrarily adding edges in the lower "template" graph along the dotted lines, which connect exactly the unrelated vertices of M.

Mangrove M

Template for graphs R
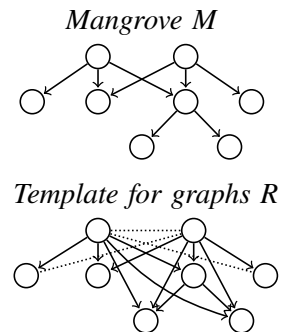
Lemma 3.2. *There is a logspace DTM that on input of any pair (M, R) consisting of a mangrove M and a transitive closure R of M's related vertices outputs the transitive closure of M.*

Proof. Let a mangrove M and a transitive closure R of M's related vertices be given as input as well as two vertices $a, b \in V(M) = V(R)$. We claim that the following algorithm, which clearly needs only logarithmic space, correctly decides whether there is a non-empty path from a to b in M:

```
1    current ← a
2    while not (current, b) ∈ E(M) do
3        if there is exactly one v ∈ V(M) with (current, v) ∈ E(M) ∧ (v, b) ∈ E(R)
4            then current ← v
5            else reject
6    accept
```

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

Lemma 3.2 states that given a mangrove and a transitive closure of its related vertices, we can compute the (exact) transitive closure of the mangrove. However, the algorithm may incorrectly accept or reject when the input does not satisfy the promise that M is a mangrove and R is a transitive closure of M's related vertices. Although this situation does not arise in our proofs, for completeness we include the following lemma, which states that we can detect whether an input is a mangrove with its correct transitive closure.

Lemma 3.3. MANGROVE-AND-TC $= \{(G, R) \mid G$ is a mangrove and R is its transitive closure$\} \in$ L.

Proof. A directed graph G is a mangrove if, and only if, for every vertex r the subgraph induced on the vertices reachable from r in G is a tree rooted at r: First, if G is a mangrove, then in this subgraph there is a unique path from r to every vertex, which is one possible definition of a rooted tree. Second, if all of these subgraphs are trees, then G is acyclic and there cannot be two different paths from any r to any other vertex.

The logspace algorithm for deciding MANGROVE-AND-TC works as follows. On input (G, R), it first tests whether all edges of G are also present in R and it tests whether $(u, v) \in R \wedge (v, w) \in R$ always implies $(u, v) \in R$. If R passes these tests, we know its edge set is a superset of the transitive closure of G. The second step of the algorithm is to iterate over all vertices r of G. Each time it considers the subgraph T_r of G induced the vertex set $\{r\} \cup \{v \mid (r, v) \in R\}$. This subgraph is the graph induced by the vertices that R "claims" to be reachable from r in G. The algorithm checks whether T_r is, indeed, a tree rooted at r – a property that is well known to be decidable in logarithmic space. If all T_r pass the test, the algorithm accepts, otherwise is rejects.

For the correctness of the algorithm, just note that each T_r must contain at least the vertices reachable from r in G because R is a superset of the transitive closure of G. However, if R is actually too large and T_r contains at least one vertex v that is not reachable from r in G, then T_r is no longer a tree rooted at r and the algorithm will reject. Thus, the algorithm only accepts if R is the transitive closure of G. Finally, since all T_r are trees rooted at r, we can also conclude that G must be a mangrove. $\square$

3.2 Descriptors and Descriptor Decompositions.

Let $G = (V, E)$ be a connected undirected graph. A *descriptor D in G* is either (a) just a bag $B \subseteq V$ and called *simple* or (b) a pair (B, v) consisting of a bag $B \subseteq V$ and a *component selector* $v \in V - B$. We write $B(D)$ for the bag of D. We say that D *describes* the following graph $G(D)$: If D is simple, $G(D) = G[B]$. Otherwise let $G(D) = G[V(C) \cup B]$ where C is the component of $G[V - B]$ that contains v. We write $V(D)$ for the vertex set of the graph $G(D)$. The *interior* $I(D)$ of $G(D)$ is $V(D) - B(D)$. Let D_G denote the descriptor $(\emptyset, v_0)$ where v_0 is a chosen vertex of G. Note that $G(D_G) = G$ since we assumed G to be connected. An example of a graph G, a descriptor $D = (\{b_1, b_2, b_3\}, v)$, and the sets $V(D)$ and $I(D)$ is shown right. The graph $G(D)$ is the induced subgraph $G[V(D)]$.
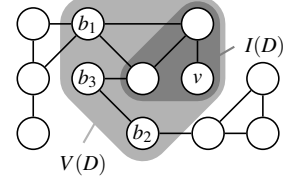
Definition 3.4 (Descriptor Decomposition). Let G be a connected undirected graph. A *descriptor decomposition* of G is a directed graph M whose vertices are descriptors in G, one of them is D_G, and where for every node D of M with children $D_1, \ldots, D_m$ the following holds:

1. For each child D_i, we have $V(D_i) \subseteq V(D)$ and $I(D_i) \subseteq I(D)$ and at least one inclusion is proper.
2. For each child D_i, the set $V(D_i)$ contains at least one vertex from $I(D)$.
3. For each child D_i, the set $I(D_i)$ is disjoint from all $V(D_j)$ for $j \neq i$.
4. Each edge in $G(D)$ that is not between two vertices in $B(D)$ must be present in some $G(D_i)$.

Observe that property 1 implies that all descriptor decompositions are DAGs.

We next prove that given a descriptor decomposition M of a graph G, the graph $M[W]$ where W is the set of all vertices reachable from D_G in M "nearly forms a tree decomposition of G": We only need to add an internal vertex between each node and its children whose bag contains all "interactions" between the children of the node. Formally, we define a graph $T(M)$ as follows: For each descriptor D reachable from D_G in M, it contains two vertices D^{normal} and D^{interact}. If $D_1, \ldots, D_m$ are the children of D in M, then there are edges from D^{normal} to D^{interact} and from D^{interact} to each D_i^{normal}. Label D^{normal} with the bag $B(D)$. Label D^{interact} with the bag that contains all vertices present in at least two of the sets in $\{B(D), B(D_1), \ldots, B(D_m)\}$. Below, we show an example of a descriptor D, its children D_1, D_2, and D_3 in M, and their bags; as well as the resulting nodes and bags in $T(M)$.
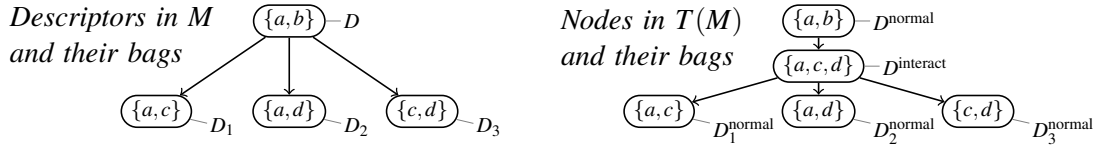
Lemma 3.5. *If M is a descriptor decomposition of G, then $T(M)$ is a tree decomposition of G.*

Proof. Our first claim is that M is a mangrove, which will imply that $T(M)$ is a tree. By property 1 of descriptor decompositions, M is a DAG. To prove that there is at most one path between any two vertices of M, suppose there is a vertex D_s with two different children D_1 and D_2 such that a vertex D_t is reachable from both. By property 2 of descriptor decompositions, $G(D_t)$ contains at least one $v \in I(D_1)$. By the third property, $G(D_2)$ does *not* contain v, but $G(D_t)$ must be a subgraph of $G(D_2)$ by the first property, which is a contradiction.

We claim that $T(M)$ is a tree decomposition of G. Since we just saw that M is a mangrove, $M[W]$ where W is the set of all vertices reachable from D_G in M is a tree and thus also $T(M)$. We need to check the two properties of a tree decomposition:

1. Consider the set of all nodes of $T(M)$ whose bags contain some vertex v. It suffices to prove that there is a unique node of $T(M)$ whose bag contains v, but whose parent node's bag does

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

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

Lemma 3.6. *There is a logspace* DTM *that on input of any graph G together with a descriptor decomposition M of G outputs $T(M)$.*

Proof. In Lemma 3.5 we proved that M is a mangrove. We claim that the following graph R is a transitive closure of M's related vertices: Its vertex set is $V(M)$ and there is an edge from D to D' in R iff D and D' satisfy property 1 of Definition 3.4, that is, $V(D') \subseteq V(D)$ and $I(D') \subseteq I(D)$ and at least one of these two inclusions is proper. Then R is clearly a superset of the transitive closure of M. Second, if there are two disjoint paths leading from a vertex D_s to two vertices D_1 and D_2, then, as argued in Lemma 3.5, $G(D_1)$ and $G(D_2)$ each contains at least one vertex not contained in the other graph. Thus, there is no edge between them in R.

Observe that the graph R is logspace-computable since Reingold's algorithm [36] allows us to check in logarithmic space on input of G and D and a vertex v whether $v \in G(D)$. This allows us to apply Lemma 3.2 to M and R in order to compute the set of vertices reachable from D_G in M in logarithmic space, yielding $T(M)$. $\qquad\square$

By Lemmas 3.5 and 3.6, in order to compute a tree decomposition of a graph, it suffices to compute a descriptor decomposition. We next show that such a descriptor decomposition can be obtained in logarithmic space. As remarked earlier, algorithms for computing tree decompositions internally use different kinds of *separators*. The ones we use are also known as *balanced separators*.

Definition 3.7. Let G be an undirected graph and let $U \subseteq V(G)$. A *separator* $S \subseteq V(G)$ *separates U in G* if each component of $G[V(G) - S]$ contains at most $|U|/2$ vertices of U. An *s-separator* is a separator of size at most s.

It is a folklore fact that for all G with $\text{tw}(G) \leq k$ every $U \subseteq V(G)$ has a $(k+1)$-separator S in G.

We use the following convention in the remaining part of this paper: Whenever we write that a DTM should "choose" some vertex or set, we mean that a deterministic choice is made. For instance, we can always choose the lexicographically first vertex or set.

Definition 3.8 (Child descriptors). Let G be a connected undirected graph with $\text{tw}(G) \leq k$. We define the *child descriptors* of a descriptor D in G as follows: If D is simple, it has no child descriptors. Otherwise we call D *small* if $|B(D)| \leq 2k+2$ and choose a $(k+1)$-separator S of $V(D)$ in the graph $G(D)$; and we call D *large* if $|B(D)| > 2k+2$ and choose a $(k+1)$-separator S of $B(D)$ in $G(D)$. Let C_1 to C_m be the components of $G[I(D) - S] = G[V(D) - (S \cup B(D))]$. For each $i \in \{1, \ldots, m\}$ choose a vertex $v_i \in C_i$ and let B_i be the set of all vertices in $B(D) \cup S$ that are adjacent in $G(D)$ to a vertex from C_i. Then the descriptors (B_i, v_i) are the child descriptors of D and, unless $S \subseteq B(D)$, additionally the simple descriptor $D_0 = B(D) \cup S$.

To the right, we show how child descriptors are constructed. The left graph is $G(D)$ for the small descriptor $D = (\{b_1, b_2\}, v)$. The set $S = \{s_1, s_2\}$ is a separator of $V(D)$ in $G(D)$. Removing the highlighted set $B(D) \cup S$ yields the two components C_1 and C_2. The sets B_1 and B_2 contain the vertices from $B(D) \cup S$ adjacent to C_1 and C_2, respectively. The child descriptors of D are $D_1 = (\{b_1, b_2, s_1\}, v_1)$, $D_2 = (\{b_2, s_1\}, v_2)$, and $D_0 = \{b_1, b_2, s_1\}$.
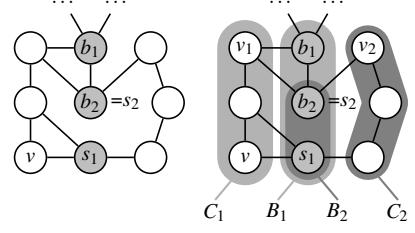
Lemma 3.9 (Size Lemma). *Let D be a non-simple descriptor and let D' be a child descriptor of D.*

1. *If D' is simple, then $|B(D')| \leq |B(D)| + k + 1$.*
2. *If D is small, then $|V(D')| \leq |V(D)|/2 + 3k + 3$ and $|B(D')| \leq 3k + 3$.*
3. *If D is large, then $|B(D')| < |B(D)|$.*

Proof. The claim for simple D' follows from the fact that its bag is of the form $B \cup S$ with $|S| \leq k + 1$. If $D = (B, v)$ is small, then each B_i can contain at most $|B| + |S| \leq 2k + 2 + k + 1 = 3k + 3$ vertices as claimed. The chosen separator S separates $V(G)$ in $G(D)$. Then each component C_i of $G[I(D) - S]$ can contain at most half the vertices of $V(D)$ and, hence, $V(D_i)$ can contain at most $|V(D)|/2 + |B| + |S| \leq |V(D)|/2 + 2k + 2 + k + 1$ vertices as claimed. If D is large, then S separates B in $V(D)$. This means that each B_i can contain at most $|B|/2$ vertices from B and must hence have size at most $|B|/2 + |S|$. Since D is large, we have $|B| \geq 2k + 3$ and, therefore, $|S| = k + 1 < |B|/2$. We conclude that $|B_i| \leq |B|/2 + |S| < |B|/2 + |B|/2 = |B|$. □

We are now ready to define the desired descriptor decomposition and to show that it is, indeed, a descriptor decomposition, has logarithmic depth, and is logspace-computable.

Definition 3.10. Let G be an undirected, connected graph with $\operatorname{tw}(G) \leq k$. Let $M(G)$ be the graph whose vertex set contains all descriptors D in G with $|B(D)| \leq 3k + 3$ for non-simple D and $|B(D)| \leq 4k + 4$ for simple D and where there are edges from each descriptor exactly to its child descriptors.

Lemma 3.11. *The graph $M(G)$ is a descriptor decomposition of G.*

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

Lemma 3.12. *The tree decomposition $T(M(G))$ has width at most $4k + 3$ and depth at most $c \log_2 n$, where c is a constant depending only on k and n is the number of vertices of G.*

Proof. Concerning the width, observe that all bags of $T(M)$ attached to normal nodes have maximum size $4k + 4$. For interaction nodes, the bag attached to the node is a subset of $B \cup S$ with $|B| \leq 3k + 3$ and $|S| \leq k + 1$. Concerning the depth, let $(D_1, \ldots, D_m)$ be a path in $T(M)$. On this path, normal and interaction nodes alternate and we focus only on the normal nodes. Because of the size lemma, if there are several large descriptors in a row, each time the bag size decreases by one, so after at

most $k+1$ steps there must be a small descriptor. Then again by the size lemma, the size of the next $V(D_{i+1})$ is at most half the size of $V(D_i)$ plus some constant. We conclude that the length of the path can be at most logarithmic. □

Lemma 3.13. *For every $k \geq 1$, there is a logspace DTM that on input of any graph G with $\mathrm{tw}(G) \leq k$ outputs $M(G)$.*

Proof. Since the size of the vertex set of $M(G)$ is polynomially bounded, all we need to show is that a logspace DTM can compute the set of child descriptors of a descriptor D. For this, it finds the separator S by testing for each possible set S of size $k+1$ whether it separates the correct set in $G(D)$. For this, the machine uses Reingold's algorithm [36] to determine the components into which S separates $G(D)$. The machine then picks one such S and then determines, again using Reingold's algorithm, the sets B_i and outputs the desired child descriptors. □

3.3 Computing Balanced Binary Tree Decompositions.

Our objective is to show that we can turn any tree decomposition of logarithmic depth into a balanced binary tree decomposition via an appropriate logspace DTM.

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 iff 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 of T' a *white* node if there is no node n in T with $\iota(n) = w$. An example of an embedding is shown below, 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.
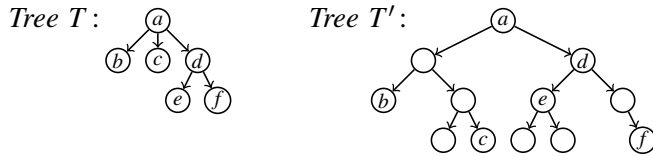
Given a tree T, our objective is to reduce the degree of T's nodes by computing, in logarithmic space, a binary tree T' together with an embedding of T into T'. It is easy to compute *some* binary tree T' into which we can embed T by replacing high-degree nodes by little binary trees, but if T has height h, the resulting tree T' may have height up to $h \log_2 n$. In particular, this transformation will transform a tree of height $O(\log n)$ into a tree of height $O(\log^2 n)$. The theorem below shows that a more careful balancing of high-degree nodes allows us to do much better; we expect that this approach is not completely new, but could not find it in the literature.

Theorem 3.14. *There is a logspace DTM that on input of any tree T with n vertices and height h outputs a binary tree T' of height at most $O(h + \log n)$ together with an embedding $\iota \colon V(T) \to V(T')$.*

Proof. It suffices to describe how T can be embedded into a tree T' of height $O(h + \log n)$ such that each node of T' has degree at most 3. Clearly, if we can achieve this in logarithmic space, we can also embed T into a binary tree by splitting every node of T' of degree 3 and adding white children to nodes of degree 1. This will at most double the height of T'.

On input T, the machine treats every node $v \in V(T)$ independently and only considers the nodes v of degree 4 or more. Given such a node v, let $u_1, \ldots, u_k$ be its children in T. Let n_i denote the number of leafs in the subtree of T rooted at u_i. For each v the machine computes (see below) a ternary tree S_v such that v is the root of S_v and the children of v in T are exactly the leafs of S_v and all

other nodes are white nodes. In the resulting tree T' we then remove the edges from v to its children and add the new (white) nodes of the tree S_v together with its edges.

The tree S_v is constructed as follows. Each node u of S_v will be labeled by an interval $[i,j] \subseteq \{1,\ldots,k\}$, denoted I_u. Such an interval tells us which of v's children in T are present in the subtree rooted at u in S_v. In particular, the root v of S_u is labeled with the interval $I_v = [1,k]$ and each leaf u_i is labeled with the singleton interval $I_{u_i} = [i,i]$. For each internal node u of S_v its children are labeled with intervals that form a partition of I_u. We next describe how these partitions are chosen.

If $I_u = [i,j]$ consists of at most three elements, we can trivially partition it so that the children are all leafs. Otherwise, let I_u contain four or more indices. For an interval $I \subseteq \{1,\ldots,k\}$, let n_I denote the sum $\sum_{i \in I} n_i$. We choose a number $s \in I_u$ according to the following rule: If there is some $s \in I_u$ with $n_s > n_{I_u}/2$, choose this s; otherwise let $s \in I_u$ be minimal such that $n_{[i,s]} > n_{I_u}/2$, but $n_{[i,s-1]} \leq n_{I_u}/2$. Let $J_1 = [i, s-1]$ and $J_2 = [s+1, j]$ and let u have three children with the attached intervals J_1, $[s,s]$, and J_2. Observe that $n_{J_1} \leq n_{I_u}/2$ and also $n_{J_2} \leq n_{I_u}/2$. An example of how the partitioning works is shown below:

v and its children   *Tree S_v*
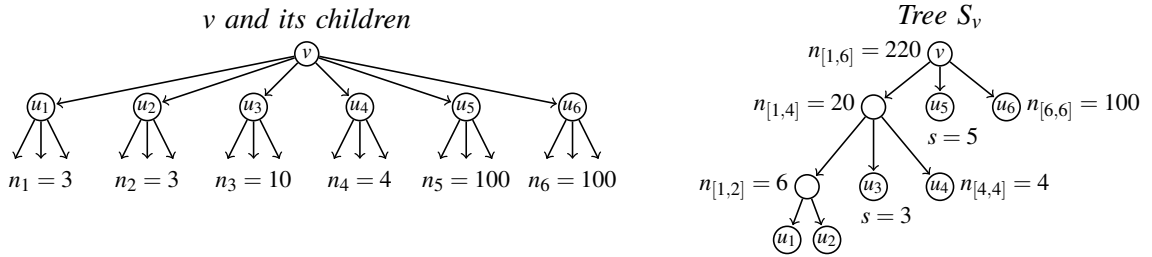
It remains to prove two properties: First, we need to show that the trees S_v can all be computed in logspace and, second, we need to show that the height of the resulting T' is at most $O(h + \log n)$.

For the first claim, we use Lemma 3.2 to compute S_v. The following graph M is a mangrove: Its vertices are all non-empty intervals $[i,j] \subseteq \{1,\ldots,k\}$. For non-singleton intervals there are edges exactly to the two or three intervals that make up the partition of the interval as described above. Note that the set of vertices reachable from the interval $[1,k]$ in M is exactly the desired tree S_v. A transitive closure R of M's related vertices is given by the graph in which there is an edge from every interval I to every interval J with $J \subsetneq I$. By Lemma 3.2 we can then compute the set of nodes reachable in M from $[1,k]$ in logarithmic space.

For the second property, consider any root-to-leaf path of T' and consider the sequence $(I_1, \ldots, I_k)$ of intervals attached to the nodes on this path. Then $n_{I_1} \geq \cdots \geq n_{I_k}$. Furthermore, suppose there are two white nodes in a row on this path and let I_i and I_{i+1} be the intervals attached to these nodes. We claim that $n_{I_{i+1}} \leq n_{I_i}/2$: Since the second node is white, it must be one of the intervals J_1 or J_2 created in a partitioning step. However, both of these intervals have size at most $n_{I_i}/2$. This shows that there can be at most $\log_2 n$ white nodes on the path whose predecessor is also white. Since there are at most h non-white nodes on the path, we get a height of at most $2h + \lceil \log_2 n \rceil$. □

Corollary 3.15. *For every c, there is a logspace DTM that on input of any tree T with n vertices and height at most $c \cdot \log_2 n$ outputs a balanced binary tree T' together with an embedding $\iota : V(T) \to V(T')$.*

Proof. By Theorem 3.14, on input T we can compute, in logarithmic space, an embedding into a binary tree T' of height at most $h := (2c+1)\lceil \log_2 n \rceil$. We can then embed T' into a balanced binary tree T' of height exactly h: For every leaf l of T' that is at a height $h_l < h$, add new white nodes to form a balanced binary tree of height $h - h_l$ and root l. Computing this second embedding can clearly be done in logspace. □

Lemma 3.16. *There is a logspace DTM that on input of any logarithmic depth tree decomposition of a graph G outputs a balanced, binary tree decomposition of G of the same width.*

Proof. Given a tree decomposition (T,B), the machine applies Corollary 3.15 to T, yielding a balanced binary tree T' and an embedding $\iota: V(T) \to V(T')$. We label the nodes $n \in V(T')$ with bags $B'(n)$ as follows: Find the first non-white node n' on the path from n to the root and let $v \in V(T)$ be the node with $\iota(v) = n'$. Then $B'(n) = B(v)$. The resulting labeled tree T' is clearly a tree decomposition of $\mathcal{A}$ of the same width as T. $\qquad\square$

4 Logspace Cardinality Version of Courcelle's Theorem

In this section we prove Theorem 1.3, the logspace cardinality version of Courcelle's Theorem. We follow a classical method of proving Courcelle's Theorem: Starting with a tree decomposition of a structure $\mathcal{A}$ and a formula ϕ, we construct a labeled tree $\mathcal{T}$ and a formula ψ such that $\mathcal{A} \models \phi$ iff $\mathcal{T} \models \psi$. Using standard arguments, one can then construct a tree automaton M that decides whether $\mathcal{T} \models \psi$ holds. The main new problem is showing how runs of the automaton can be used to determine the desired solution histograms. This is done by constructing special arithmetic trees that we call *convolution trees* and by showing that they can be evaluated in logarithmic space.

An *s-tree structure* is a $\tau_{s\text{-tree}}$-structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ over the vocabulary $\tau_{s\text{-tree}} = \{E^2, P_1^1, \ldots, P_s^1\}$ where $(V, E^{\mathcal{T}})$ is a tree. An *s*-tree structure is *binary* or *balanced*, if $(V, E^{\mathcal{T}})$ is binary or balanced, respectively.

Lemma 4.1. *Let $k \geq 1$ and let $\phi(X_1, \ldots, X_d)$ be an MSO-τ-formula. Then there are an $s \geq 1$, an MSO-$\tau_{s\text{-tree}}$-formula $\psi(X_1, \ldots, X_d)$, and a logspace DTM that on input of any τ-structure $\mathcal{A}$ with universe A and $\mathrm{tw}(\mathcal{A}) \leq k$ outputs a balanced binary s-tree structure $\mathcal{T}$ such that for all indices $i \in \{0, \ldots, |A|\}^d$ we have $\mathrm{histogram}(\mathcal{A}, \phi)[i] = \mathrm{histogram}(\mathcal{T}, \psi)[i]$.*

Proof. Our proof is a variant of the proof by Arnborg et al. [3] with three main differences: First, we work directly on arbitrary structures instead of graphs. Second, we argue that the transformations can be performed in logarithmic space. Third, we extend the reduction so that it preserves the solution histogram without using an additional dedicated node weight function.

Let $\phi(X_1, \ldots, X_d)$ be a fixed MSO-τ-formula and let $\mathcal{A}$ be a τ-structure of tree width at most k. We first explain how the *s*-tree structures $\mathcal{T}$ is constructed in logarithmic space. The machine first constructs a balanced binary tree decomposition T of $\mathcal{A}$ of width $k' = 4k + 3$ using Lemma 3.1. The machine then builds a preliminary *s*-tree structure $\mathcal{T}$, where s will be specified later. The preliminary *s*-tree structure will neither be binary nor balanced, but it will be easy to fix this later.

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*. The idea is that the bag nodes induce exactly the tree decomposition and to each bag node we attach a bunch of element nodes that tell us which elements of $\mathcal{A}$'s universe are present in the bag node. In detail, the set V_B is exactly $V(T)$. The set V_E is the disjoint union of the sets $\{a_1^n, \ldots, a_{r_n}^n\}$ for $n \in V(T)$ with attached bag $B(n) = \{a_1, \ldots, a_{r_n}\}$, where some ordering is chosen for each bag. For an element node $x = a_i^n$, we write $n(x)$ for the node $n \in V(T)$, we write $i(x)$ for the index i, and we write $a(x)$ for the element $a_i \in A$. The edges of $\mathcal{T}$ are as follows: All edges of the tree decomposition T are also present in $\mathcal{T}$. Additionally, for each $x \in V_E$ there is an edge from $n(x)$ to x.

Before we describe the s unary predicates that are present in $\mathcal{T}$, we first explain how a special coloring of the element nodes can be used to determine which element nodes represent the same element from $\mathcal{A}$'s universe. Let us call a coloring $C: V_E \to \{1, \ldots, 2k' + 2\}$ of the element nodes *valid* if for every two element nodes x and y for which $n(x)$ and $n(y)$ are identical or adjacent in T, we have $C(x) = C(y)$ iff $a(x) = a(y)$. (Note that element nodes x and y with non-adjacent $n(x)$ and $n(y)$ can have the same color even though $a(x) \neq a(y)$.) We make some observations concerning valid colorings:

1. A valid coloring can be obtained as follows: Assign up to $k'+1$ colors to the elements of the root bag. Then, given a bag node n whose element nodes are already colored and a child bag node n', the colors of the element nodes of n' are determined as follows: Given an element node x with $n(x) = n'$, if there is an element node y with $n(y) = n$ and $a(x) = a(y)$, assign the color $C(y)$ to x. Otherwise, choose a new color for x. We will not run out of colors in this process, since every bag contains at most $k+1$ elements.

2. A valid coloring can be computed in logarithmic space: To determine the color of an element node x, the algorithm can walk down from the root to $n(x)$, just keeping track of the colors of the elements of the current node's bag.

3. For each color $c \in \{1, \ldots, 2k'+2\}$, consider the set $V_c = \{n(x) \mid C(x) = c, x \in V_E\}$ and the connected components of $\mathcal{T}[V_c]$. Then these components are in one-to-one correspondence to the universe of $\mathcal{A}$.

4. There is an MSO-$\tau_{s\text{-tree}}$-formula $\psi_{\text{equ}}(x, y)$ that is true if x and y are element nodes for which $n(x)$ and $n(y)$ are in the same of the above components. This formula quantifies over the nodes on the path between the two nodes.

For every color c and every component of $\mathcal{T}[V_c]$ we choose exactly one *representative* element node. An example of the construction of $\mathcal{T}$ for the tree decomposition $\boxed{\{a,b\}} \leftarrow \boxed{\{b,c\}} \rightarrow \boxed{\{c,d\}}$ is shown right. The element nodes are the small nodes, whose colors encode the bags locally. Possible representative nodes of the elements of the universe $\{a, b, c, d\}$ are indicated.
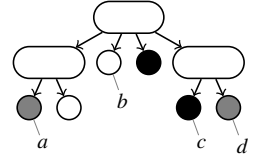
We can now describe the s unary predicates that are present in $\mathcal{T}$. First, there is a set $P_B^{\mathcal{T}}$ that is true for a node of $\mathcal{T}$ iff it is a bag node. Second, there are $2k'+2$ sets $P_1^{\mathcal{T}}, \ldots, P_{2k'+2}^{\mathcal{T}}$ that encode the coloring of the element nodes: $x \in P_c^{\mathcal{T}}$ iff $C(x) = c$. Third, there is a predicate $P_R^{\mathcal{T}}$ that singles out the representative element nodes. Note that there is a one-to-one correspondence between $P_R^{\mathcal{T}}$ and the universe of $\mathcal{A}$. This will allow us to obtain a histogram preserving reduction by only allowing nodes in $P_R^{\mathcal{T}}$ to be contained in the sets that are used for the free relation variables of ψ.

To represent a relation $R^{\mathcal{A}}$ of arity r of $\mathcal{A}$, we introduce new predicates $P_{i_1, \ldots, i_r, j}^{\mathcal{T}}$ for every $j \in \{1, \ldots, r\}$ and $i_j \in \{1, \ldots, k'+1\}$. They locally encode the tuples of $R^{\mathcal{A}}$ at the bags with $(i_1, \ldots, i_r)$ being the local indices of the element of a tuple of $R^{\mathcal{T}}$ and j being the position of an element in the tuple. In detail, for every tuple $(x_1, \ldots, x_r) \in V_E^r$ with $n(x_1) = \cdots = n(x_r)$ and $(a(x_1), \ldots, a(x_r)) \in R^{\mathcal{A}}$, let $x_j \in P_{i(x_1), \ldots, i(x_r), j}^{\mathcal{T}}$ for $j \in \{1, \ldots, r\}$. We make the following observations:

1. Since a tree decomposition puts the elements of a tuple completely into at least one bag, for all tuples $(a_1, \ldots, a_r) \in R^{\mathcal{A}}$ there are element nodes $x_1, \ldots, x_r$ with $n(x_1) = \cdots = n(x_r)$ and $x_1 \in P_{i(x_1), \ldots, i(x_r), 1}^{\mathcal{T}}, \ldots, x_r \in P_{i(x_1), \ldots, i(x_r), r}^{\mathcal{T}}$.

2. There is an 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 iff $(a(x_1), \ldots, a(x_r)) \in R^{\mathcal{A}}$: 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 $P_{i_1, \ldots, i_r, 1}(y_1) \wedge \cdots \wedge P_{i_1, \ldots, i_r, r}(y_r)$.

3. The $P_{i_1, \ldots, i_r, j}^{\mathcal{T}}$ can be constructed in logarithmic space.

The structure $\mathcal{T}$ is neither balanced nor binary. However, this can easily be fixed: Just as in the proof of Lemma 3.16 we introduce white nodes and replace high-degree nodes by small binary trees and fill up the tree so that it becomes balanced. However, the argument is actually simpler than in the proof of Lemma 3.16 since we start with a tree of bounded degree. We introduce a predicate $P_W^{\mathcal{T}}$, where $\mathcal{T}$ is now the resulting balanced, binary tree, that is true exactly for the introduced white nodes. The formulas ψ_{equ} and ψ_R for $R \in \tau$ can then easily be adjusted so that they take the presence of white nodes into account.

This concludes the construction of the s-tree structure $\mathcal{T}$, where s is chosen such that all of the $P_i^{\mathcal{T}}$ described above are accounted for. The tree can, as claimed, be constructed in logarithmic space.

It remains to explain how we obtain the formula ψ from ϕ. This can be achieved via the following transformation: First, extend the formula ϕ such that only elements and subsets of $P_R^{\mathcal{T}}$ 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 $\psi_R(x_1,\ldots,x_r)$. $\qquad\square$

The next step is to establish a relation between s-tree structures and tree automata. For this, given a binary s-tree structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ and sets $S_1,\ldots,S_d \subseteq V$ let us write $T(\mathcal{T},S_1,\ldots,S_d)$ for the tree $(V,E^{\mathcal{T}})$ where we label each node $v \in V$ with the bitstring $l_1 \ldots l_s x_1 \ldots x_d \in \{0,1\}^{s+d}$ with $l_i = 1 \iff v \in P_i^{\mathcal{T}}$ and $x_i = 1 \iff v \in S_i$. An example is shown right for the 1-tree structure $\mathcal{T}$ with $V = \{1,2,3,4,5\}$, $E^{\mathcal{T}} = \{(1,2),(1,3),(2,4),(2,5)\}$, $P_1^{\mathcal{T}} = \{1,3,5\}$, $S_1 = \emptyset$, and $S_2 = \{1,4\}$. We write $T(\mathcal{T})$ in case $d = 0$. A tree automaton can work on $T(\mathcal{T},S_1,\ldots,S_d)$ if we choose a distinguished left child for each inner node. Different versions of the following fact are used in many versions of Courcelle's Theorem, the below formulation is implicitly shown in [3]. To prove it, one inductively constructs tree automata for subformulas and combines them to automata for composed formulas.
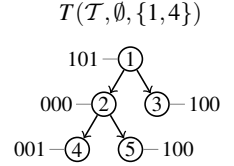
Fact 4.2. *For every $s \geq 0$ and every MSO-$\tau_{s\text{-tree}}$-formula $\phi(X_1,\ldots,X_d)$, there is a tree automaton M with alphabet $\Sigma = \{0,1\}^{s+d}$ such that for all binary s-tree structures $\mathcal{T}$ and all subsets $S_1,\ldots,S_d \subseteq V$ we have $\mathcal{T} \models \phi(S_1,\ldots,S_d)$ iff M accepts $T(\mathcal{T},S_1,\ldots,S_d)$.*

In view of Fact 4.2 and Lemma 4.1, we need to determine how many sets $S_1,\ldots,S_d \subseteq V$ of certain sizes make M accept $T(\mathcal{T},S_1,\ldots,S_d)$. We reduce this problem to evaluating *convolution trees*. In the following, an $[r]^d$-*array* is an d-dimensional array of non-negative integers where all indices $i = (i_1,\ldots,i_d)$ are elements of the index set $[r]^d = \{0,\ldots,r-1\}^d$. We call r the *range*. The *addition* of two arrays is defined component-wise in the obvious way. The *convolution* of an $[r]^d$-array A and an $[s]^d$-array B is the $[r+s-1]^d$-array $C = A * B$ defined by $C[i] = \sum_{j \in [r]^d, k \in [s]^d, j+k=i} A[j] \cdot B[k]$. A *convolution tree* T is a tree whose leafs are labeled by arrays of appropriate sizes and whose inner nodes are labeled by $+$ or $*$. Its *value* $\text{val}(T)$ is the array resulting from recursively applying the operation in each node to the values of the child trees. In the right example the (2×2)-matrices at the leafs are $[2]^2$-arrays; position $(0,0)$ lies at the upper left corner.
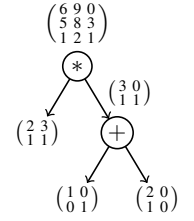
Lemma 4.3. *Let M be a tree automaton with alphabet $\{0,1\}^{s+d}$. Then there exists a logspace DTM that maps every binary balanced s-tree structure $\mathcal{T}$ to a convolution tree T where $\text{val}(T)[i_1,\ldots,i_d]$ is exactly the number of sets $S_1,\ldots,S_d \subseteq V$ with $|S_j| = i_j$ for which M accepts $T(\mathcal{T},S_1,\ldots,S_d)$.*

*Proof.* For this proof, we introduce the following terminology: Given a set A, a *multicoloring* of A is a tuple $(A_1,\ldots,A_d)$ of subsets $A_j \subseteq A$ for $j \in \{1,\ldots,d\}$. Given two disjoint sets A and B and sets of multicolorings X and Y of A and B, respectively, we write $X \otimes Y$ for the set of multicolorings $\{(A_1 \cup B_1,\ldots,A_d \cup B_d) \mid (A_1,\ldots,A_d) \in X, (B_1,\ldots,B_d) \in Y\}$. Given a set X of multicolorings of A, let $\text{histogram}(X)$ denote the $[|A|+1]^d$-array whose entry at index $i = (i_1,\ldots,i_d)$ is the number of multicolorings $(A_1,\ldots,A_d) \in X$ with $|A_1| = i_1$, $\ldots$, $|A_d| = i_d$. We make two simple, but useful observations: First, given two disjoint sets X_1 and X_2 of multicolorings of the same set A, we have $\text{histogram}(X_1 \cup X_2) = \text{histogram}(X_1) + \text{histogram}(X_2)$. Second, given two disjoint sets A and B and sets of multicolorings X and Y of A and B, respectively, then $\text{histogram}(X \otimes Y) = \text{histogram}(X) * \text{histogram}(Y)$.

Let $\mathcal{T}$ be an s-tree structure with node set V. To simplify the induction later on, we also allow the "empty" s-tree structure $\mathcal{T}$ with $V = \emptyset$. The desired convolution tree T has an $+$-node at the root whose children are the trees $C(\mathcal{T},q)$, to be defined in a moment, for $q \in Q_a$. For each state q of M, let $S(\mathcal{T},q)$ be the set of multicolorings $(S_1,\ldots,S_d)$ of V for which M assigns the state

15

q to $T(\mathcal{T}, S_1, \ldots, S_d)$. Provided we can construct the convolution trees $C(\mathcal{T}, q)$ in such a way that $\mathrm{val}\big(C(\mathcal{T}, q)\big) = \mathrm{histogram}\big(S(\mathcal{T}, q)\big)$, then $\mathrm{val}(T) = \sum_{q \in Q_a} \mathrm{histogram}\big(S(\mathcal{T}, q)\big)$ is, indeed, the correct value. It remains to explain how the $C(\mathcal{T}, q)$ are defined. We give an inductive definition in the following.

First, let $\mathcal{T}$ be empty. Then $V = \emptyset$ and $[|V| + 1]^d = \{(0, \ldots, 0)\}$. Let $C(\mathcal{T}, q)$ be a single leaf whose attached array is a single entry, which we set to 1 for $q = q_0$ and to 0 otherwise. Then the value of this tree is, indeed, $\mathrm{histogram}\big(S(\mathcal{T}, q)\big)$ since the automaton assigns q_0 to the empty tree and all S_i can only be empty.

Second, let $\mathcal{T}$ have root r and let l be the label of the root in $T(\mathcal{T})$. Let $\mathcal{T}_{\mathrm{left}}$ and $\mathcal{T}_{\mathrm{right}}$ be the left and right subtrees and let V_{left} and V_{right} be their node sets, respectively. Then the root of the tree $C(\mathcal{T}, q)$ is a $+$-node that has one child node for each triple $(q_{\mathrm{left}}, q_{\mathrm{right}}, x) \in Q \times Q \times \{0, 1\}^d$ with $\delta(q_{\mathrm{left}}, q_{\mathrm{right}}, lx) = q$. Each child node is a $*$-node (a convolution node), which has three children: One child is $C(\mathcal{T}_{\mathrm{left}}, q_{\mathrm{left}})$, one child is $C(\mathcal{T}_{\mathrm{right}}, q_{\mathrm{right}})$, and one child is a leaf to which we attach the histogram of the singleton set $I_x = \{(A_1, \ldots, A_d)\}$, where $(A_1, \ldots, A_d)$ is the multicoloring of $A = \{r\}$ where for every $j \in \{1, \ldots, d\}$ we have $A_j = \{r\}$ if the jth bit of x is 1 and $A_j = \emptyset$, otherwise. An examples of this construction is shown below.
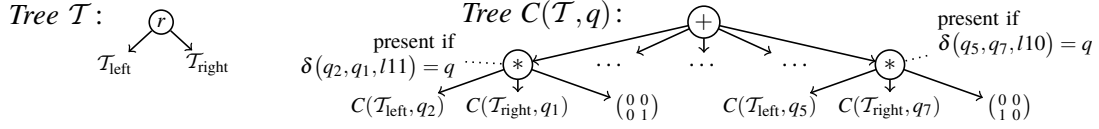
It remains to show that $\mathrm{val}\big(C(\mathcal{T}, q)\big) = \mathrm{histogram}\big(S(\mathcal{T}, q)\big)$ holds. For a triple $(q_{\mathrm{left}}, q_{\mathrm{right}}, x) \in Q \times Q \times \{0, 1\}^d$, let $S(q_{\mathrm{left}}, q_{\mathrm{right}}, x)$ be the set of all multicolorings $(S_1, \ldots, S_d) \in S(\mathcal{T}, q)$ of V such that

1. M assigns q_{left} to $T_{\mathrm{left}}(S_1 \cap V_{\mathrm{left}}, \ldots, S_d \cap V_{\mathrm{left}})$,
2. M assigns q_{right} to $T_{\mathrm{right}}(S_1 \cap V_{\mathrm{right}}, \ldots, S_d \cap V_{\mathrm{right}})$, and
3. the jth bit of x is 1 iff $r \in S_j$.

Then $S(\mathcal{T}, q)$ is the disjoint union of the $S(q_{\mathrm{left}}, q_{\mathrm{right}}, x)$ with $\delta(q_{\mathrm{left}}, q_{\mathrm{right}}, lx) = q$. Thus, the histogram of $S(\mathcal{T}, q)$ is the sum of the histograms of the $S(q_{\mathrm{left}}, q_{\mathrm{right}}, x)$. Next, by the definition of $S(q_{\mathrm{left}}, q_{\mathrm{right}}, x)$, it can be expressed as $S(T_{\mathrm{left}}, q_{\mathrm{left}}) \otimes S(T_{\mathrm{right}}, q_{\mathrm{right}}) \otimes I_x$. This implies that the histogram of $S(q_{\mathrm{left}}, q_{\mathrm{right}}, x)$ is the convolution of the histograms of $S(T_{\mathrm{left}}, q_{\mathrm{left}})$, $S(T_{\mathrm{right}}, q_{\mathrm{right}})$, and I_x.

The tree T can be computed recursively starting from the root of a given a balanced binary s-tree structure. For every node we only have to store a constant number of bits that depend on the transition function of M. Together with the fact that the input tree has logarithmic depth, the overall stack space is bounded by a function of M times the logarithm of the input size. $\qquad \square$

All that remains to be done is to evaluate convolution trees in logarithmic space. For this we introduce a bit of terminology. Let N and R be two fixed bases, both to be chosen later. Given an $[r]^d$-array A, let $\mathrm{num}(A) = \sum_{i \in [r]^d} A[i] N^{i_1 + i_2 R + \cdots + i_d R^{d-1}}$. Trivially, $\mathrm{num}(A + B) = \mathrm{num}(A) + \mathrm{num}(B)$. For an $[r]^d$-array A and an $[s]^d$-array B we have $\mathrm{num}(A * B) = \mathrm{num}(A) \cdot \mathrm{num}(B)$, because

$$
\begin{aligned}
\mathrm{num}(A) \cdot \mathrm{num}(B) &= \left(\sum_{j \in [r]^d} A[j] N^{j_1 + j_2 R + \cdots + j_d R^{d-1}} \right) \left(\sum_{k \in [s]^d} B[k] N^{k_1 + k_2 R + \cdots + k_d R^{d-1}} \right) \\
&= \sum_{j \in [r]^d, k \in [s]^d} A[j] B[k] N^{(j_1 + k_1) + (j_2 + k_2) R + \cdots + (j_d + k_d) R^{d-1}} \\
&= \sum_{i \in [r+s-1]^d} \left(\sum_{j \in [r]^d, k \in [s]^d, j+k=i} A[j] B[k] \right) N^{i_1 + i_2 R + \cdots + i_d R^{d-1}} = \mathrm{num}(A * B).
\end{aligned}
$$

Lemma 4.4. *There exists a logspace DTM that on input of any convolution tree outputs its value.*

Proof. For a convolution tree T let R be the sum of all ranges of leaf arrays. Then $\mathrm{val}(T)$ will be an $[r]^d$-array for some $r \leq R$. Let n denote the number of T's nodes, let m denote 1 plus the maximum

value of any integer present in any of its leaf arrays, let M denote 1 plus the maximum integer present in the array $\text{val}(T)$. We claim that $M \leq (R^d m)^n$ holds, which can be seen by structural induction: If T is a leaf, the claim is obviously correct. If T's root is a $+$-node with child trees T_1 and T_2, then $M \leq M_1 + M_2$ and, by the induction hypothesis, this can be bounded by $(R_1^d m_1)^{n_1} + (R_2^d m_2)^{n_2} \leq (R^d m)^{n_1} + (R^d m)^{n_2} \leq (R^d m)^{n_1+n_2} \leq (R^d m)^n$. If T's root is a convolution node, we have $M \leq R^d \cdot M_1 \cdot M_2$ and, by the induction hypothesis, this is bounded by $R^d (R_1^d m_1)^{n_1} (R_2^d m_2)^{n_2} \leq (R^d m)^{n_1+n_2+1} = (R^d m)^n$.

On input of a convolution tree T the machine first computes the length p of the binary representation of $(R^d m)^n$. Then $p \leq \log_2 (R^d m)^n + 1 = nd \log_2 R + n \log_2 m + 1$ is polynomial in the length of any reasonable encoding of T and p bits will suffice to encode any number encountered during an evaluation of the tree T. In particular, setting $N = 2^{p+1}$, for all $i \in [r]^d$ the integer $\text{val}(T)[i_1, \ldots, i_d]$ can be found at the p bits prior to position $i_1 + i_2 R + \cdots + i_d R^{d-1}$ from the right (least-significant) end of the binary representation of $\text{num}(\text{val}(T))$. As an example, for $N = 2^4$ and $R = 3$, the entry 3 at index $(0,1)$ of the $[2]^2$-array $A = \left(\begin{smallmatrix} 1 & 3 \\ 2 & 4 \end{smallmatrix}\right)$ can be found at the underlined bits of $\text{num}(A) = 1N^{0 \cdot R^0 + 0 \cdot R^1} + 2N^{1 \cdot R^0 + 0 \cdot R^1} + 3N^{0 \cdot R^0 + 1 \cdot R^1} + 4N^{1 \cdot R^0 + 1 \cdot R^1} = 274465 = 0100\underline{0011}000000100001_2$.

The machine maps the convolution tree T to an arithmetic tree T' whose leafs are labeled with ordinary integers by replacing every leaf array A by $\text{num}(A)$ and every convolution node by a multiplication node. Then, since $\text{num}(A + B) = \text{num}(A) + \text{num}(B)$ and $\text{num}(A * B) = \text{num}(A) \cdot \text{num}(B)$, we immediately get that the value of the resulting ordinary arithmetic $\{+, \times\}$-tree is $\text{num}(\text{val}(T))$. Since evaluating $\{+, \times\}$-trees can be done in logarithmic space [5, 12, 14, 25], we get the claim. □

5 Logspace Version of Bodlaender's Theorem

In this last section we prove Theorem 1.1, the logspace version of Bodlaender's Theorem. For the proof, we first need to show that TREE-WIDTH-$k \in L$ holds for all k. Then, we reapply the ideas from Section 3, only we make sure that in the constructed descriptor decomposition M the implicit tree decomposition $T(M)$ has width k. The first step toward proving Theorem 1.1 is thus proving Lemma 1.4.

Proof of Lemma 1.4. First, we need to show that TREE-WIDTH-$k \in L$ holds for all k. It is well known [38, 21] that for every k the graph property "the graph has tree width k" can be characterized by a finite set of forbidden minors. Furthermore, it is also well known [3], that the question of whether a graph contains a given minor is expressible in MSO-logic. This allows us to decide TREE-WIDTH-k as follows: On input of a graph G, use Lemma 3.1 to obtain a tentative tree decomposition T of G. Then, test whether the output is, indeed, a tree decomposition of G. This test can easily be performed in logarithmic space. If T is not a tree decomposition of G of width at most $4k + 3$, we know that $\text{tw}(G) > k$ and reject; which is correct since the algorithm from Lemma 3.1 will always output a valid tree decomposition in case $\text{tw}(G) \leq k$. We can now apply Theorem 1.2 to G for the MSO-formula ϕ that characterizes tree width k. Since, internally, Theorem 1.2 makes use of the tree decomposition from Lemma 3.1, we can conclude that Theorem 1.2 will correctly decide whether $G \models \phi$ holds.

Second, we need to show that for every $k \geq 1$ the problem TREE-WIDTH-k is hard for L. For a fixed $k \geq 1$, we reduce the well-known L-complete problem ACYCLICITY for undirected graphs to TREE-WIDTH-k via the following first-order reduction: On input of an undirected graph G with $V(G) = \{v_1, \ldots, v_n\}$, build a new undirected graph G' as follows: For each vertex $v_i \in V(G)$ the set $V(G')$ contains k vertices $v_i^1, \ldots, v_i^k$. In the edge set $E(G')$ they are connected so that they form a k-clique. Next, for each edge $(v_i, v_j) \in E(G)$ with $i < j$, the following edges are present in $E(G')$: For all $p \in \{1, \ldots, k\}$ and $q \in \{p, \ldots, k\}$ there is an edge between v_i^q and v_j^p. As an example, the graph $\textcircled{v_1}\!-\!\textcircled{v_2}\,\textcircled{v_4}\!-\!\textcircled{v_3}$ is mapped to the graph shown right for $k = 3$.
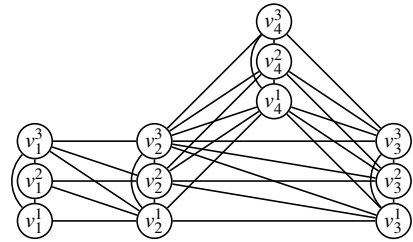
We claim that G is acyclic iff G' has tree width k. To prove this, first assume that G is acyclic. Then each component of G' will be a k-tree and, thus, G will have tree width k. To see this, note that a tree decomposition of a component of G' can be obtained from G as follows: Attach the bag $\{v^1, \ldots, v^k\}$ to each vertex v of G. Then, for each edge (v_i, v_j) of G with $i < j$, replace the edge by a path of length k and attach the following bags to the new vertices: $\{v_i^1, \ldots, v_i^k, v_j^1\}$, $\{v_i^2, \ldots, v_i^k, v_j^1, v_j^2\}$, $\ldots$, $\{v_i^k, v_j^1, \ldots, v_j^k\}$. The resulting graph is clearly still acyclic, its bags cover all edges of G', and the subgraphs of all vertices whose bags contain a given vertex are connected.

Second assume that G contains a cycle. We show that G' contains a $(k+2)$-clique as a minor and, since the tree width of a graph does not increase by taking minors [21] and for every tree decomposition each clique is completely contained in at least one bag, this implies $\mathrm{tw}(G') > k$. Let $(v_1, \ldots, v_r, v_1)$ be a cycle of G. In the subgraph $G'[\{v_1^1, \ldots, v_1^k\} \cup \cdots \cup \{v_r^1, \ldots, v_r^k\}]$, for each $j \in \{2, \ldots, r\}$ merge the clique $\{v_j^1, \ldots, v_j^k\}$ to a single node v_j'. In the resulting minor both nodes v_2' and v_r' have edges to all vertices of the clique $\{v_1^1, \ldots, v_1^k\}$. By merging the path $(v_2', \ldots, v_r')$ into the edge (v_2', v_r'), we obtain the $(k+2)$-clique $\{v_1^1, \ldots, v_1^k, v_2', v_r'\}$. $\qquad\square$

Proof of Theorem 1.1. We describe how to construct a tree decomposition of width k for a graph $G = (V, E)$ with $\mathrm{tw}(G) \le k$ in logarithmic space. Recall that Lemma 3.6 states that on input of a graph G together with a descriptor decomposition M of G, we can compute a tree decomposition of G in logarithmic space. Thus, in the following it suffices to explain how a descriptor decomposition M of G can be obtained in logarithmic space for which $T(M)$ has width k. Clearly, M has to be different from the one constructed in Section 3 since we can no longer allow bags larger than $k+1$. As we define the new descriptor decomposition M below, we also explain how M can be constructed via some logspace DTM.

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

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

The vertex set $V(M)$ of descriptors contains D_G and all descriptors D with $|B(D)| = k+1$ such that there is a tree decomposition T of $G(D)$ in which $B(D)$ is attached to some bag of T. Given a descriptor D, we can test whether it is an element of $V(M)$ in logarithmic space as follows: We apply Lemma 1.4 to the graph $G(D) \cup K_{B(D)}$ and include D if this graph has tree width at most k. Observe that, indeed, if there is a tree decomposition T of $G(D)$ in which $B(D)$ is attached to some bag of T, then this tree decomposition is also a tree decomposition of the same width of $G(D) \cup K_{B(D)}$. The other way round, in every tree decomposition of $G(D) \cup K_{B(D)}$ there must be a node whose bag contains the clique $K_{B(D)}$ (this is a fundamental property of tree decompositions).

To define the edge set $E(M)$, we explain, in the same spirit as in Definition 3.8, which descriptors are the *child descriptors* of a given descriptor $D \in V(M)$. If D is simple, it has no child descriptors. Otherwise, we search for a bag $B' \subseteq V(D)$ with the following properties:

1. $|B'| = k+1$ and $B' \cap I(D) \ne \emptyset$.
2. There is no edge between a vertex in $B(D) - B'$ and a vertex in $I(D)$.
3. Let $C_1, \ldots, C_m$ be the components of the graph $G[I(D) - B']$. Then for each $i \in \{1, \ldots, m\}$ the graph $G[V(C_i) \cup B'] \cup K_{B'}$ must have tree width at most k.

Clearly, if B' with the above properties exists, we can find it in logarithmic space by iterating over all possible B' and each time invoking Lemma 1.4 on $G[V(C_i) \cup B'] \cup K_{B'}$. We choose one such B' and for every components C_i of $G[I(D) - B']$ we choose a vertex $v_i \in C_i$ and let all (B', v_i) be child descriptors of D; additionally, the simple descriptor B' is a child descriptor of D.

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

It remains to argue that the resulting graph M is a descriptor decomposition and that the tree width of $T(M)$ is $k+1$. To see that M is a descriptor decomposition, first note that D_G is an element of $V(M)$. Concerning the four properties of a descriptor decomposition, properties 1 to 3 follow for exactly the same reasons as in Lemma 3.11. For property 4, we can account for all edges as follows: Edges inside the C_i and between C_i and B' are covered by the graphs $G(D_i)$. Edges inside B' are covered by the simple descriptor B'. Edges inside B need not be covered. Edges between $B - B'$ and $I(D)$ do not exist and $B' - B \subseteq I(D)$.

To prove that the width of $T(M)$ is $k+1$, first note that all bags $B(D)$ attached to normal vertices of $T(M)$ have size at most $k+1$ by construction. Second, note that the bag attached to an interaction vertex D^{interact} is a subset of B'. Hence, the size of interaction bags is at most $k+1$. $\qquad\square$

6 Conclusion

Like the classical theorems of Bodlaender and of Courcelle, their logspace versions are useful tools in classifying the complexity of problems on graphs of bounded tree width. We have sketched a number of applications; indeed our own proof of the logspace version of Bodlaender's Theorem makes heavy use of the logspace version of Courcelle's Theorem. We are confident that applications beyond the ones indicated will be found.

There are two intriguing open problems that we would like to point out. First, what is the complexity of the graph isomorphism problem on graphs of bounded tree width? Researches have steadily lowered the complexity bound from P [7] to TC^1 [24] and to LOGCFL [20]. While one bottleneck in the latest paper [20], namely the construction of tree decompositions in logspace, is removed by the present paper, it is still unclear whether the complexity can be lowered to L. One promising approach may be to first study the graph automorphism problem. Another approach is to use the fact that the information-rich solutions histograms are invariant under isomorphisms. They might thus serve, for some appropriate formula ϕ, as a method of canonization. Second, can one devise logspace algorithms for more general width parameters [26]? One example is *clique width,* whose defining decompositions, the *k-expressions*, can be approximated in polynomial time [34].

References

[1] E. Allender and K.-J. Lange. $\text{RUSPACE}(\log n) \subseteq \text{DSPACE}(\log^2 n / \log\log n)$. *Theory of Computing Systems* 31(5):539–550, 1998. Online at http://dx.doi.org/10.1007/s002240000102

[2] 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. Online at http://dx.doi.org/10.1137/0608024

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

[4] V. Arvind, B. Das, and J. Köbler. The space complexity of *k*-tree isomorphism. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC 2007)*, pp. 822–833. Springer, LNCS 4835, 2007.   Online at http://dx.doi.org/10.1007/978-3-540-77120-3_71

[5] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing* 21(1):54–58, 1992.   Online at http://dx.doi.org/10.1137/0221006

[6] H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In *Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1988)*, pp. 1–10. Springer, LNCS 344, 1989.   Online at http://dx.doi.org/10.1007/3-540-50728-0_32

[7] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial *k*-trees. *Journal of Algorithms* 11(4):631–643, Dec. 1990.
Online at http://dx.doi.org/10.1016/0196-6774(90)90013-5

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

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

[10] H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal* 51(3):255–269, 2008.
Online at http://dx.doi.org/10.1093/comjnl/bxm037

[11] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica* 7(1–6):555–581, June 1992.   Online at http://dx.doi.org/10.1007/BF01758777

[12] 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.
Online at http://dx.doi.org/10.1137/0221046

[13] N. Chandrasekharan and S. T. Hedetniemi. Fast parallel algorithms for tree decomposing and parsing partial *k*-trees. In *Proceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing*, pp. 283–292, 1988.

[14] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *Theoretical Informatics and Applications* 35(3):259–275, May–June 2001.   Online at http://dx.doi.org/10.1051/ita:2001119

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

[16] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control* 64(1–3):2–22, Jan.–Mar. 1985.   Online at http://dx.doi.org/10.1016/S0019-9958(85)80041-3

[17] 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*, pp. 193–242. Elsevier and MIT Press, 1990.

[18] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science* 109(1–2):49–82, Mar. 1993.
Online at http://dx.doi.org/10.1016/0304-3975(93)90064-Z

[19] 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)*, pp. 215–226. Schloss Dagstuhl LZI, LIPIcs 5, 2010.
Online at http://dx.doi.org/10.4230/LIPIcs.STACS.2010.2456

[20] B. Das, J. Torán, and F. Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, pp. 227–238. Schloss Dagstuhl LZI, LIPIcs 5, 2010.
Online at http://dx.doi.org/10.4230/LIPIcs.STACS.2010.2457

[21] R. Diestel. *Graph Theory*. Springer, July 2005.
Online at http://diestel-graph-theory.com/index.html

[22] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
Online at http://dx.doi.org/10.1007/3-540-29953-X

[23] G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL certificates. *Theoretical Computer Science* 270(1–2):761–777, Jan. 2002. Online at http://dx.doi.org/10.1016/S0304-3975(01)00108-6

[24] M. Grohe and O. Verbitsky. Testing graph isomorphism in parallel by playing a game. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, pp. 3–14. Springer, LNCS 4051, 2006.
Online at http://dx.doi.org/10.1007/11786986_2

[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, Dec. 2002. Online at http://dx.doi.org/10.1016/S0022-0000(02)00025-9

[26] P. Hliněný, S.-i. Oum, D. Seese, and G. Gottlob. Width parameters beyond tree-width and their applications. *The Computer Journal* 51(3):326–362, 2008.
Online at http://dx.doi.org/10.1093/comjnl/bxm052

[27] O. H. Ibarra, T. Jiang, B. Ravikumar, and J. H. Chang. On some languages in NC. In *VLSI Algorithms and Architectures, Proceedings of the 3rd Aegean Workshop on Computing*, pp. 64–73. Springer, LNCS 319, 1988. Online at http://dx.doi.org/10.1007/BFb0040374

[28] A. Jakoby and M. Liśkiewicz. Paths problems in symmetric logarithmic space. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, pp. 269–280. Springer, LNCS 2380, 2002. Online at http://dx.doi.org/10.1007/3-540-45465-9_24

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

[30] A. Jakoby and T. Tantau. Logspace algorithms for computing shortest and longest paths in series-parallel graphs. In *Proceedings of the 27th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2007)*, pp. 216–227. Springer, LNCS 4855, 2007. Online at http://dx.doi.org/10.1007/978-3-540-77050-3_18

[31] B. Jenner. Knapsack problems for NL. *Information Processing Letters* 54(3):169–174, May 1995.
Online at http://dx.doi.org/10.1016/0020-0190(95)00017-7

[32] J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS 1990)*, pp. 173–182 vol.1. IEEE Computer Society, 1990.

Online at http://dx.doi.org/10.1109/FSCS.1990.89536

[33] B. Monien. On a subclass of pseudopolynomial problems. In *Proceedings of the 9th Symposium on Mathematical Foundations of Computer Science 1980 (MFCS 1980)*, pp. 414–425, LNCS 88, 1980.

Online at http://dx.doi.org/10.1007/BFb0022521

[34] S.-i. Oum and P. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B* 96(4):514–528, July 2006.

Online at http://dx.doi.org/10.1016/j.jctb.2005.10.006

[35] B. A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pp. 221–228. ACM, 1992.

Online at http://dx.doi.org/10.1145/129712.129734

[36] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM* 55(4):1–24, Sept. 2008.

Online at http://dx.doi.org/10.1145/1391289.1391291

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

[38] N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B* 92(2):325–357, Nov. 2004.

Online at http://dx.doi.org/10.1016/j.jctb.2004.08.001

[39] E. Wanke. Bounded tree-width and LOGCFL. *Journal of Algorithms* 16(3):470–491, May 1994.

Online at http://dx.doi.org/10.1006/jagm.1994.1022